END TERM PROJECT REPORT

An Analytics Project:

PREDICTION OF APPLICANT GOING DEFAULT IN NEXT 12 MONTHS FROM NEW CREDIT CARD APPLICATION

Submitted by

Group 1 – Team_404

MS22S006	**Aditya Joshi**
MS22S001	**Samina Yasmin**
MS22S005	**Mayur Morey**
MS21A033	**L R Kavith**

Department of Management Studies,

Indian Institute of Technology, Madras

15th December, 2022

Acknowledgement

The successful completion of this project would not have been possible without the due help and enlightening support of many individuals. On the very outset of this project report, we like to extend our sincere and heartfelt gratitude to all the people who helped us in this endeavour. Without their active guidance, help, cooperation and encouragement, completion of this project would not have been possible.

We are extremely thankful to our Professor, Dr. Nandan Sudarsanam for giving us an opportunity to undertake the project and for guiding us through the process. We also express our sincere gratitude to the teaching assistant for the course, Mr. Harshal Bhalerao for patiently answering all our queries and helping us out with the project. We are deeply indebted to everyone who gave us their wholehearted support in making this project fortuitous.

Date: 15th December 2022

Table of Contents

Problem Statement

Given a credit card application, predict if the applicant will default in the next 12 months of the application.

The customer's demographic variables, finance/credit-based features, transactional data from external credit cards, and behavioral features are all provided as metadata to help with the forecast.

This project will enable a bank to improve their profits, provide optimal incentives and rewards, optimize customer tagging, and minimize credit risk.

Problem Background

Bank can issue two types of credit cards. The first one is a charge card. For this card, the balance needs to be repaid in full each month. The second type of card is the lending card for which customer can pay the balance over a period subject to the interest rate being charged.

One can apply for either of the two available card types. Banks must first approve the applicant before issuing the card to the individuals. The process through which the lender determines whether an applicant is creditworthy and should be granted a credit line is known as underwriting. Banks have access to both the information on application forms and the consumer bureau.

Bureau is a company that aggregates data on consumer borrowing and payments in order to determine a person's creditworthiness and impose a cap on the total amount of credit that may be extended to that person by creditors.

The dataset includes bureau and client application data with default tagging, so if a customer has cumulatively missed three payments across all outstanding trades, his default indicator is 1, otherwise 0. Data comprises of independent variables at time T0 and the individual's actual performance (Default/Non-Default) at time T12, which is 12 months later.

About Amex

Amex is short form American Express, a global services company that provides customers with access to products, insights and experiences that enrich lives and build successful business. It is an American multinational corporation specialized in payment card services headquartered at 200 Vesey Street in the Battery Park City neighbourhood of Lower Manhattan in New York City. The company was founded in 1850 and is one of the 30 components of the Dow Jones Industrial Average. It has 1.2 trillion worldwide billed business with a presence in 130 markets and a total of 114 million cards in force making it the world’s largest card issuer by purchase volume.

Industry Analysis

Credit Card Industry Trends

Although the use of credit cards, a mainstay of the US payments ecosystem, declined at the beginning of the pandemic, things began to turn around by the end of 2021. JPMorgan Chase

reported 19.8% growth over the two years ending in Q3 2021, while Wells Fargo's credit card point-of-sale volume increased 29.9% during the same time period.

Although early in the pandemic, people favoured debit over credit as they sought to reduce financial risk, borrowing is once again on the rise. Gains are anticipated to continue in 2022, prompting issuers and fintech's to adjust incentives and introduce new products.

Credit now accounts for almost one-third (36.3%) of the value of in-store retail transactions. Due to consumers' continued support of online shopping, online credit card usage will soon surpass $500 billion for the first time. However, due in part to customers' growing preference for debit, this payment method's percentage of card and digital retail transactions may somewhat decline.

Soon, an increase in ancillary expenditure will fuel fierce competition among issuers, prompting a move toward larger cardholder perks.

With pandemic restrictions loosening, customers are making more leisure-related purchases, particularly in the crucially important for credit card volume travel and entertainment (T&E). Amex's T&E billed business increased by 124% yearly in the third quarter of 2021, although it still lagged pre-pandemic levels.

Issuers will feel pressure to improve their offerings because of the rise in credit consumption. According to data from the New York Fed, more than 25% of US customers applied for a new credit card in the 12 months before October 2021, up from 15.7% in October 2020 and on par with pre-pandemic levels.

Incentives other than straightforward rewards will drive innovation in card programmes. Providers will sharpen advantages in new areas, such as payment flexibility, exclusive member experiences, and access to money management tools, while also broadening their areas of concentration.

53 million persons in the US do not have regular credit ratings, and more have subprime scores that keep them out of the market. Although there is risk associated with this group, issuers—many of whom joined a credit access campaign in 2021—may continue to make an effort to connect with these clients. To convert problematic borrowers into dependable customers, they'll deploy secured cards, like the new suite from U.S. Bank, or other credit-building tools, like Wells Fargo Reflect. They might also copy Amex's agreement with Nova Credit by forming alliances to target underserved but potentially creditworthy groups, such immigrants.

Credit Default Explained

Whether a debt is a loan or a security, default occurs, when necessary, interest or principal payments are not made. Debt commitments can be ignored by people, companies, and even entire nations. For creditors, default risk is an essential factor.

Following a borrower's default on a loan, there may be:

- A lower credit score, which is a number that represents a borrower's creditworthiness, and negative information on their credit report
- Decreased chance of future credit acquisition
- Increased interest rates for any future debt

- Income garnishment and other punishments. A legal procedure known as garnishment authorises a third party to take money from a borrower's pay check or bank account on their behalf.

As of 2021, the top issuers had around $933 billion in outstanding receivables, with American Express having the largest year-over-year growth of 21.80%. These debts owing to credit card issuers are known as outstanding receivables, and they are considered assets.

As of 2021, the top issuers had around $933 billion in outstanding receivables, with American Express having the largest year-over-year growth of 21.80%. These debts owing to credit card issuers are known as outstanding receivables, and they are considered assets.

The Center for Microeconomic Data at the Federal Reserve Bank of New York reports that credit card balances increased by $71 billion from the same period last year to $841 billion in the first three months of 2022.

The S&P/Experian Consumer Credit Default Indices showed a Q2 2022 default rate increase for the seventh consecutive quarter in a press release on July 19, 2022. While the default rates for auto loans and first mortgages increased by one basis point and two basis points, respectively, and were both higher than the credit card default rate, which increased by six basis points to 2.55%.

Parameters to identify defaults

Following are the parameters used to identify defaults:

1. Charge-off Rate (Credit Card):
   The percentage of defaulted credit card balances in relation to the total amount of outstanding credit is represented by the credit card charge-off rate. To keep an eye on how well their credit card loans are performing, credit card firms measure charge-off rates. The overall percentage of credit card balances in default can also be determined comprehensibly by calculating the credit charge-off rate across the industry.
2. Exposure at Default:
   The total amount a bank is exposed to in the event a loan defaults is known as exposure at default (EAD). Financial institutions determine their risk using the internal ratingsbased (IRB) methodology. To estimate respective EAD systems, banks frequently utilise internal risk management default models. A term for EAD used outside of the banking sector is credit exposure.
3. Loss Given Default:
   The expected financial loss incurred by a bank or other financial institution as a result of a borrower defaulting on a loan is known as loss given default (LGD). A single dollar amount of possible loss or a percentage of the entire exposure now of default are used to represent LGD. After reviewing all outstanding loans, a financial institution determines its overall LGD by calculating cumulative losses and exposure.
4. Credit scorecard:
   In the business world, credit scorecards are used to give people scores that indicate how risky they are likely to be. This riskiness is based on estimating the likelihood of default, which can be defined differently but generally refers to a set of indicators that a customer is unlikely to pay. They are highly helpful because they are easy to use and

can be explained to non-technical audiences, which is important when boards of directors need to know why decisions have been taken or when people want to know why their loan application has been rejected.

About Dataset

Data Dictionary

Out of the 53 variables included in the training data set, 51 are defined in the data dictionary that is provided.

