****

**International Institute of Information Technology**

AI511 Machine Learning

***Project: It’s a Fraud***

Submitted by:  ***TEAM ENIGMA***

**Team Members :**

| Sajal Gupta | MT2022096 |
| --- | --- |
| Darshitkumar Pipariya | MT2022035 |

**Table of Content**

[**1. Introduction**](#_xf66v2n31g07) **3**

[1.1 Problem Statement](#_azbgtrmel3us) 3

[1.2 Overview](#_yagkny96jio6) 3

[**2. Dataset Description**](#_pipqkv1ncj68) **4**

[2.1 Files](#_ciqs8okkwdd) 4

[2.2 Columns](#_2fddymciv9zk) 4

[**3. Exploratory Data Analysis**](#_ed8zsdnts6nf) **5**

[3.1 Object Type columns](#_sfd11sdsi8fi) 6

[3.1.1 ProductCD](#_661f5d3p6amt) 6

[3.1.2 card4](#_2skgkynbrh4g) 7

[3.1.3 card6](#_kz9usj4lvqw2) 8

[3.1.4 P\_emaildomain](#_xlibp82kuf83) 9

[3.1.5 M6](#_9xej0r627o32) 10

[3.2 Numerical columns](#_ssvgkoap1xqz) 13

[3.2.1 card1, card2, card3, card5](#_wtpogn5rj6vs) 13

[3.2.2 addr1 and addr2](#_5ix7b23bz1vh) 18

[3.2.3 D1, D4, D10, D15](#_w9vvcsw9vldo) 20

[3.2.4 C1-C14 columns](#_ju8zxq50uuc5) 25

[3.2.5 Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations.](#_yez9wnfxtt83) 26

[**4. Training Models Evaluation**](#_tgo3a5rjum55) **28**

[**Conclusion**](#_8t2foqg2jtvv) **32**

[**References**](#_dy3k3fdlqnqo) **33**

# 1. Introduction

## 1.1 Problem Statement

The traditional approaches used for detecting transaction fraud involve manual monitoring by humans which involves interaction with the cardholders. This approach is not efficient as it not only consumes a lot of time but is also quite expensive in terms of the resources it uses to detect the fraudulence of a transaction. Moreover, in the real world the majority of the transactions being manually monitored turn out to be legit and only a few of them turn out to be fraudulent which wastes a lot of time and energy.

Hence our objective is to develop an automated screening system that requires minimalistic human intervention to detect the legitimacy of transactions using Machine Learning approaches.

## 1.2 Overview

Steps followed to achieve our objective:

* Here the goal is to create a classification model that can predict whether the transaction is fraudulent or not.
* Understand Dataset and perform the EDA on Data and preprocessing of Data.
* Train the Model that has the best accuracy, F1-score and AUC score for that do the following
  + Use a different classification model
  + Use sampling techniques,
  + Use cross-validation
  + Use Perform hyperparameter tuning for all models

# 

# 2. Dataset Description

## 2.1 Files

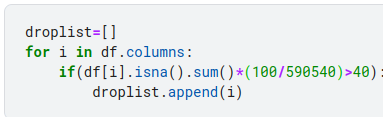
* **train.csv**: This training dataset comprises the transaction details, identity data and target label for fraud transactions.
* **test.csv**: same as train dataset but without target label.
* **sample\_submission.csv** - a sample submission file in the correct format

## 2.2 Columns

* **TransactionDT**: timedelta from a given reference DateTime (not an actual timestamp) “TransactionDT "corresponds to the number of seconds in a day.
* **TransactionAMT**: transaction payment amount in USD
* **ProductCD**: product code, the product for each transaction
* **card1 - card6**: payment card information, such as card type, card category, issue bank, country, etc.
* **addr**: address  
  “both addresses are for purchaser  
  addr1 as billing region  
  addr2 as billing country”
* **dist**: distances between (not limited) billing address, mailing address, zip code, IP address, phone area, etc.”
* **P\_ and (R\_\_) emaildomain**: purchaser and recipient email domain  
  “ certain transactions don't need a recipient, so R\_emaildomain is null.”
* **C1-C14**: counting, such as how many addresses are found to be associated with the payment card, etc.
* **D1-D15**: timedelta, such as days between the previous transaction, etc.
* **M1-M9**: match, such as names on card and address, etc.
* **Vxxx**: Vesta engineered rich features, including ranking, counting, and other entity relations.
* **id01 to id11** are numerical features for identity.  
  Other columns such as network connection information (IP, ISP, Proxy, etc), and digital signature (UA/browser/os/version, etc) associated with transactions are also present

# 3. Exploratory Data Analysis

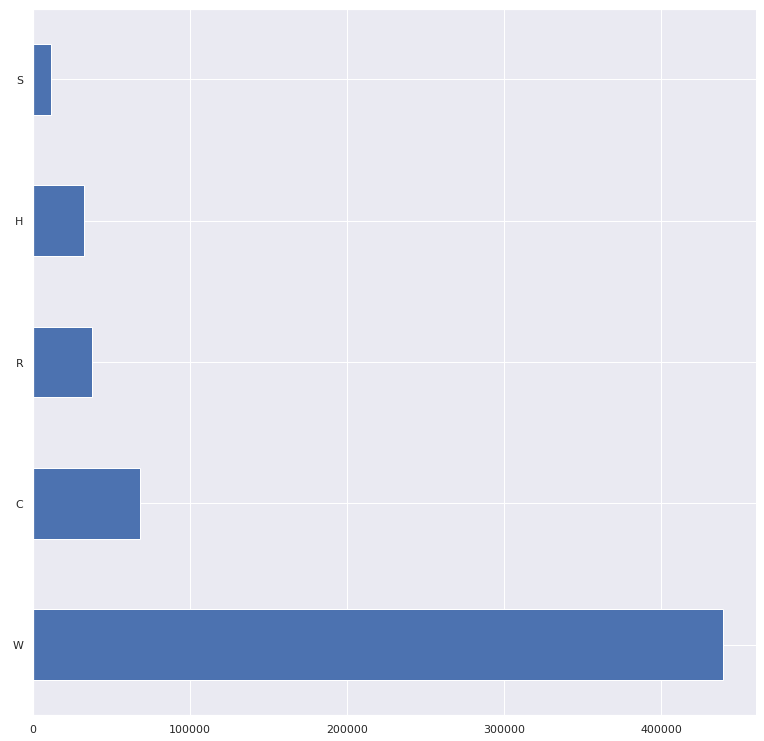
* In train data we have 434 columns and 442905 rows and In test data, we have 433 columns and 147635 rows. We have concatenated both train and test data column-wise to perform the same data processing.
* We have 400 columns of float64 type, 3 columns of int64 and 31 columns of object type.
* Memory usage is 1.9+ GB and to run a model on it we have to reduce it.
* We have 232 columns that have a null value of more than 40%. we are dropping them now we have 202 columns left.



