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

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

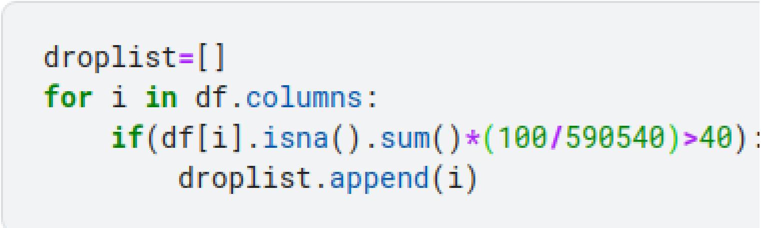
- **train.csv**: This training dataset comprises the transaction details, identity data and target label for fraud transactions.
- **test.csv**: same as train dataset but without target label.
- **sample_submission.csv** - a sample submission file in the correct format

2.2 Columns

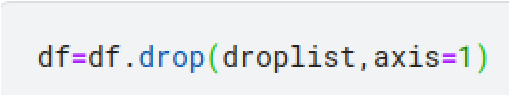
- **TransactionDT**: timedelta from a given reference DateTime (not an actual timestamp) “TransactionDT "corresponds to the number of seconds in a day.
- **TransactionAMT**: transaction payment amount in USD
- **ProductCD**: product code, the product for each transaction
- **card1 - card6**: payment 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.



```
droplist=[]
for i in df.columns:
    if(df[i].isna().sum()*(100/590540)>40):
        droplist.append(i)
```