Name	Definition
application_key	Application ID (primary key)
mvar1	Credit worthiness score calculated on the basis of borrower's credit history
mvar2	A score calculated based on the number and riskiness of credit enquiries made to any lender by a borrower
mvar3	Severity of default by the borrower on any loan(s). Severity is a function of amount, time since default and number of defaults
mvar4	Severity of default by the borrower on auto loan(s). Severity is a function of amount, time since default and number of defaults
mvar5	Severity of default by the borrower on education loan(s). Severity is a function of amount, time since default and number of defaults
mvar6	Minimum of credit available on all revolving credit cards (in $)
mvar7	Maximum of credit available on all active credit lines (in $)
mvar8	Maximum of credit available on all active revolving credit cards (in $)
mvar9	Sum of available credit on credit cards that the borrower has missed 1 payment (in $)
mvar10	Total amount of credit available on accepted credit lines (in $)
mvar11	Amount of dues collected post default where due amount was more than 500 (in $)
mvar12	Sum of amount due on active credit cards (in $)
mvar13	Annual amount paid towards all credit cards during the previous year (in $)
mvar14	Annual income (in $)
mvar15	Estimated market value of a properety owned/used by the borrower (in $)
mvar16	Number of active revolving credit cards on which full credit limit is utilized by the borrower

Name	Definition
mvar17	Number of active credit cards on which full credit limit is utilized by the borrower
mvar18	Number of active credit lines on which full credit limit is utilized by the borrower
mvar19	Number of active credit cards on which atleast 75% credit limit is utilized by the borrower
mvar20	Number of active credit lines on which atleast 75% credit limit is utilized by the borrower
mvar21	Average utilization of active revolving credit card loans (%)
mvar22	Average utilization of line on all active credit lines activated in last 2 years (%)
mvar23	Average utilization of line on all active credit cards activated in last 1 year (%)
mvar24	Average utilization of line on credit cards on which the borrower has missed 1 payment during last 6 months (%)
mvar25	Average tenure of active revolving credit cards (in days)
mvar26	Tenure of oldest credit card among all active credit cards (in days)
mvar27	Tenure of oldest revolving credit card among all active revolving credit cards (in days)
mvar28	Number of days since last missed payment on any credit line
mvar29	Tenure of oldest credit line (in days)
mvar30	Maximum tenure on all auto loans (in days)
mvar31	Maximum tenure on all education loans (in days)
mvar32	Sum of tenures (in months) of active credit cards
mvar33	Duration of stay at the current residential address (in years)

Name	Definition
mvar34	Number of active credit lines over the last 6 months on which the borrower has missed 1 payment
mvar35	Number of revolving credit cards over the last 2 years on which the borrower has missed 1 payment
mvar36	Number of active credit lines
mvar37	Number of credit cards with an active tenure of at least 2 years
mvar38	Number of credit lines activated in last 2 years
mvar39	Number of credit lines on which the borrower has current delinquency
mvar40	Utilization of line on active education loans (%)
mvar41	Utilization of line on active auto loans (%)
mvar42	Financial stress index of the borrower. This index is a function of collection trades, bankruptcies files, tax liens invoked, etc.
mvar43	Number of credit lines on which the borrower has never missed a payment in last 2 yrs, yet considered as high risk loans based on market prediction of economic scenario
mvar44	Ratio of maximum amount due on all active credit lines and sum of amounts due on all active credit lines
mvar45	Number of mortgage loans on which the borrower has missed 2 payments
mvar46	Number of auto loans on which the borrower has missed 2 payments
mvar47	Type of product that the applicant applied for. (C = Charge; L = Lending)
mvar48	Integer value assigned to an application basis a predefined function
mvar49	Bucketized credit worthiness score for the applicant
mvar50	Compound feature created as a product of bucketized credit worthiness and mvar48
default_ind	Indicator for default

Most of the variables are named using a generic naming convention of <mvar_variable_number> and are not representative of what they stand for.

The data variables can be divided into two types:

- Application variables – these variables are collected from the applicant by asking the applicant to fill-in various application forms for credit cards. Some examples of these variables are: Annual income, Type of card applying for – charge or lending, duration of stay in current address.

- Bureau variables – these variables are fetched from bureau that monitors customers credit score, default details etc. Some examples for these variables are: performance on external credit card trades, performance on other trades (auto, personal, education, mortgage loans)

Data Files Description

There are 2 files offered for the project, and they both have 100,000 data entries:

- The 83000 entries in the TrainingData.csv file can be utilised for model training and validation. There are 53 columns.
- The model is tested using a 47000-record dataset called testX.csv. It has 52 columns, as there is no default indicator column.

Libraries Used in the Project

Below mentioned are the libraries that were used in this project:

1. **Pandas** - Pandas is a fast, powerful, flexible, and easy to use open-source data analysis and manipulation tool, built on top of the Python programming language.
2. **Numpy** - NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more.
3. **Missingno** - Missingno is a Python library that provides the ability to understand the distribution of missing values through informative visualizations. The visualizations can be in the form of heat maps or bar charts.
4. **Sklearn** – Sklearn provides simple and efficient tools for predictive data analysis. Accessible to everybody, and reusable in various contexts. Built on NumPy, SciPy, and matplotlib. Can be used for encoding, regression, classification, clustering, performance measurement, and much more.
5. **Matplotlib** - Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
6. **Seaborn** - Seaborn is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.
7. **Scipy** - SciPy is a collection of mathematical algorithms and convenience functions built on the Numpy extension for Python. It adds significant power to the interactive Python session by exposing the user to high-level commands and classes for the manipulation and visualization of data.
8. **Imblearn** - Imbalanced-learn is a python package offering a number of re-sampling techniques commonly used in datasets showing strong between-class imbalance.
9. **Xgboost** - GBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting

(also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

10. **Catboost** - CatBoost is a machine learning algorithm that uses gradient boosting on decision trees. It is available as an open-source library.
11. **Missingpy** - Missingpy is a library for missing data imputation in Python. It has an API consistent with scikit-learn, so users already comfortable with that interface will find themselves in familiar terrain. Currently, the library supports k-Nearest Neighbors based imputation and Random Forest based imputation (MissForest)
12. **Shap** - SHAP (SHapley Additive exPlanations) is a game theoretic approach to explain the output of any machine learning model. It connects optimal credit allocation with local explanations using the classic Shapley values from game theory and their related extensions.
13. **Miceforest** - Miceforest imputes missing data using LightGBM in an iterative method known as Multiple Imputation by Chained Equations (MICE). It was designed to be fast, flexible, and production ready.
14. **Collections** - Collections module implements specialized container datatypes providing alternatives to Python's general purpose built-in containers, dict, list, set, and tuple.
15. **Sys** - Sys module provides access to some variables used or maintained by the interpreter and to functions that interact strongly with the interpreter. It is always available.

Exploratory Data Analysis

Data Exploration

1. Data Dimensions

```
In [17]: train_data.shape
Out[17]: (83000, 52)
```

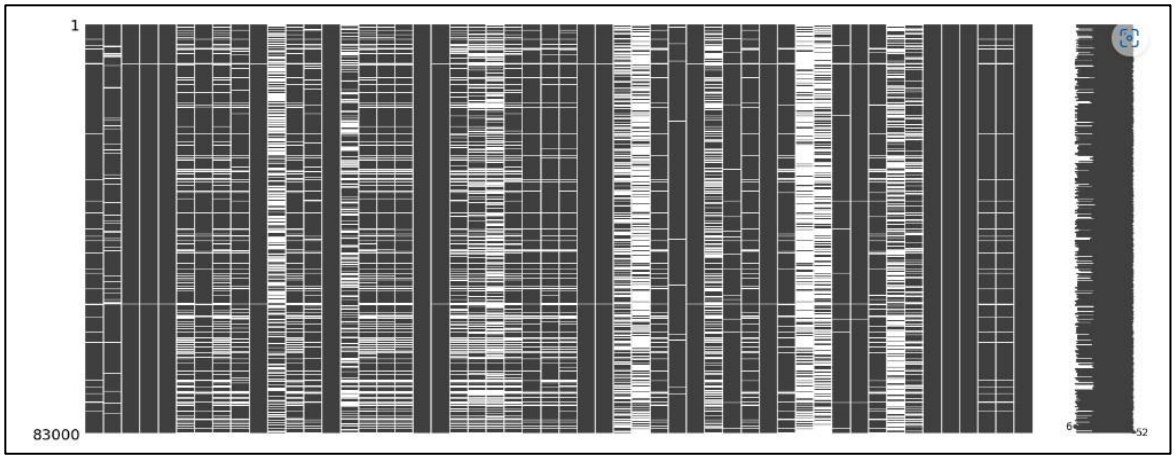
```
In [19]: test_data.shape
Out[19]: (47000, 51)
```

2. Missing Values

```
In [20]: counter = null_counter_columns(train_data)
         counter
```

```
Out[20]: mvar1          0.044976
         mvar2          0.070916
         mvar3          0.006446
         mvar4          0.006446
         mvar5          0.006446
         mvar6          0.237361
         mvar7          0.092458
         mvar8          0.237458
         mvar9          0.140747
         mvar10         0.006446
         mvar11         0.562855
         mvar12         0.175639
         mvar13         0.116735
         mvar14         0.000000
         mvar15         0.403843
         mvar16         0.231843
         mvar17         0.198783
         mvar18         0.185048
         mvar19         0.000060
         mvar20         0.006446
         mvar21         0.282675
         mvar22         0.369494
         mvar23         0.509771
         mvar24         0.235301
         mvar25         0.094723
         mvar26         0.131675
         mvar27         0.164458
         mvar28         0.006446
         mvar29         0.006446
         mvar30         0.457687
         mvar31         0.705289
         mvar32         0.094723
         mvar33         0.022518
         mvar34         0.006446
         mvar35         0.420096
         mvar36         0.038060
         mvar37         0.094723
         mvar38         0.006446
         mvar39         0.076253
         mvar40         0.783976
         mvar41         0.689928
         mvar42         0.024373
         mvar43         0.010711
         mvar44         0.098181
         mvar45         0.553253
         mvar46         0.284373
         mvar47         0.000000
         mvar48         0.000000
         mvar49         0.000000
         mvar50         0.044976
         mvar51         0.044976
         default_ind    0.000000
         dtype: float64
```

```
In [6]: def null_counter_columns(dataset):
            """
            Returns the list of columns with the percentage of missing values it contains
            """
            counter = dataset.isna().sum()/len(dataset)
            msno.matrix(dataset)
            return counter
```