## 3.1 Object Type columns

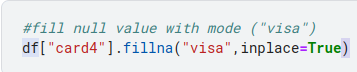
### 3.1.1 ProductCD

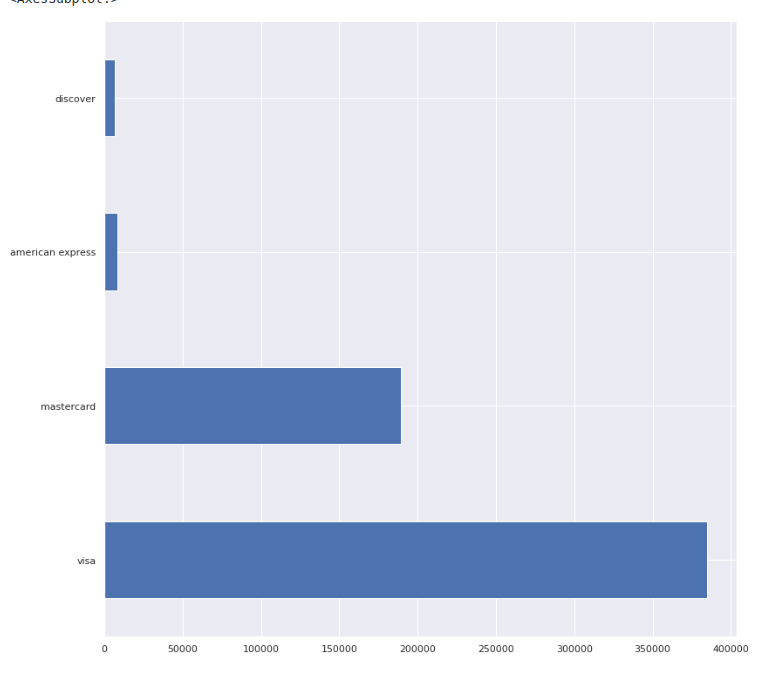
It does not have any null value. It has 5 categories. We have used a one-hot encoder for this column.

****

### 3.1.2 card4

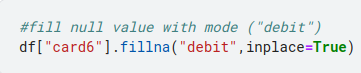
It has 1577 null values.to fill null we have used the mode of card4.it has 4 categories .we have used a one-hot encoder.

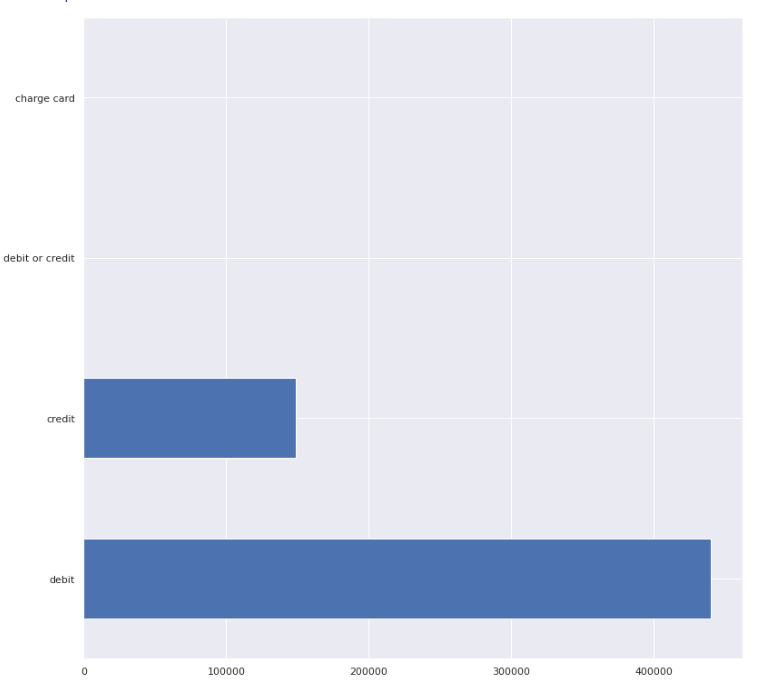


.

### 3.1.3 card6

It has 1571 null values.to fill null we have used the mode of card4.it has 4 categories. We have used a one-hot encoder.