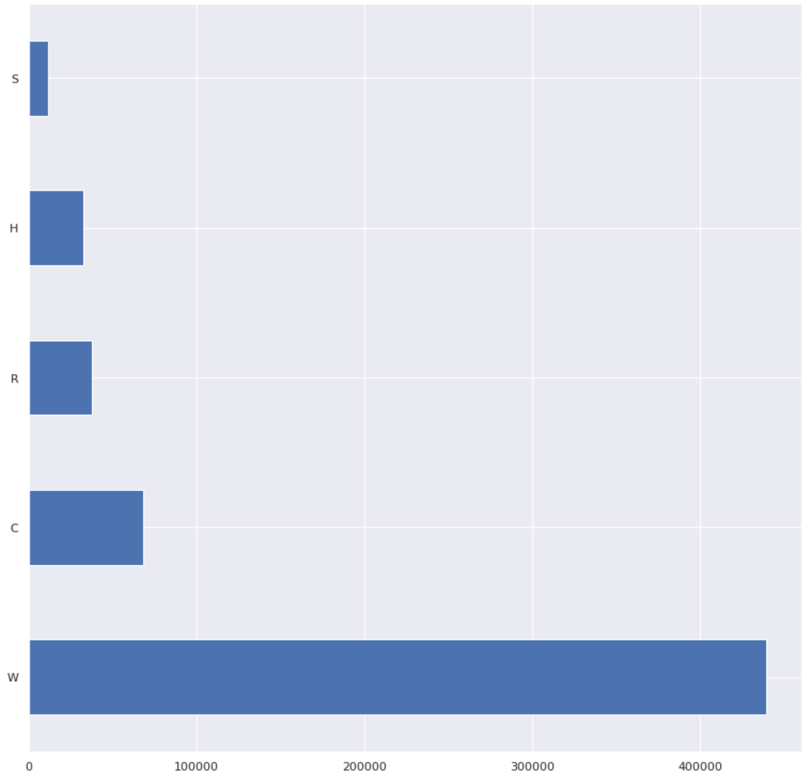


```
df=df.drop(droplist,axis=1)
```

3.1 Object Type columns

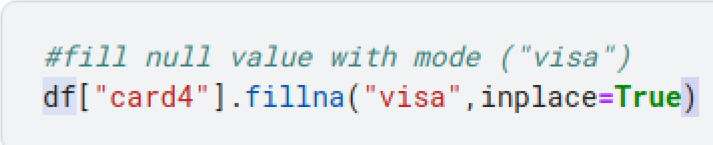
3.1.1 ProductCD

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

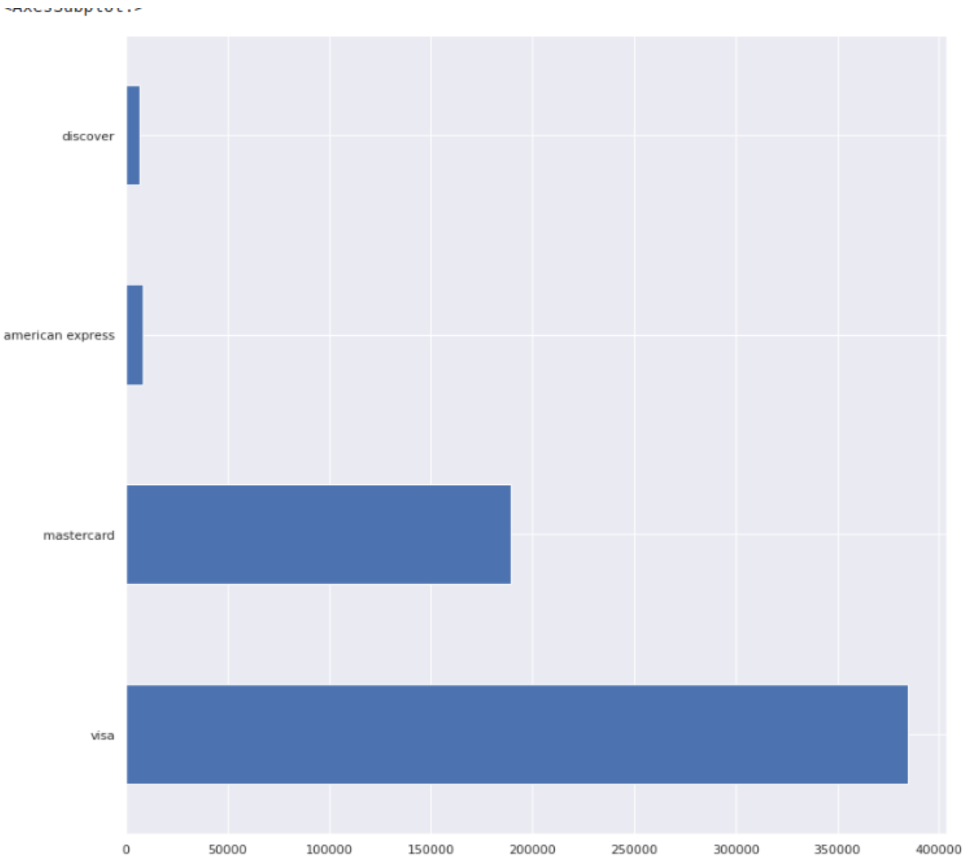


3.1.2 card4

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.



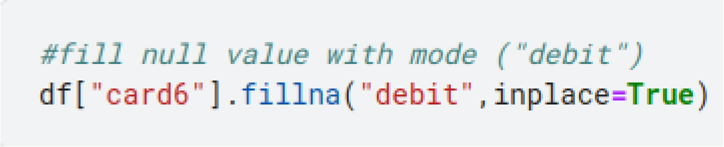
```python
#fill null value with mode ("visa")
df["card4"].fillna("visa",inplace=True)
```