3. Data Description

```
In [21]: train_data.describe()
```

Out[21]:

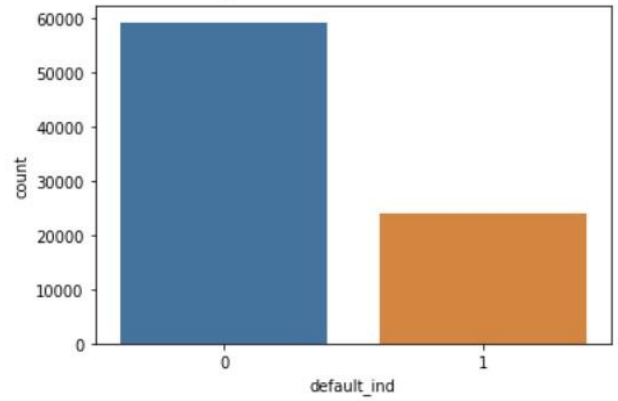
	mvar1	mvar2	mvar3	mvar4	mvar5	mvar6	mvar7	mvar8	mvar9	mvar10	...
count	79267.000000	77114.000000	82465.000000	82465.000000	82465.000000	63299.000000	7.532600e+04	63291.000000	71318.000000	8.246500e+04	...
mean	1747.511865	1.054816	5.401784	0.461151	1.084012	1633.429280	1.750797e+04	6822.332227	34030.596329	3.022857e+04	...
std	94.830127	1.556682	11.091569	1.704292	5.743899	3667.183981	4.677222e+04	10060.346814	50673.150005	6.617798e+04	...
min	1477.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000000	0.000000	0.000000e+00	...
25%	1680.000000	0.131800	0.000000	0.000000	0.000000	41.000000	1.750250e+03	496.000000	3542.250000	1.153000e+03	...
50%	1743.000000	0.513000	0.300000	0.000000	0.000000	297.000000	7.020500e+03	2507.000000	14389.000000	9.525000e+03	...
75%	1813.000000	1.386200	6.595000	0.000000	0.000000	1381.000000	1.811300e+04	9869.500000	44413.000000	3.428800e+04	...
max	1950.000000	31.018100	399.334000	25.754000	165.492000	94302.000000	5.637108e+06	291810.000000	840658.000000	5.647073e+06	...

8 rows × 52 columns

4. Imbalanced Classes

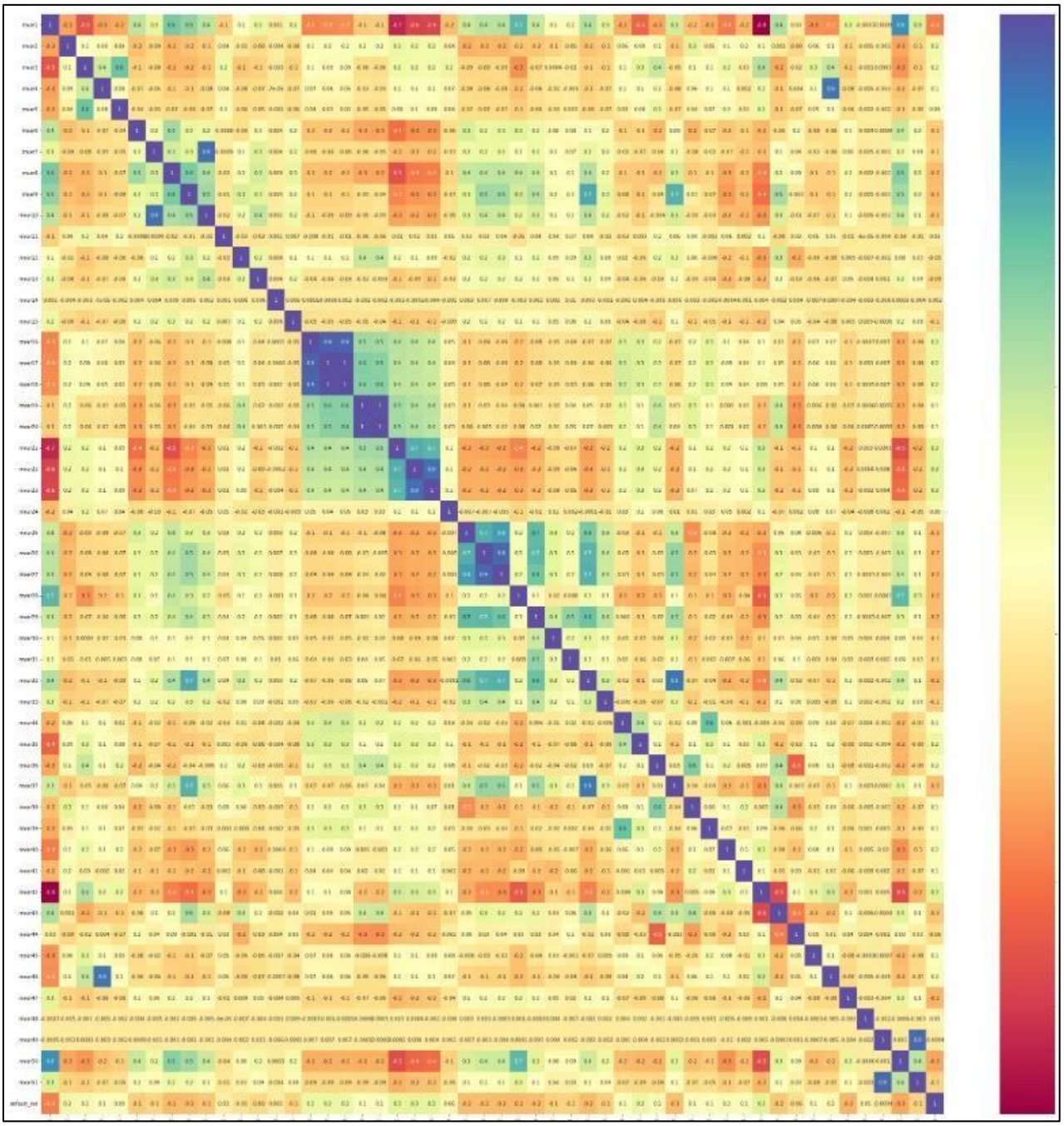
```
sns.countplot(x='default_ind', data=train_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fbcd4244ee0>
```



Data imbalance is observed with “Default” cases only 28.74% of the cases

5. Correlation



We can observe multiple columns extremely correlated with each other – hence might cause issues of multi-collinearity.

Get pairs of columns very highly correlated to each other to avoid manual selection from the heatmap.

```
def print_highly_correlated(df, features, threshold=0.5):
  corr_df = df[features].corr(method='spearman', min_periods = 3) # get correlations
  correlated_features = np.where(np.abs(corr_df) > threshold) # select ones above the abs threshold
  correlated_features = [(corr_df.iloc[x,y], x, y) for x, y in zip(*correlated_features) if x != y and x < y]
  s_corr_list = sorted(correlated_features, key=lambda x: -abs(x[0])) # sort by correlation value

  if s_corr_list == []:
    print("There are no highly correlated features with correlation above", threshold)
  else:
    for v, i, j in s_corr_list:
      cols = df[features].columns
      print ("%s and %s = %.3f" % (corr_df.index[i], corr_df.columns[j], v))
```

```
print_highly_correlated(train_data_check,train_data_check.columns,threshold=0.7)
```

```
mvar19 and mvar20 = 0.959
mvar32 and mvar37 = 0.902
mvar7 and mvar10 = 0.882
mvar3 and mvar28 = -0.842
mvar1 and mvar42 = -0.826
mvar25 and mvar32 = 0.798
mvar9 and mvar32 = 0.797
mvar26 and mvar32 = 0.789
mvar1 and mvar28 = 0.786
mvar25 and mvar26 = 0.782
mvar1 and mvar50 = 0.775
mvar9 and mvar10 = 0.767
mvar1 and mvar3 = -0.721
mvar1 and mvar10 = 0.720
mvar7 and mvar9 = 0.706
mvar29 and mvar32 = 0.705
mvar21 and mvar24 = 0.701
```