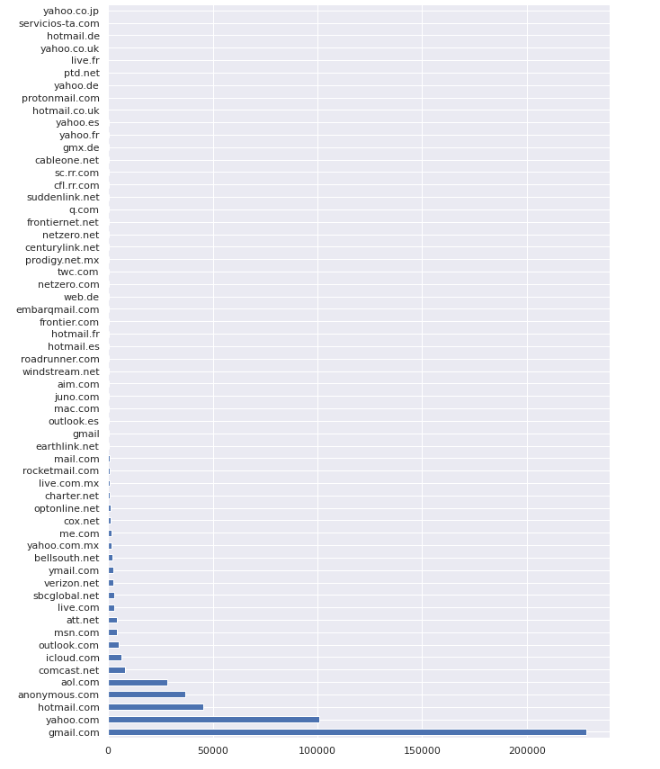
### 3.1.4 P\_emaildomain

It has 94456 null values and 55 categories. Here we fill null values according to category distribution and we are combining 18 categories which do not have fraud transactions as goodmail. Now we have 48 different categories.to encode it we are using a one-hot encoder.







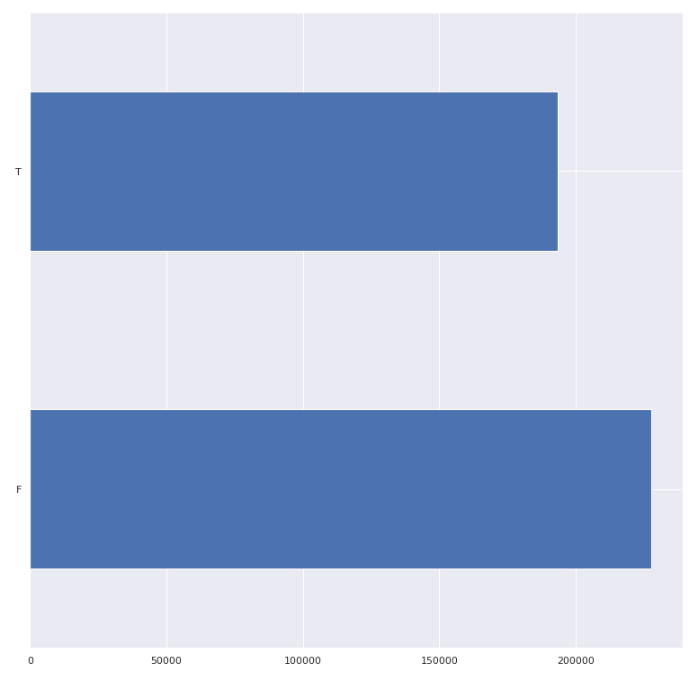


### 

### 3.1.5 M6

It has 169360 null values and 2 categories. Here we fill null values according to category distribution. and to encode it we are using a label encoder.





## 

## 3.2 Numerical columns

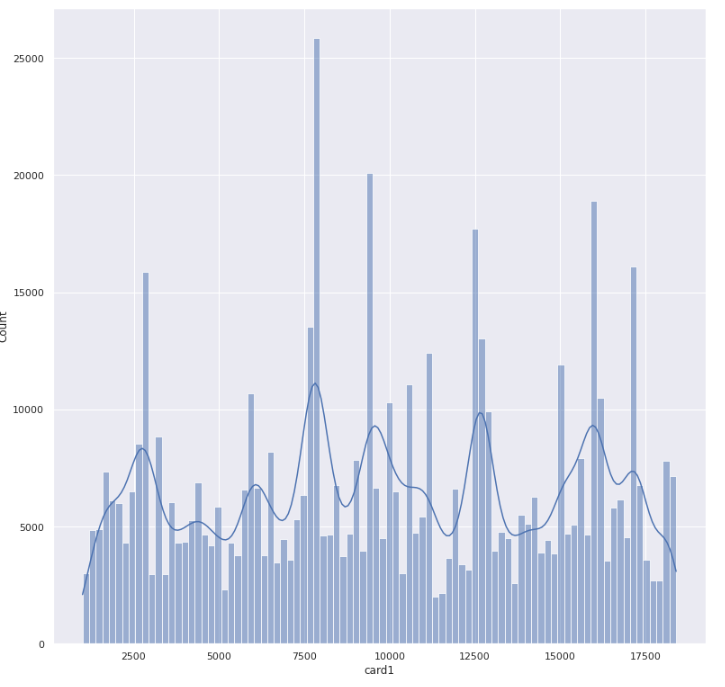
### 3.2.1 card1, card2, card3, card5

card1 has no null value.

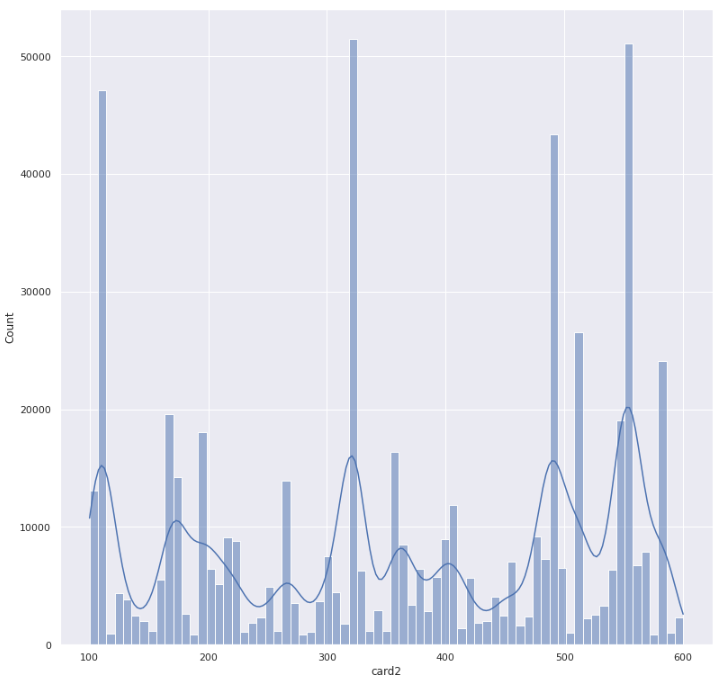
card2 , card3 , card4 columns have 8933,1565,4259 null values respectively.

we will fill the null value by the distribution of the numerical values as you can see below graphs. Here numerical value represents the categories.