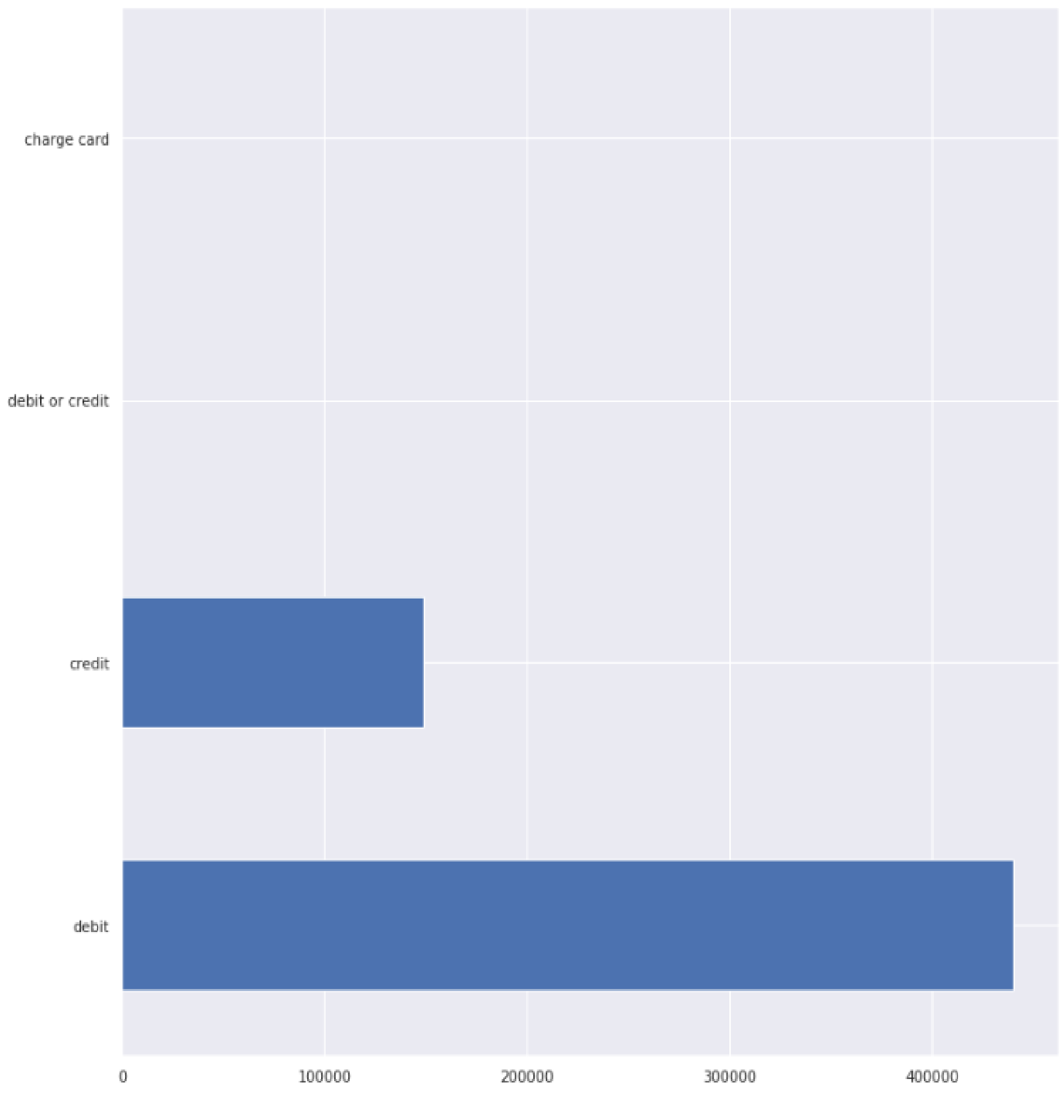
```
<AxesSubplot:>
```

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.

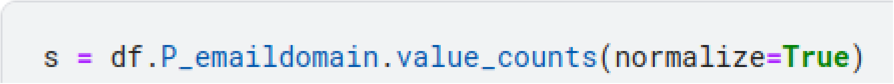


```
#fill null value with mode ("debit")
df["card6"].fillna("debit",inplace=True)
```

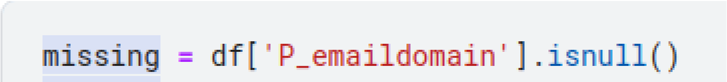


3.1.4 P_emaildomain

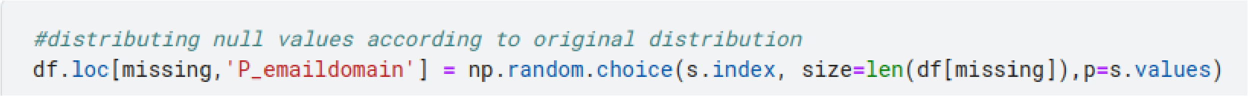
It has 94456 null values and 55 categories. Here we fill null values according to category distribution and we are combining 18 categories which do not have fraud transactions as goodmail. Now we have 48 different categories.to encode it we are using a one-hot encoder.