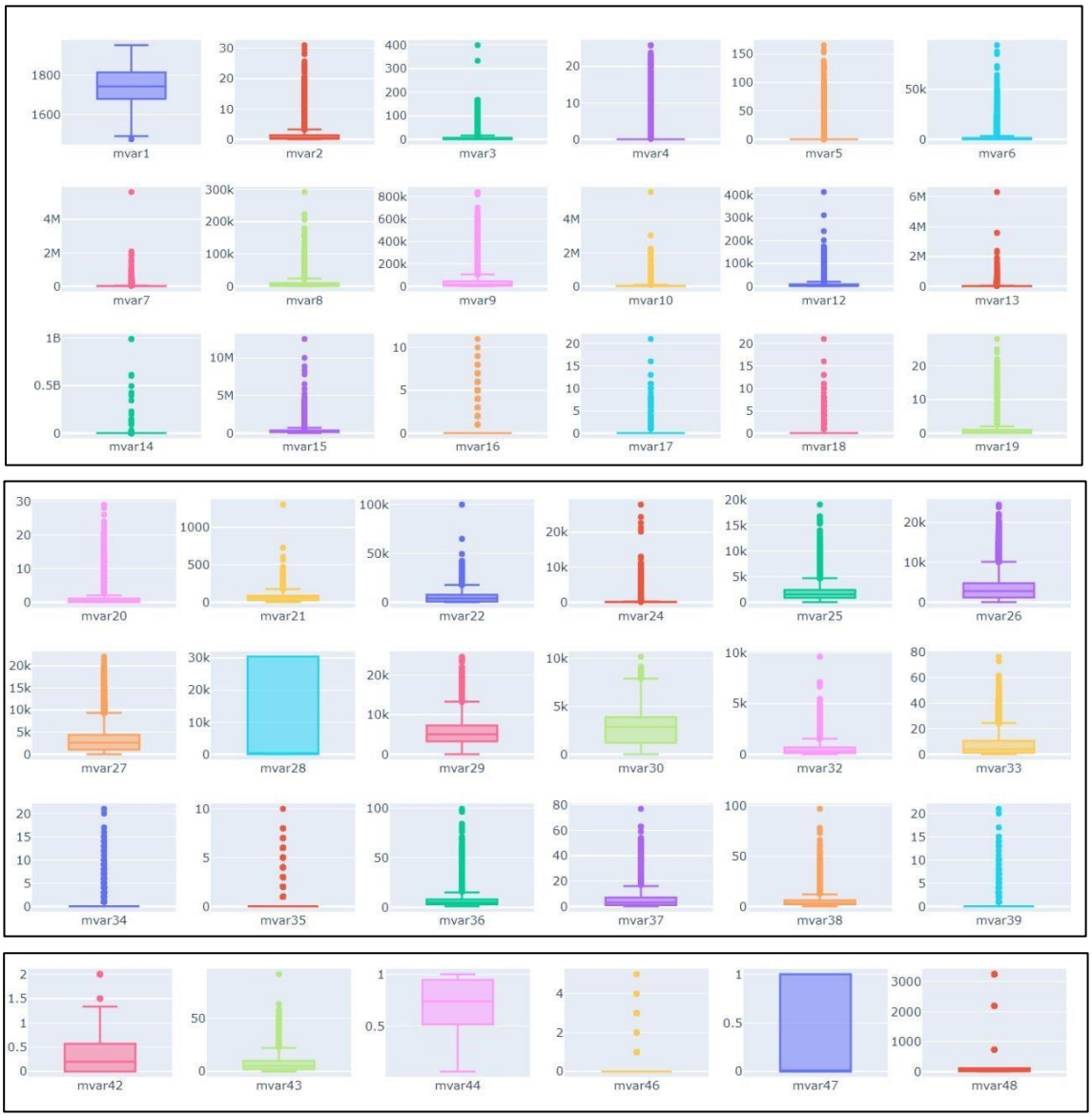
Generating VIFs to determine which columns to drop

```
def generate_VIF(Y):

    from statsmodels.stats.outliers_influence import variance_inflation_factor
    Y.dropna(inplace=True) # vif can't be calculated with nan values
    Y = Y._get_numeric_data()

    # Compute VIF
    vif = pd.DataFrame()
    vif["variables"] = Y.columns
    vif["VIF"] = [variance_inflation_factor(Y.values, i) for i in range(Y.shape[1])]
    print(vif.sort_values(by='VIF', ascending=False))
```

```
generate_VIF(train_data_check)
```

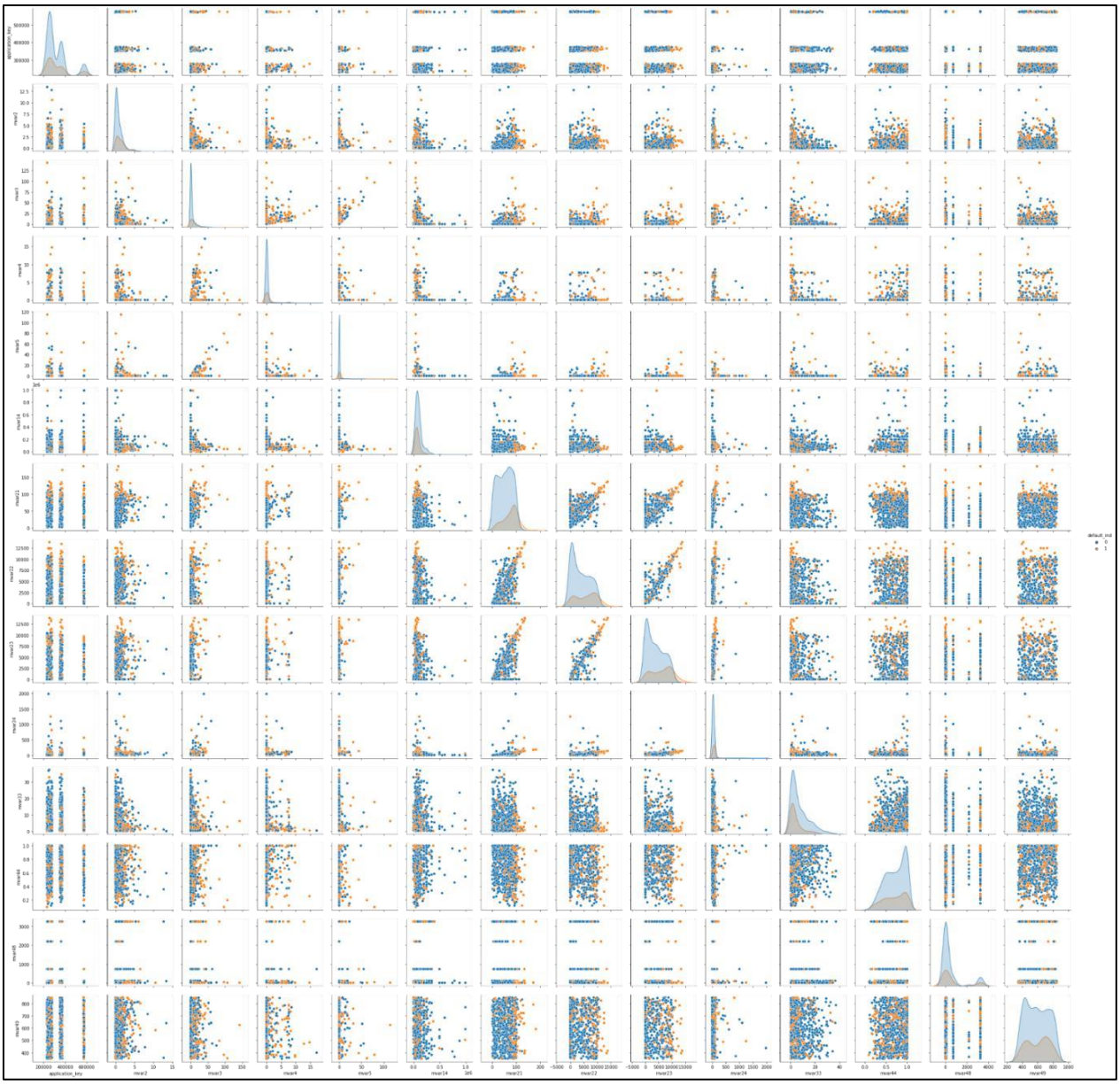
	variables	VIF
13	mvar20	47.430395
12	mvar19	46.578376
22	mvar32	9.869848
6	mvar10	8.564757
0	mvar1	8.230847
4	mvar7	7.119121
26	mvar37	5.649634
34	mvar50	4.622262
5	mvar9	4.248400
30	mvar43	3.885726
17	mvar25	3.630380

6. Univariate Analysis and Outlier Detection



There is a significant existence of outliers in more than one column which needs to be kept or removed as suitable for the business problem in hand. Also, outliers are important if they add business value to the problem. For example, mvar14 which talks about the annual income has many outliers. Now if we remove the rows having outliers, we will miss out on valuable data of the category of person who falls in the rich category. So, outliers need to be treated with caution.