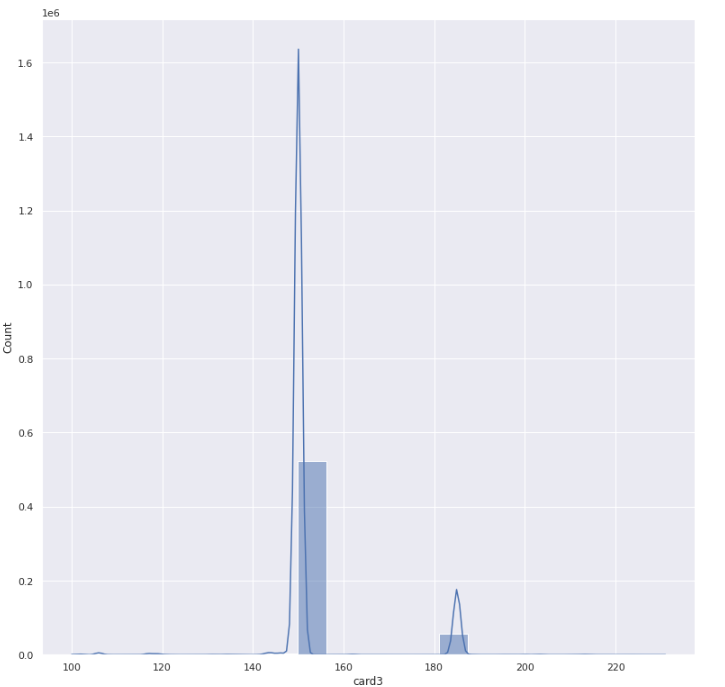




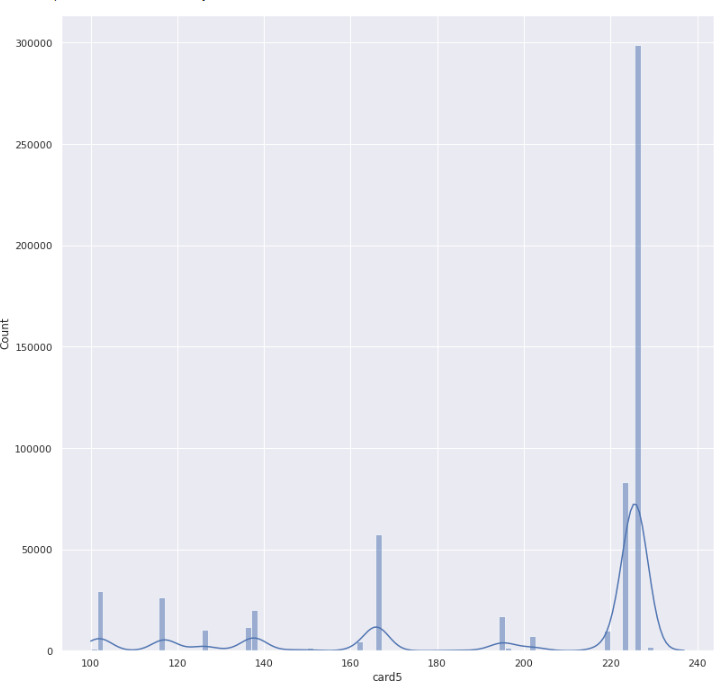
**card1**



**card2**



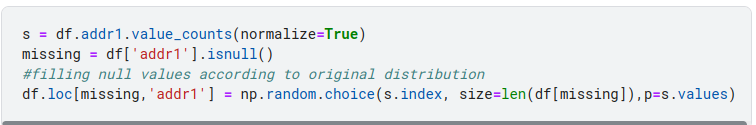
**card3**

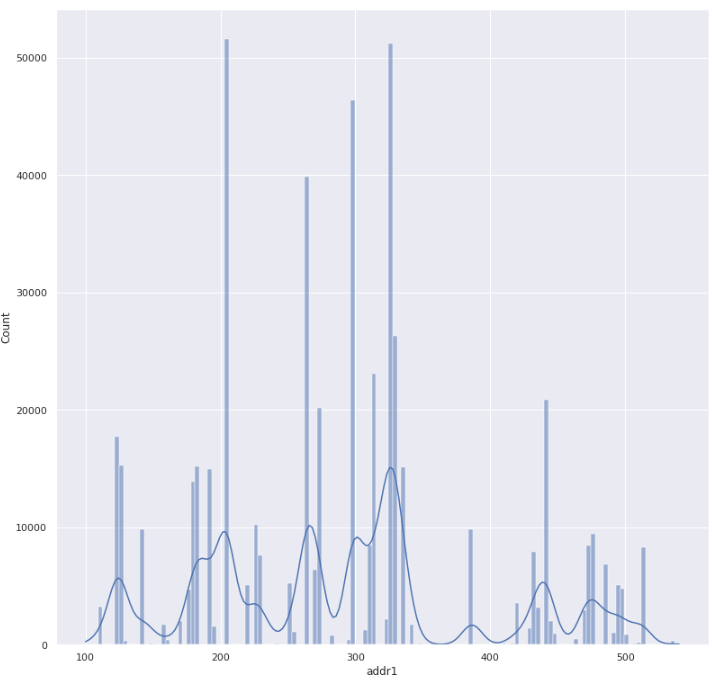


**card5**

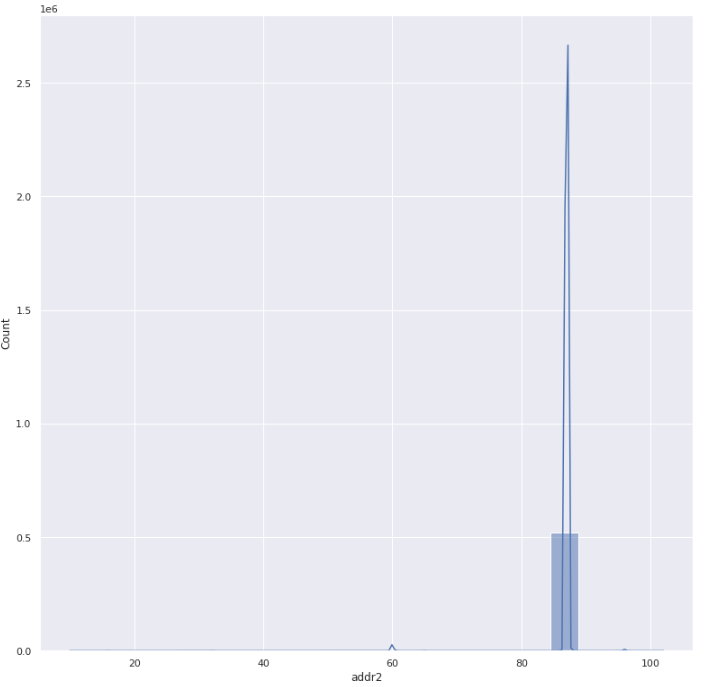
### 3.2.2 addr1 and addr2

Both addr1 and addr2 columns have 65706 null values.we will fill the null value by the distribution of