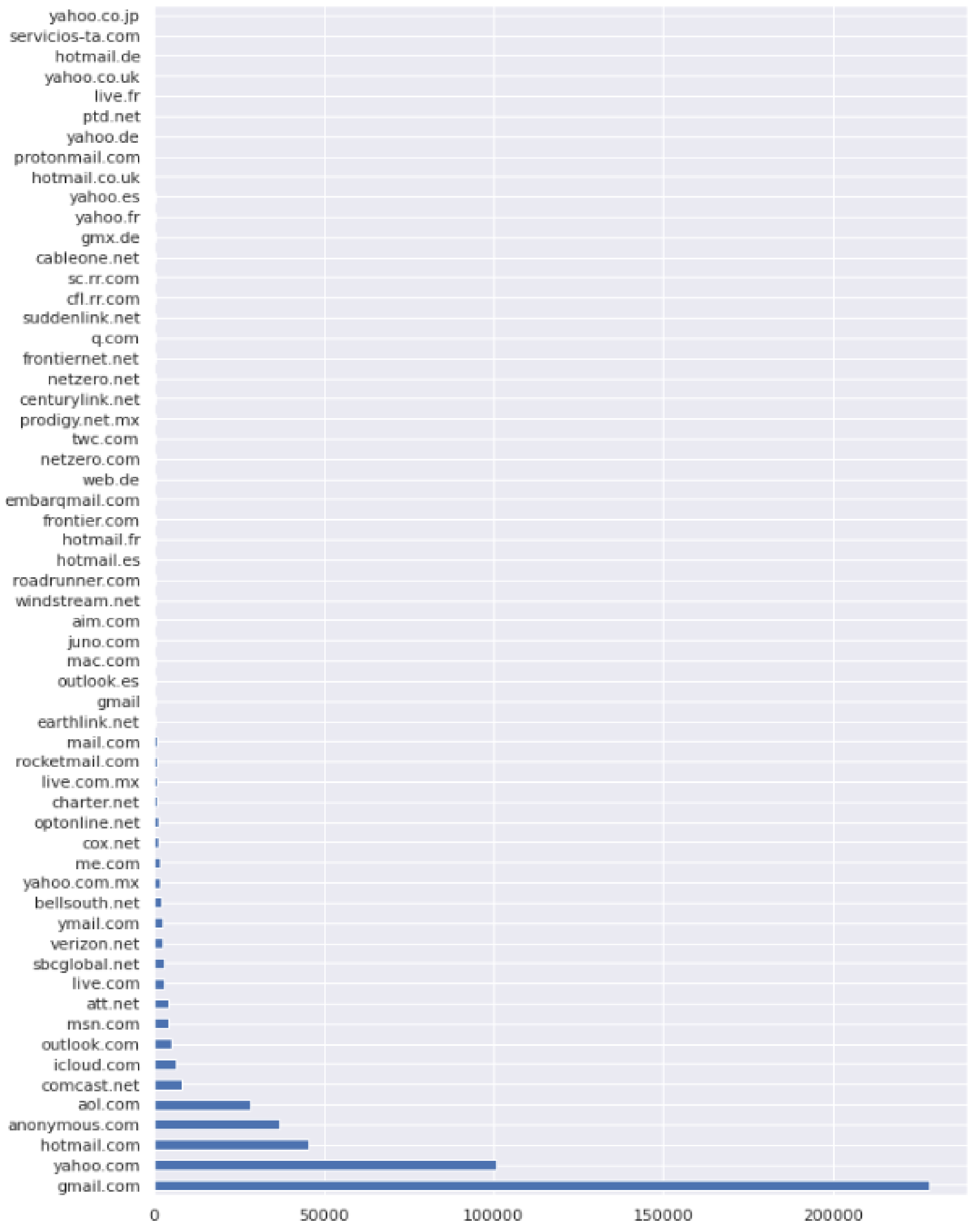
```
s = df.P_emaildomain.value_counts(normalize=True)
```



```
missing = df['P_emaildomain'].isnull()
```