7. Independent and Dependent Variable Relationship & Feature Distribution



Data Pre-Processing

Handling Categorical Encoding

```
def binary_encoder(dataset,feature,label_encoder_obj):
    """
    Encode binary categorical variables with the same encoding as mentioned in the input object
    """
    for f in feature:
        dataset[f] = label_encoder_obj.fit_transform(dataset[f])
    return dataset
```

```
In [23]: train_data['mvar47'].head()

Out[23]: application_key
         230032    0.0
         230035    1.0
         230036    1.0
         230039    0.0
         230041    1.0
         Name: mvar47, dtype: float64
```

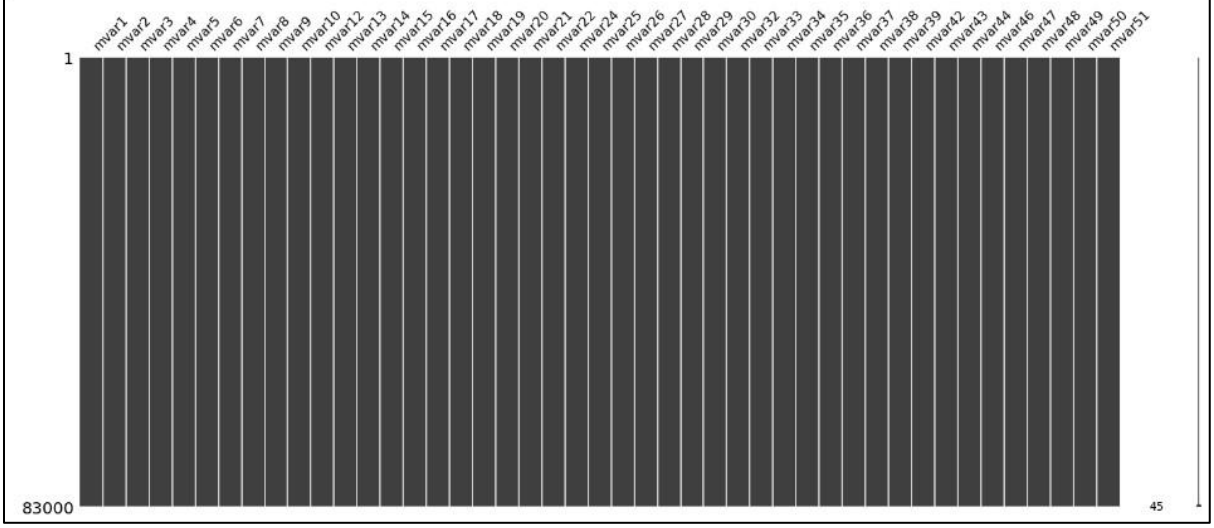
Handling Missing Data

Multiple Imputation by Chained Equations 'fills in' (imputes) missing data in a dataset through an iterative series of predictive models. In each iteration, each specified variable in the dataset is imputed using the other variables in the dataset. These iterations should be run until it appears that convergence has been met. This process is continued until all specified variables have been imputed. Additional iterations can be run if it appears that the average imputed values have not converged.

```
kernel = mf.ImputationKernel(data=train_data,save_all_iterations=True,random_state=2)
kernel.mice(3,verbose=True)
```

```
new_data_imputed = kernel.impute_new_data(train_data)
new_completed_data = new_data_imputed.complete_data(0)
new_completed_data = pd.DataFrame(data=new_completed_data,columns=train_data.columns,index=train_data.index)
```

```
counter = null_counter_columns(new_completed_data)
```



Data Transformation

Some algorithms require standardization of the data to deal with the outliers or to make the data in each column as close to normally distributed for accurate training and prediction.

Power transforms are a family of parametric, monotonic transformations that are applied to make data more Gaussian-like. This is useful for modelling issues related to heteroscedasticity (non-constant variance), or other situations where normality is desired.

```
pt = PowerTransformer()
testing=pt.fit_transform(new_completed_data)
testing = pd.DataFrame(data=testing,columns=new_completed_data.columns,index=train_data.index)
```

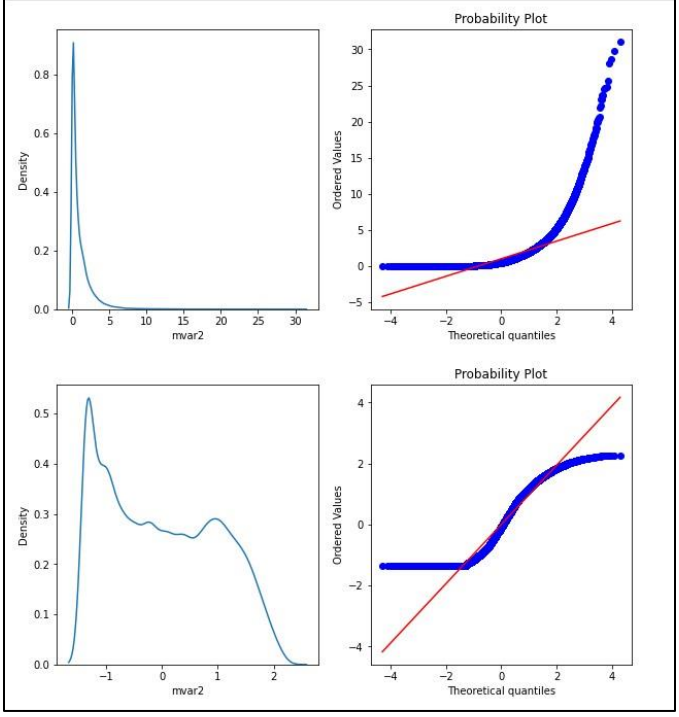
```
testing.describe()
```

	mvar1	mvar2	mvar3	mvar4	mvar5	mvar6	mvar7	mvar8	mvar9	mvar10	...
count	8.300000e+04	8.300000e+04	8.300000e+04	8.300000e+04	8.300000e+04	8.300000e+04	8.300000e+04	8.300000e+04	8.300000e+04	8.300000e+04	...
mean	-9.376210e-15	-8.435590e-15	-7.529775e-14	1.016004e-14	5.363610e-14	9.765127e-15	-1.148303e-14	-1.719752e-14	3.455909e-15	-1.094386e-14	...
std	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	1.000006e+00	...
min	-3.016902e+00	-1.364390e+00	-9.130545e-01	-3.948037e-01	-3.236793e-01	-1.713153e+00	-2.142772e+00	-2.131957e+00	-2.977565e+00	-1.765098e+00	...
25%	-7.049797e-01	-9.290530e-01	-9.130545e-01	-3.948037e-01	-3.236793e-01	-5.614197e-01	-6.334886e-01	-7.048643e-01	-7.561707e-01	-6.328043e-01	...
50%	-9.097514e-03	-8.920048e-02	-4.875930e-01	-3.948037e-01	-3.236793e-01	8.681649e-03	6.426474e-02	-2.674298e-02	5.143237e-03	1.064272e-01	...
75%	7.016956e-01	8.694415e-01	1.120480e+00	-3.948037e-01	-3.236793e-01	6.260766e-01	6.520722e-01	7.758689e-01	7.549122e-01	7.169748e-01	...
max	2.064062e+00	2.276340e+00	1.867600e+00	2.659489e+00	3.144278e+00	3.370522e+00	7.345685e+00	3.826068e+00	3.313694e+00	5.294214e+00	...

8 rows × 46 columns

```
def normality(data,feature):
    """
    Used to plot the distribution graph and Q-Q plot for a specified feature in the input dataset
    """
    plt.figure(figsize=(10,5))
    plt.subplot(1,2,1)
    sns.kdeplot(data[feature])
    plt.subplot(1,2,2)
    stats.probplot(data[feature],plot=pylab)
    plt.show()
```

```
normality(new_completed_data,'mvar2')
normality(testing,'mvar2')
```



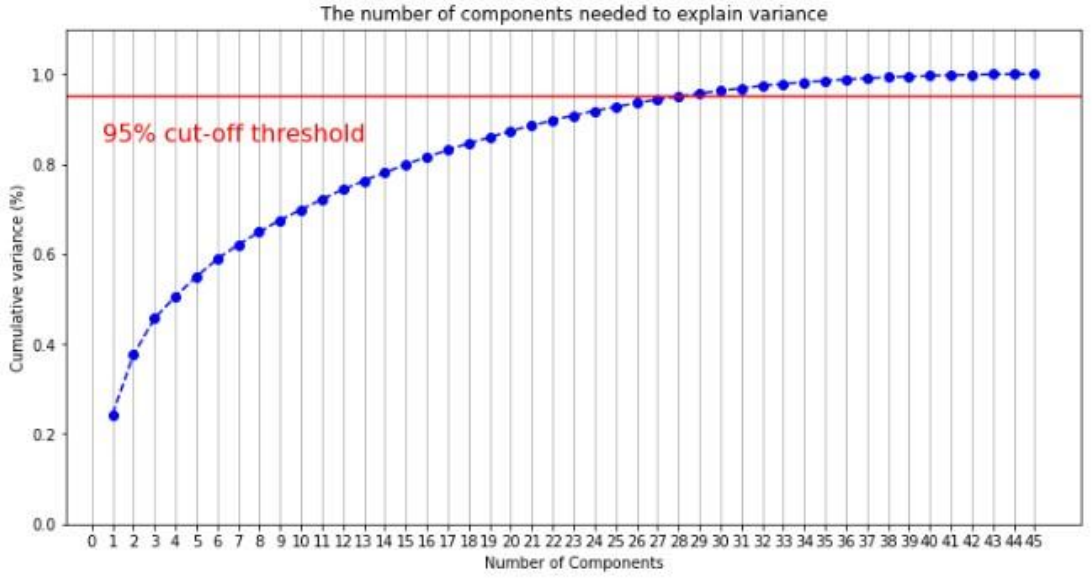
Principal Component Analysis

```
pca_testing = pd.DataFrame(data=testing,columns=testing.columns,index=testing.index)
pca_testing.drop(['default_ind'],inplace=True,axis=1)
```

```
from sklearn.decomposition import PCA

principal = PCA(n_components=0.95)
principal.fit(pca_testing)
pca_data = principal.transform(pca_testing)
print(pca_data.shape)
```

```
(83000, 28)
```

```
pca = PCA().fit(pca_testing)
plt.rcParams["figure.figsize"] = (12,6)

fig, ax = plt.subplots()
xi = np.arange(1, 46, step=1)
y = np.cumsum(pca.explained_variance_ratio_)

plt.ylim(0.0,1.1)
plt.plot(xi, y, marker='o', linestyle='--', color='b')

plt.xlabel('Number of Components')
plt.xticks(np.arange(0, 46, step=1)) #change from 0-based array index to 1-based human-readable label
plt.ylabel('Cumulative variance (%)')
plt.title('The number of components needed to explain variance')

plt.axhline(y=0.95, color='r', linestyle='-')
plt.text(0.5, 0.85, '95% cut-off threshold', color = 'red', fontsize=16)

ax.grid(axis='x')
plt.show()
```