the numerical values as you can see in the below graphs. Here numerical value represents the categories.





**addr1**

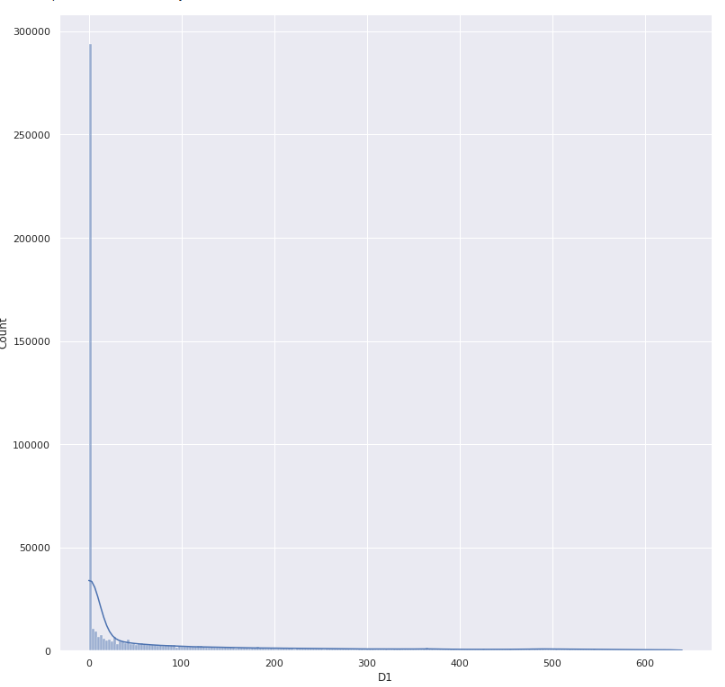
****

**addr2**

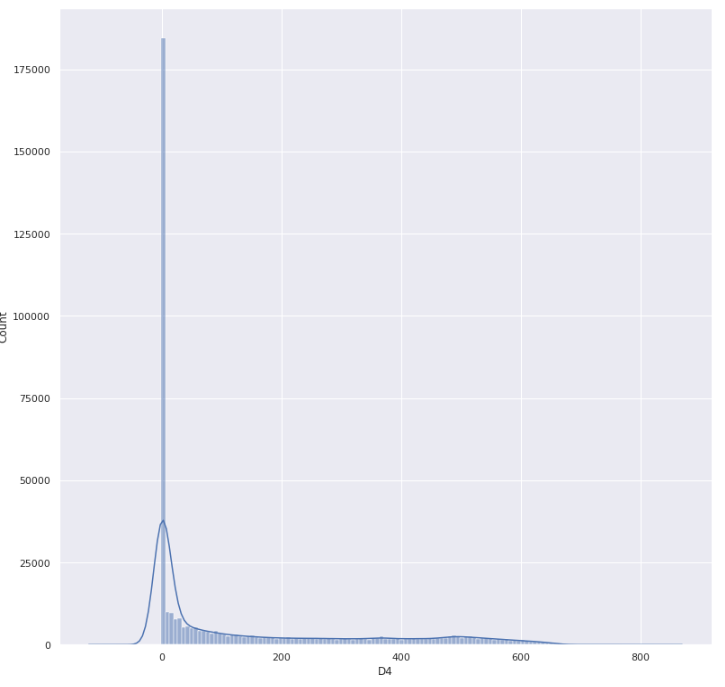
### 3.2.3 D1, D4, D10, D15

D1, D4, D10 and D15 columns have 1269,168922,76022,89133 null values respectively.we will fill the null value by the mode of each column.

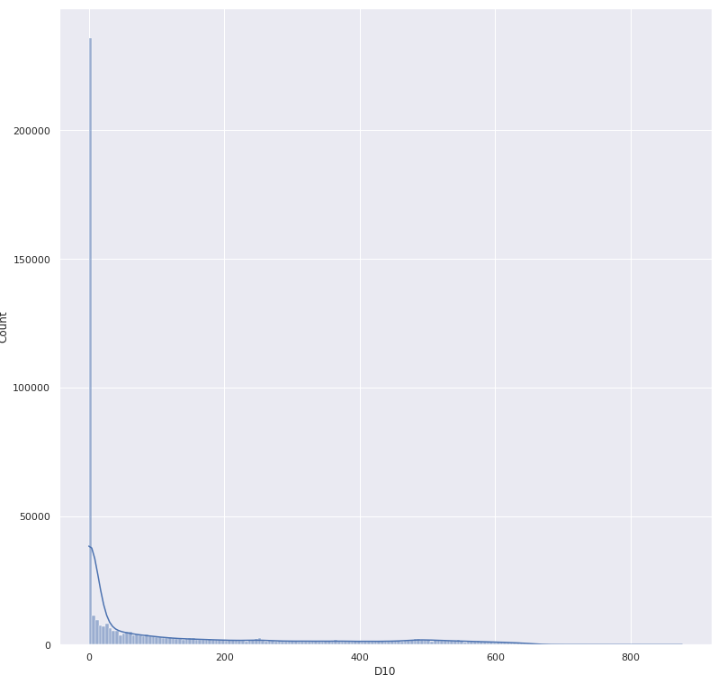




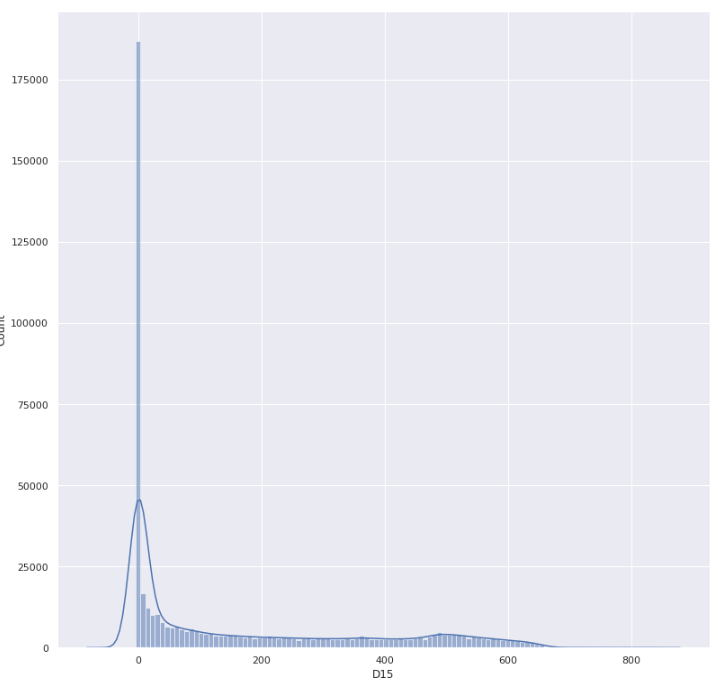
**D1**

****

**D4**

****

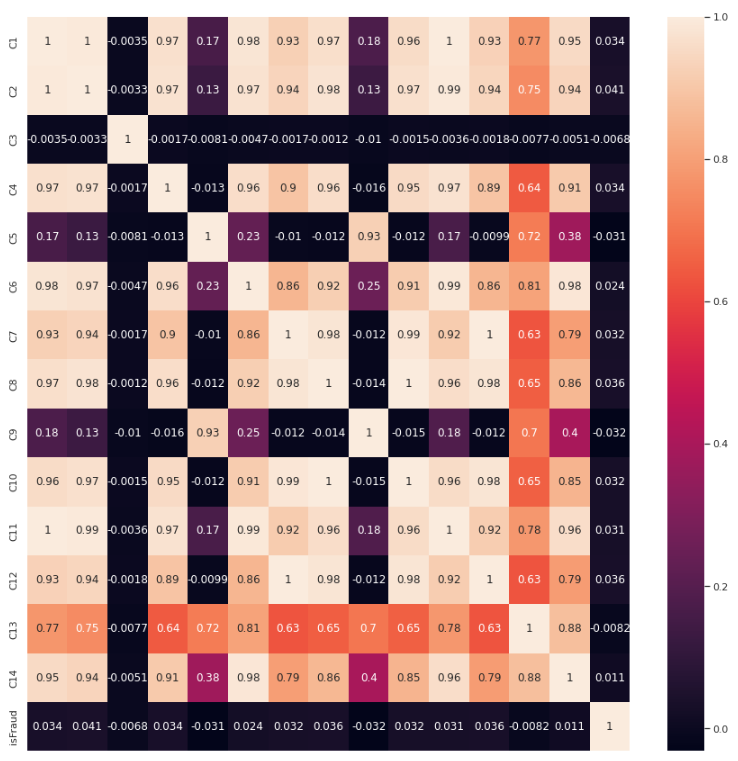
**D10**

****

**D15**

### 3.2.4 C1-C14 columns

c1-c14 have no null values. Some of the columns have high correlations as we can see in the graph below. we will drop the columns which have a correlation greater than 90%. Ex. C2, C4, C6, C7, C8, C9, C10, C11, C12, C14



### 3.2.5 Vxxx: Vesta engineered rich features, including ranking, counting, and other entity relations.

V12-V34 has 76073 null values.

V35-V52 has 168969 null values.

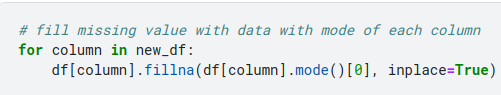
V53-V74 has a 77096 null value.

V75-V94 has an 89164 null value.

V95-V137 has 314 null values. V279, V280, V284-V287, V290-V295, V297-V299, V302-V312, V316-V321 have 12 null values.

V281-V283, V288-V289, V296, V300, V301, and V313-V315 has 1269 null values.