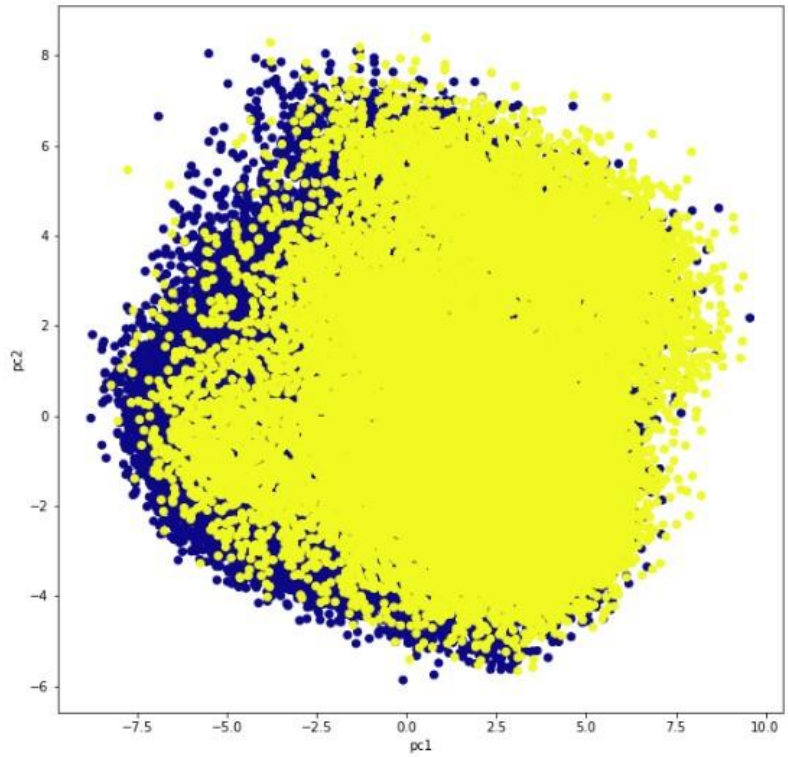



```
print(principal.explained_variance_ratio_)
```

```
[0.24379878 0.13268154 0.07999794 0.04988329 0.04306781 0.03968988
 0.03171009 0.02812474 0.02650996 0.02351483 0.02273867 0.02179444
 0.01930628 0.01849819 0.0176653  0.01684691 0.01554899 0.01442029
 0.01392757 0.01360055 0.01214715 0.01168007 0.01077875 0.0101036
 0.00924756 0.00827057 0.00769533 0.00697421]
```

```
#Plotting component 1 and component 2 - 2D Graph - to explain maximum vaariance in data using 1st - 2 components

plt.figure(figsize=(10,10))
plt.scatter(pca_data[:,0],pca_data[:,1],c=testing['default_ind'],cmap='plasma')
plt.xlabel('pc1')
plt.ylabel('pc2')
```

```
Text(0, 0.5, 'pc2')
```



Model Training

1. Logistic Regression

A classification process called logistic regression is used to determine the likelihood that an event will succeed or fail. When the dependent variable is binary (True/False, Yes/No, 0/1, etc.), it is used. By analysing the relationship from a given collection of labelled data, it helps classifying data into distinct classes. From the provided dataset, it learns a linear relationship before introducing a non-linearity in the form of the Sigmoid function.

Advantages:
Multiple classes (multinomial regression) and a natural probabilistic perspective of class predictions can be added with ease. It doesn't make any assumptions about how classes are distributed in feature space.

Disadvantages:
Logistic regression has a linear decision surface; hence it cannot address non-linear issues. Real-world situations rarely involve linearly separable data.

An average level of multicollinearity between the independent variables is required for logistic regression.

```
def LogisticRegression_Classifier(X_train, X_test, y_train, y_test):
    logreg = LogisticRegression()
    logreg.fit(X_train, y_train.values.ravel())
    y_pred = logreg.predict(X_test)
    print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))

    print("ROC-AUC Score: ",roc_auc_score(y_test,y_pred))
    from sklearn.metrics import confusion_matrix
    confusion_matrix = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:\n",confusion_matrix)
    from sklearn.metrics import classification_report
    print("Classification Report:\n",classification_report(y_test, y_pred))

    return logreg
```

ROC-AUC Score	Transformed Data		PCA Data	
Model	Regular	SMOTE	Regular	SMOTE
Logistic Regression	0.6547	0.7102	0.6551	0.7099

2. Naïve Bayes Classifier

We employ the machine learning technique Naive Bayes to address classification issues. The Bayes Theorem is the basis of it. It is one of the most straightforward yet effective ML algorithms in use and has applications across numerous sectors.
Imagine having a classification problem to answer, and you've already developed the features and the hypothesis. However, your superiors want to see the model. To train the dataset, you

have many variables and thousands of data points. The Naive Bayes classifier, which is far quicker than other classification algorithms, would be the best course of action in this case.

Advantages:
It can outperform other models and needs a lot less training data if its assumption about the independence of characteristics is correct.

This algorithm is efficient and can greatly reduce processing time.

Disadvantages:
Naive Bayes makes the uncommon but unfounded assumption that all predictors (or features) are independent. This restricts the algorithm's usability in practical usage cases.

This approach encounters the "zero-frequency problem," where it gives a categorical variable with zero probability if its category was not present in the training dataset but was present in the test data set. To solve this problem, it would be better if you employed a smoothing method.

```
def naive_bayes(X_train, X_test, y_train, y_test):

    gnb = BernoulliNB()
    gnb.fit(X_train, y_train.values.ravel())

    #Applying and predicting
    y_pred_nb = gnb.predict(X_test)
    cv_scores = cross_val_score(gnb, X, y.values.ravel(),cv=10,scoring='precision')
    print("Cross-validation precision: %f" % cv_scores.mean())

    print("ROC-AUC Score: ",roc_auc_score(y_test,y_pred_nb))
    from sklearn.metrics import confusion_matrix
    confusion_matrix = confusion_matrix(y_test, y_pred_nb)
    print("Confusion Matrix:\n",confusion_matrix)
    from sklearn.metrics import classification_report
    print("Classification Report:\n",classification_report(y_test, y_pred_nb))

    return gnb
```

ROC-AUC Score	Transformed Data		PCA Data	
Model	Regular	SMOTE	Regular	SMOTE
Naïve Bayes Classifier	0.6922	0.6904	0.5836	0.6856

3. Random Forest Classifier

A random forest is made up of several Decision Trees that each independently predict something. To determine the final result, the values are averaged (Regression) or max-voted (Classification).

This model's strength is in its ability to construct several trees from the features with various sub-features. Because each tree's features are chosen at random, the trees do not grow very deep and just concentrate on the set of features.

Finally, after combining them, we get an ensemble of Decision Trees that offers a forecast that has been learned.

Advantages:

- It lessens decision tree overfitting and increases accuracy.
- It automates filling in data's missing values.

Disadvantages:

- As it creates several trees to integrate their outputs, it uses a lot of resources and computational power.
- As it integrates numerous decision trees to decide the class, training takes a lot of time.

```
def randomforest_classifier(X_train, X_test, y_train, y_test):

    rt=RandomForestClassifier(n_estimators=100,class_weight="balanced_subsample")
    rt.fit(X_train,y_train.values.ravel())
    y_pred=rt.predict(X_test)
    print("Recall:",recall_score(y_test, y_pred))
    roc_value = roc_auc_score(y_test, y_pred)
    print("ROC Value:",roc_value)

    from sklearn.metrics import confusion_matrix
    confusion_matrix = confusion_matrix(y_test, y_pred)
    print("Confusion Matrix:\n",confusion_matrix)
    from sklearn.metrics import classification_report
    print("Classification Report:\n",classification_report(y_test, y_pred))

    return rt
```

ROC-AUC Score	Transformed Data		PCA Data	
Model	Regular	SMOTE	Regular	SMOTE
Random Forest Classifier	0.6410	0.8143	0.6318	0.7832

4. XGBoost Classifier

When employing XGBoost, each regression tree transfers an input data point to one of its leaves that has a continuous score when the weak learners are viewed as regression trees. The convex loss function, which is based on the variation between the goal outputs and the forecast outputs, is merged into a regularised objective function that XGB minimises. After that, the training process continues iteratively as additional trees are added with the ability to forecast residuals and prior tree errors, which are then combined with the earlier trees to get the final prediction.

Advantages:

- In comparison to gradient boosting machines, XGB has a variety of hyper-parameters that can be modified.
- XGBoost offers the ability to manage missing values right out of the box.
- Intuitive features like parallelization, distributed computing, cache optimization, and more are offered by it.