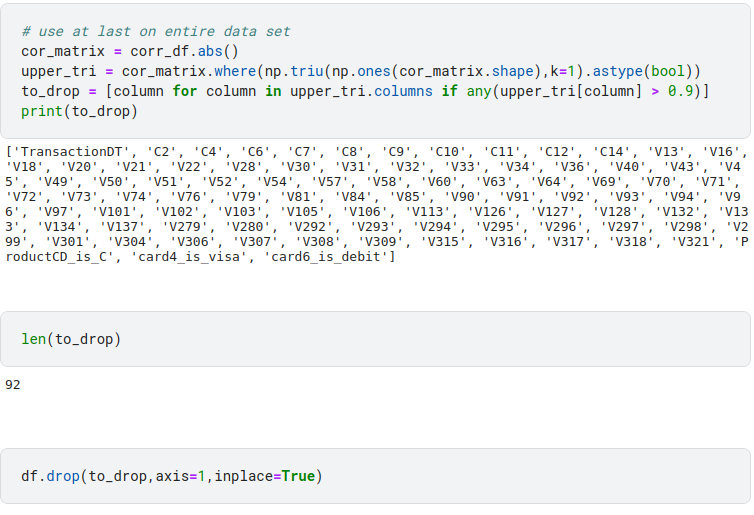




We are filling null values using the mode of the columns. There are 86 columns which have correlations greater than 90. We will drop them.

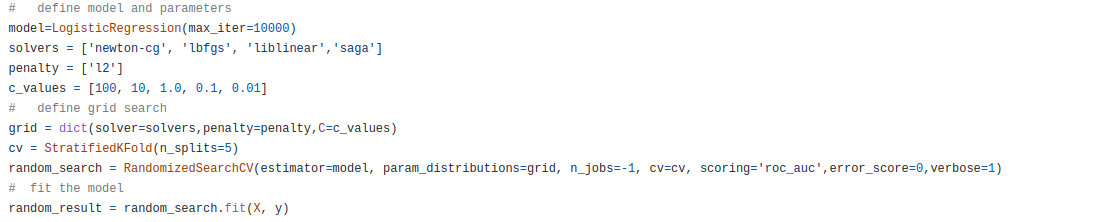
Ex.

'V13', 'V16', 'V18', 'V20', 'V21', 'V22', 'V28', 'V30', 'V31', 'V32', 'V33', 'V34', 'V36', 'V40', 'V42', 'V43', 'V45', 'V48', 'V49', 'V50', 'V51', 'V52', 'V54', 'V57', 'V58', 'V59', 'V60', 'V63', 'V64', 'V68', 'V69', 'V70', 'V71', 'V72', 'V73', 'V74', 'V76', 'V79', 'V80', 'V81', 'V84', 'V85', 'V88', 'V89', 'V90', 'V91', 'V92', 'V93', 'V94', 'V96', 'V97', 'V101', 'V102', 'V103', 'V105', 'V106', 'V113', 'V126', 'V127', 'V128', 'V132', 'V133', 'V134', 'V137', 'V279', 'V280', 'V292', 'V293', 'V294', 'V295', 'V296', 'V297', 'V298', 'V299', 'V301', 'V302', 'V304', 'V306', 'V307', 'V308', 'V309', 'V315', 'V316', 'V317', 'V318', 'V321'

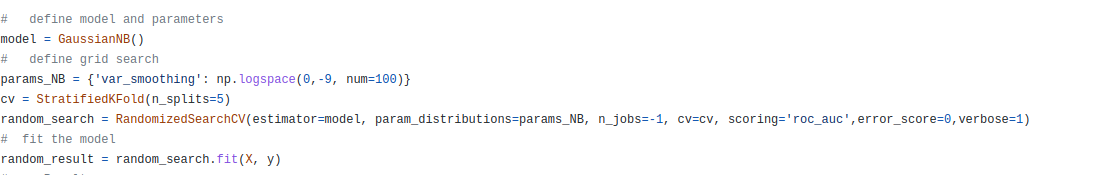


# 4. Training Models Evaluation

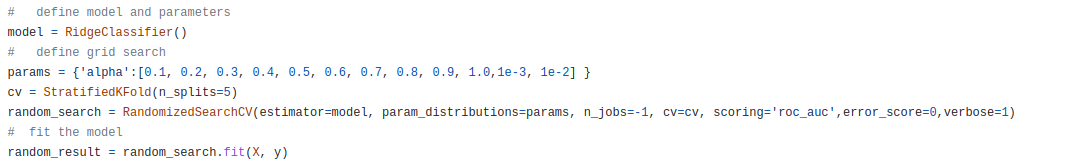
Models are evaluated on ROC-AUC Score. For hyperparameter tuning, we have used RandomSearchCV except for XGboost. In XGboost we have used the hyperopt library.

[](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/logisticregressionkfoldperametertuning.py)

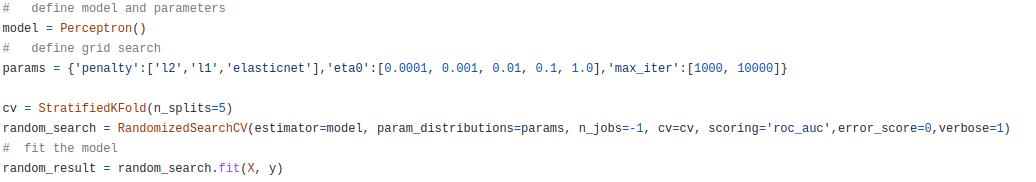
Logistic

[](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/naivebayeskfoldprametertuning.py)

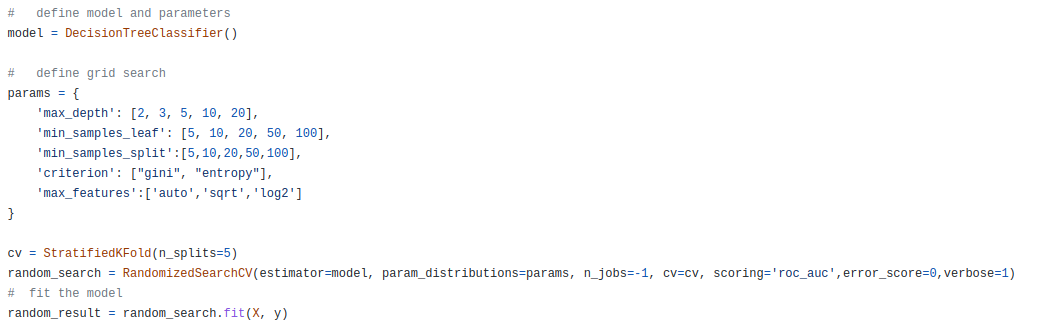
Naive Bayes

[](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/ridgeclassifiercv.py)

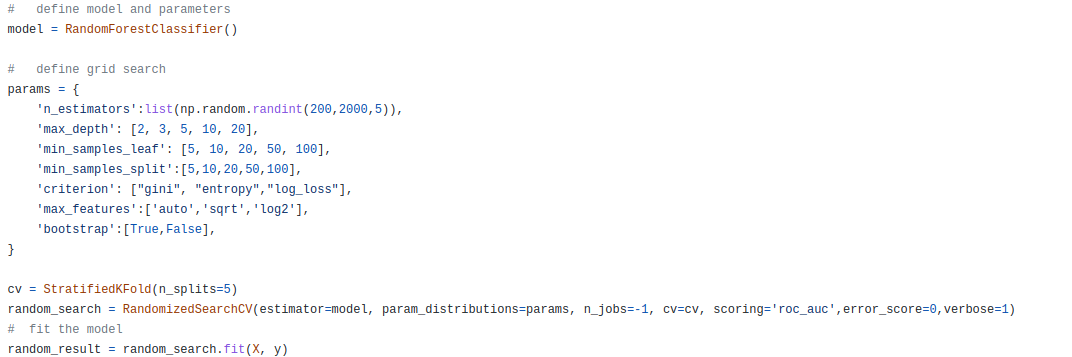
Ridge Classifier

[](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/perceptron.py)

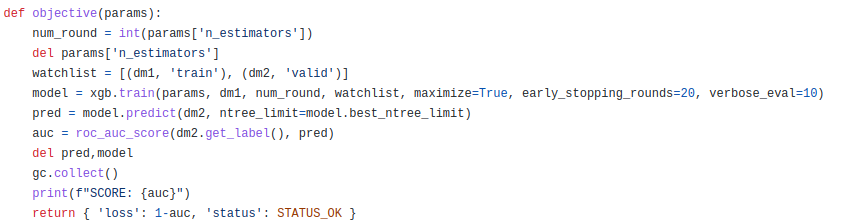
Perceptron

[](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/decisiontreekfoldperametertuning.py)

Decision Tree

[](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/randomforestkfoldperameterwithtuning.py)

Random Forest

[](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/xgboost.py)

[](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/xgboost.py)

XGboost

| **Model** | **Hyper Parameters** | **Value** | **Model Score** |
| --- | --- | --- | --- |
| [**Logistic regression classifier**](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/logisticregressionkfoldperametertuning.py) | **max\_iter** | **1000** | **0.57204** |
| **solver** | **libliner** |
| **penalty** | **l2** |
| **c** | **0.01** |
| [**Naive Bayes**](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/naivebayeskfoldprametertuning.py) | **var\_smoothing** | **0.0533669923120631** | **0.70151** |
| [**Ridge Classifier**](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/ridgeclassifiercv.py) | **alpha** | **1** | **0.53441** |
| [**Perceptron**](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/perceptron.py) | **penalty** | **l1** | **0.52756** |
| **max\_iter** | **1000** |
| **eta0** | **0.0001** |
| [**Decision Tree**](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/decisiontreekfoldperametertuning.py) | **min\_sample\_split** | **5** | **0.62807** |
| **min\_samples\_leaf** | **50** |
| **max\_feature** | **sqrt** |
| **max\_depth** | **20** |
| **criterion** | **gini** |
| [**Random Forest**](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/randomforestkfoldperameterwithtuning.py) | **n\_estimators** | **1635** | **0.56897** |
| **min\_samples\_split** | **5** |
| **min\_samples\_leaf** | **100** |
| **max\_feature** | **sqrt** |
| **max\_depth** | **10** |
| **criterion** | **entropy** |
| **bootstrap** | **TRUE** |
|  |  |  |  |
| [**Xgboost**](https://github.com/Darshitpipariya/ML_project_Fruad_detection/blob/main/xgboost.py) | **alpha** | **2** | **0.87369** |
| **colsample\_bytree** | **0.85** |
| **eta** | **0.125** |
| **gamma** | **0.75** |
| **lambda** | 1.8 |
| **max\_depth** | **12** |
| **n\_estimators** | **7** |
| **min\_child\_weight** | **500** |
| **scale\_pos\_weight** | **60** |
| **subsample** | **0.7** |
| **objective** | **binary:logistic** |
| **eval\_metric** | **auc** |
| **tree\_method** | **hist** |
| **bootstrap** | **gbtree** |

# 

# 

# 

# 

# 

# Conclusion

In this Project we had a large training dataset that contains columns with high NULL values.

To handle large NULL values we have dropped the columns, for others, we have filled with either mode or values according to distribution. We have also used a one-hot encoder and a label encoder to handle object-type features (columns).

We have used many classification models with cross-validation and parameter tuning using RandomSearch or hyperopt.

We got the best result in the XGboost model and parameter tuning using hyperopt.

Finally, the Xgboost model trained with the Hyperopt tuning gave the best possible result with an 87% ROC-AUC Score approximately.

# References

1. <https://towardsdatascience.com/using-machine-learning-to-detect-fraud-f204910389cf>
2. <https://medium.com/fuzz/machine-learning-classification-models-3040f71e2529>
3. <https://xgboost.readthedocs.io/en/stable/>
4. <https://neptune.ai/blog/hyperparameter-tuning-in-python-complete-guide>
5. <https://scikit-learn.org/stable/modules/cross_validation.html>
6. <https://machinelearningmastery.com/nested-cross-validation-for-machine-learning-with-python/>
7. <https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc>
8. <https://analyticsindiamag.com/understanding-the-basics-of-svm-with-example-and-python-implementation/>