Disadvantages:

- Like all boosting techniques, XGB is susceptible to outliers.
- Contrary to LightGBM, in XGB category features must be manually encoded as dummy variables or labels before being fed to the models.

```
def xgb_classification(X_train, X_test, y_train, y_test):
    xg_reg = xgb.XGBClassifier(objective ='binary:logistic', colsample_bytree = 0.3, learning_rate = 0.1,
                               alpha = 2, n_estimators = 100)
    xg_reg.fit(X_train,y_train.values.ravel())
    preds = xg_reg.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, preds))
    print("RMSE: %f" % (rmse))

    print("ROC-AUC Score: ",roc_auc_score(y_test,preds))
    from sklearn.metrics import confusion_matrix
    confusion_matrix = confusion_matrix(y_test, preds)
    print("Confusion Matrix:\n",confusion_matrix)
    from sklearn.metrics import classification_report
    print("Classification Report:\n",classification_report(y_test, preds))

    return xg_reg
```

ROC-AUC Score	Transformed Data		PCA Data	
Model	Regular	SMOTE	Regular	SMOTE
XGBoost Classifier	0.6684	0.8307	0.6540	0.7290

5. Histogram Gradient Boosting Classifier

Gradient boosting has a significant drawback in that the model is trained slowly. This poses a challenge in particular when applying the model to huge datasets with thousands of samples (rows).
By discretizing (binning) the continuous input variables to a small subset of a few hundred distinct values, training the trees that are introduced to the ensemble can be significantly expedited. Histogram-based gradient boosting ensembles are gradient boosting ensembles that use this method and modify the training process to take into account the input variables covered by this transform.

Advantages:

- Histogram Boosting Gradient Classifier technique aids in parameter fine-tuning and accuracy improvement.
- It also can deal with the dataset without any imputation or transformation required prior

Disadvantages:

- It is very susceptible to oversampling and does not give reliable results for imbalanced classes without weight addition.

```
def HistGradientBoostingClassification(X_train, X_test, y_train, y_test):

    clf = HistGradientBoostingClassifier()
    clf.fit(X_train, y_train.values.ravel())
    y_pred = clf.predict(X_test)
    print('Accuracy of classifier on train set: {:.2f}'.format(clf.score(X_train, y_train)))
    print('Accuracy of classifier on test set: {:.2f}'.format(clf.score(X_test, y_test)))

    print("ROC-AUC Score: ",roc_auc_score(y_test,y_pred))
    from sklearn.metrics import confusion_matrix
    print("Confusion Matrix:\n",confusion_matrix(y_test, y_pred))
    from sklearn.metrics import classification_report
    print("Classification Report:\n",classification_report(y_test, y_pred))

    return clf
```

ROC-AUC Score	Transformed Data		PCA Data	
Model	Regular	SMOTE	Regular	SMOTE
Histogram Gradient Boosting Classifier	0.6673	0.8300	0.6583	0.7307

6. CatBoost Classifier

CatBoost is an acronym for categorical boosting. In this case, Cat stands for categorical, and Boost for boosting. It is a well-known gradient boosting-based ensemble machine learning approach. In terms of performance, accuracy, implementation, hyper-tuning parameters, and many other factors, this approach surpasses many other boosting algorithms like XGBoost, LightGBM, etc. Although it works well with categorical data, categorical boosting can also handle text and numerical data aspects.

Advantages:

- It provides excellent outcomes for categorical data.
- Our model can be trained using GPU technology, dramatically speeding up learning
- Using the class weight parameter of the model, we may also balance a dataset that is unbalanced.

- In addition to interpreting essential features, CatBoost also returns important features for a particular data point.

Disadvantages:

- Only when we have categorical data does it outperform other methods.
- If the variables are not correctly calibrated, the model can perform very poorly.
- Sparse matrices are not currently supported by CatBoost.
- CatBoost takes a long time to train when the dataset has a lot of numerical information.

```
def CatBoost_classifier(X_train, X_test, y_train, y_test,weight):

    accuracy= []
    recall =[]
    roc_auc= []
    precision = []

    clf = CatBoostClassifier(verbose=True,random_state=42,scale_pos_weight=weight,
                             iterations=300,depth=6,eval_metric="AUC")
    clf.fit(X_train, y_train.values.ravel(),
            eval_set=(X_test,y_test.values.ravel()))
    y_pred = clf.predict(X_test)

    accuracy.append(round(accuracy_score(y_test, y_pred),4))
    recall.append(round(recall_score(y_test, y_pred),4))
    roc_auc.append(round(roc_auc_score(y_test, y_pred),4))
    precision.append(round(precision_score(y_test, y_pred),4))

    model_names = ['Catboost_default']
    result_df1 = pd.DataFrame({'Accuracy':accuracy,'Recall':recall, 'Roc_Auc':roc_auc, 'Precision':precision}, index=model_names)
    print(result_df1)

    from sklearn.metrics import confusion_matrix
    print("Confusion Matrix:\n",confusion_matrix(y_test, y_pred))
    from sklearn.metrics import classification_report
    print("Classification Report:\n",classification_report(y_test, y_pred))

    return clf
```

ROC-AUC Score	Transformed Data		PCA Data	
Model	Regular	SMOTE	Regular	SMOTE
CatBoost Classifier [weight = 3]	0.7137	0.7789	0.7070	0.6755

7. Artificial Neural Network

Neural networks are a collection of algorithms that mimic the functioning of the human brain to find patterns in massive volumes of data. A process model supported by biological neural networks could be an artificial neural network (ANN), also known as a "Neural Network" (NN). It is made up of a networked group of synthetic neurons. A neural network is a collection of connected input/output units with weights assigned to each connection. In order to be able to correctly anticipate the class label of the input samples, the network accumulates information during the knowledge phase by modifying the weights. Due to the links between units, neural network learning is also known as connectionist learning.

Advantages:

- A neural network is made to continuously learn and produce better results. Once trained, the system may generate output without requiring complete inputs. The programme or applications become more user-friendly as they are used.
- Working with incomplete knowledge is a key advantage. After ANN training, the data may still give output with just partial knowledge.

- It is fault tolerant because it can still produce output even if one or more of its cells are corrupted. The networks are fault-tolerant thanks to this characteristic.

Disadvantages:

- Even though the data is saved online, hardware is still needed to build artificial networks in the first place. With increasing problem complexity comes more hardware costs, and maintaining their setup takes more work.
- Human analysts are unable to track and verify the derivations, even when the results are valid. The majority of neural networks are "black-box" systems that produce outcomes based on experience rather than predefined programming, making adjustments challenging.
- The machine reacts in accordance with the data that is provided to it. The results are more accurate when more data is used during training. Dependence on data is one of the key drawbacks of neural networks because someone needs to be in charge of maintaining it.

```
def ann_classification(X_train, X_test, y_train, y_test,w):

    weights = {0:1,1:w}
    #initializing model
    ann = tf.keras.models.Sequential()

    #adding hidden layers
    #first layer
    ann.add(tf.keras.layers.Dense(units=10,activation="relu"))
    #second layer
    ann.add(tf.keras.layers.Dense(units=6,activation="relu"))

    #output layer - using only 1 neuron - since binary decision variable
    #binary classification - sigmoid, (use "softmax" for multiclass classification)
    ann.add(tf.keras.layers.Dense(units=1,activation="sigmoid"))

    #compiling model
    #loss:- specifies which loss function should be used.
    #For binary classification, the value should be binary_crossentropy.
    #For multiclass classification, it should be categorical_crossentropy.

    ann.compile(optimizer="sgd",loss="binary_crossentropy",
                metrics=[tf.keras.metrics.AUC()])

    #model fitting
    #X_train:- Matrix of features for the training dataset
    #Y_train:- Dependent variable vectors for the training dataset
    #batch_size: how many observations should be there in the batch. Usually, the value for this parameter is 32 but we can expe
    #epochs: How many times neural networks will be trained. Here the optimal value that I have found from my experience is 100.

    ann.fit(X_train,y_train,batch_size=32,epochs=50,class_weight=weights)
    ann.save("/content/drive/MyDrive/7. Quarter 6/PPDA/Ann.h5")
    score_ann = ann.evaluate(X_test, y_test, verbose = 0)

    print('Test loss:', score_ann[0])
    print('Test AUC:', score_ann[1])
    y_pred_ann = ann.predict(X_test)
    y_pred_ann = np.round_(y_pred_ann)

    print("ROC-AUC Score: ",roc_auc_score(y_test,y_pred_ann))
    from sklearn.metrics import confusion_matrix
    confusion_matrix = confusion_matrix(y_test, y_pred_ann)
    print("Confusion Matrix:\n",confusion_matrix)
    from sklearn.metrics import classification_report
    print("Classification Report:\n",classification_report(y_test, y_pred_ann))

    return ann
```

ROC-AUC Score	Transformed Data		PCA Data	
Model	Regular	SMOTE	Regular	SMOTE
Artificial Neural Network [weight = 3]	0.7068	0.6503	0.7050	0.6433

Model Selection and Tuning

ROC-AUC Score	Transformed Data		PCA Data	
Model	Regular	SMOTE	Regular	SMOTE
Logistic Regression	0.6547	0.7102	0.6551	0.7099
Naïve Bayes Classifier	0.6922	0.6904	0.5836	0.6856
Random Forest Classifier	0.6410	0.8143	0.6318	0.7832
XGBoost Classifier	0.6684	0.8307	0.6540	0.7290
Histogram Gradient Boosting Classifier	0.6673	0.8300	0.6583	0.7307
CatBoost Classifier [weight = 3]	0.7137	0.7789	0.7070	0.6755
Artificial Neural Network [weight = 3]	0.7068	0.6503	0.7050	0.6433

CatBoost performs best among the above tested models. Hence, we proceed to tune the hyperparameters to give the best result.

1. CatBoost – HyperParameter Tuning

We applied Grid Search methodology on a pre-specified parameters dictionary to find the most efficient hypermeters.

```
params = {'iterations': [200,300,400],
          'depth': [5, 6, 7],
          'l2_leaf_reg': np.logspace(-20, 3),
          'eval_metric': ['AUC'],
          'random_seed': [42]
         }

clf = CatBoostClassifier(verbose=True,scale_pos_weight=2.6)
clf_grid = GridSearchCV(estimator=clf, param_grid=params, scoring='roc_auc', cv=5)


clf_grid.fit(X, y)
best_param = clf_grid.best_params_
```

Best Params:

{ 'iterations': 300, 'depth': 6, 'l2_leaf_ref': 1e-20, 'eval_metric': 'AUC', 'random_seed': 42 }

2. Weight Optimization

Due to class imbalance, weights need to be appropriately assigned to yield the desirable result.

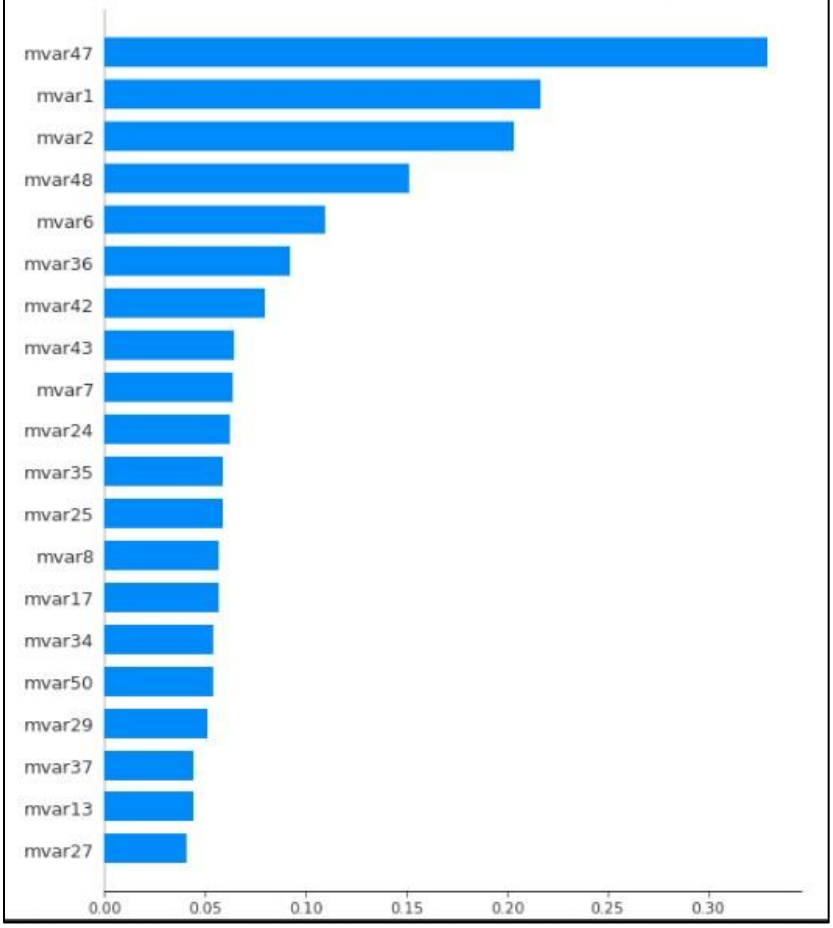
Multiple weights were tested and the following weights yielded the best result – class 0 (no default): **1**

class 1(default): **2.6**

ROC-AUC Score	
CatBoost Weight Optimization	Data
CatBoost Classifier [weight = 3]	0.7137
CatBoost Classifier [weight = 2.5]	0.7169
CatBoost Classifier [weight = 2.6]	0.7171
CatBoost Classifier [weight = 2.7]	0.7158
CatBoost Classifier [weight = 2.8]	0.7145

Feature Importance

```
explainercat = shap.TreeExplainer(model_cat_2_6)
shap_values_cat_test = explainercat.shap_values(X_test)
shap_values_cat_train = explainercat.shap_values(X_train)

shap.summary_plot(shap_values_cat_train, X_train, plot_type="bar")
```



Important Takeaways

1. **Class Imbalance**
    a. SMOTE Resampling
        i. Since data was imbalanced – we used SMOTE resampling to generate more

"default" cases in order to provide a more balanced set of data. ii. Though we observed a higher ROC value in some SMOTE cases – they did not seem reliable when the results were tested on the "testX.csv" dataset.

iii. It overclassified "default" cases leading to worse results. Hence oversampling is not always the right answer to class imbalance.

b. Weights
   i. Adding class weights in the algorithms especially CatBoost and Artificial Neural Network did provide very reliable results which were consistently performing at a higher standard.
   ii. We experimented with multiple class weights – general rule of thumb to determine the weight of minority class is ->

   $$\text{weight of minority class} = \frac{number\ of\ majority\ class\ records}{number\ of\ minority\ class\ records}$$ i.e., 2.57

   Consequently weight of 2.6 yielded the best result for both CatBoost and ANN.

2. Principal Component Analysis

a. Though intuitively it tried to best explain the dataset through the combination of features and a reduced number of components, it performed consistently worse than the full feature rich data.
b. We assume its because both CatBoost and ANN – has mechanisms to not provide enough weights to features which do not impact the classification, consecutively the most important features were itself chosen to use as means for classification.

3. Power Transformation

a. PowerTransformer supports the Box-Cox transform and the Yeo-Johnson transform. The optimal parameter for stabilizing variance and minimizing skewness is estimated through maximum likelihood.
b. Box-Cox requires input data to be strictly positive, while Yeo-Johnson supports both positive or negative data. By default, zero-mean, unit-variance normalization is applied to the transformed data.
c. Because of the above advantages, we were able to utilize it, irrespective of kind of data – the best kind of transformation was chosen to minimize skewness and fit it in a Gaussian distribution, helped for a quick transformation for the entire dataset.

4. Imputation Strategies

a. We experimented with KNN Impuation, Random Forest Imputation and MICE Imputation as well as Iterative Imputer (based on Regressing the missing values based on other data vales)
b. Iterative Imputer seemed logical initially but failed to provide the values under the constraint as per business logic for each feature. It provided negative values which did not fit the column constraints.
c. KNN Imputer was chosen next – but due to memory constraints on such huge data – the proposition was let go.
d. Finally, MICE Forest yielded the most optimal results for quick imputation across the columns with missing values.

Model Testing – on AMEX Leader Board

Team_404 (CatBoost – weight: 2.6)

Score _ 60.42% | Rank _ 13

Leaderboard

The Leaderboard is an open avenue to test out various models that you may build and also to compare as to how well you are faring against competition.

Rank	Team Name	Score	Entries	Best Submission
13	Team_404 Jump to me	60.42%	19	12:04 pm | December 13, 2022

Submission History close

1	December 15, 2022	12:26 am	60.01%
2	December 14, 2022	12:20 am	59.66%
3	December 14, 2022	12:02 am	60.27%
4	December 13, 2022	12:04 pm	60.42%
5	December 13, 2022	11:42 am	59.48%
6	December 12, 2022	12:18 am	59.56%
7	December 12, 2022	12:03 am	60.13%
8	December 11, 2022	12:24 am	60.04%
9	December 11, 2022	12:03 am	58.61%

Rank	Team Name	Score	Entries	Best Submission
13	Team_404 Jump to me	60.42%	19	12:04 pm | December 13, 2022 History
1	FirstDegreeBurn (IIT Madras)	61.17%	11	11:56 pm | December 14, 2022
2	ThirdDegreeBurn (IIT Madras)	61.07%	16	09:38 pm | December 10, 2022
3	winners_pothumukku (IIT Madras)	61.06%	24	11:32 pm | December 14, 2022
4	DataScience_Novices (IIT Madras)	61.06%	10	11:09 pm | December 14, 2022
5	RANS (IIT Madras)	61.04%	25	06:05 pm | December 13, 2022
6	wwtpts (IIT Madras)	60.99%	21	10:09 pm | December 14, 2022
7	ArtificialHackers (IIT Madras)	60.94%	11	12:38 am | December 11, 2022
8	Brute_Force (IIT Madras)	60.85%	13	06:25 pm | December 14, 2022
9	mostly_just_BT (IIT Madras)	60.79%	14	01:55 pm | December 08, 2022
10	HVSS (IIT Madras)	60.71%	21	11:12 pm | December 09, 2022

With a difference of 0.65% in leaderboard from the 1st position, we would love to learn and continue experimenting, on how to improve on the models and yield the best result possible.

THANK YOU