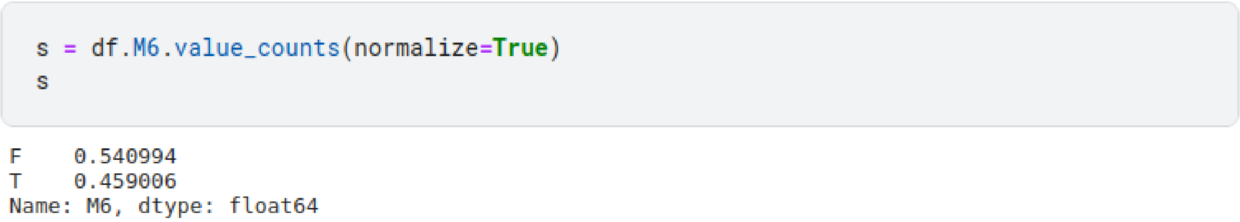


```
#distributing null values according to original distribution
df.loc[missing,'P_emaildomain'] = np.random.choice(s.index, size=len(df[missing]),p=s.values)
```


yahoo.co.jp
servicios-ta.com
hotmail.de
yahoo.co.uk
live.fr
ptd.net
yahoo.de
protonmail.com
hotmail.co.uk
yahoo.es
yahoo.fr
gmx.de
cableone.net
sc.rr.com
cfl.rr.com
suddenlink.net
q.com
frontiernet.net
netzero.net
centurylink.net
prodigy.net.mx
twc.com
netzero.com
web.de
embarqmail.com
frontier.com
hotmail.fr
hotmail.es
roadrunner.com
windstream.net
aim.com
juno.com
mac.com
outlook.es
gmail
earthlink.net
mail.com
rocketmail.com
live.com.mx
charter.net
optonline.net
cox.net
me.com
yahoo.com.mx
bellsouth.net
ymail.com
verizon.net
sbcglobal.net
live.com
att.net
msn.com
outlook.com
icloud.com
comcast.net
aol.com
anonymous.com
hotmail.com
yahoo.com
gmail.com
0
50000
100000
150000
200000

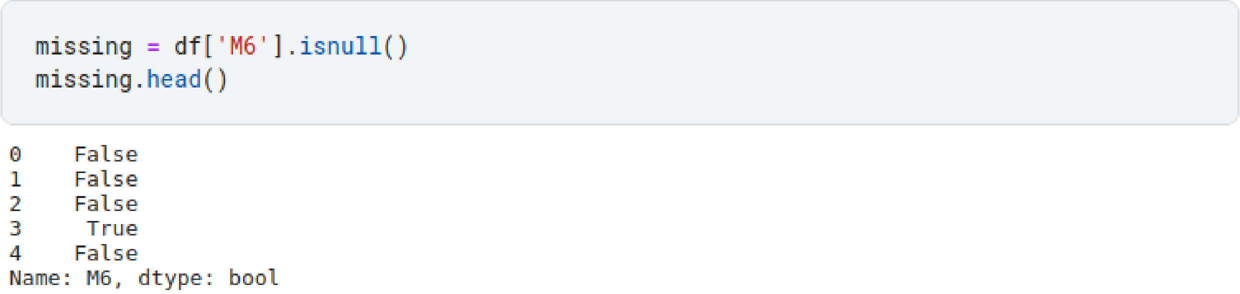
3.1.5 M6

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.



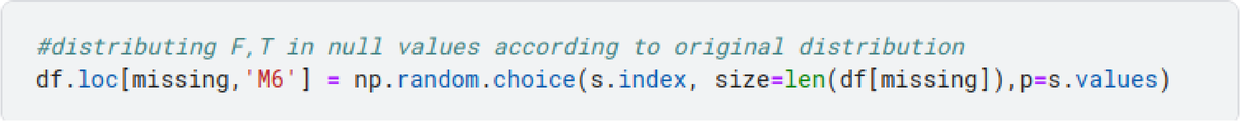
```
s = df.M6.value_counts(normalize=True)
s
```

```
F    0.540994
T    0.459006
Name: M6, dtype: float64
```

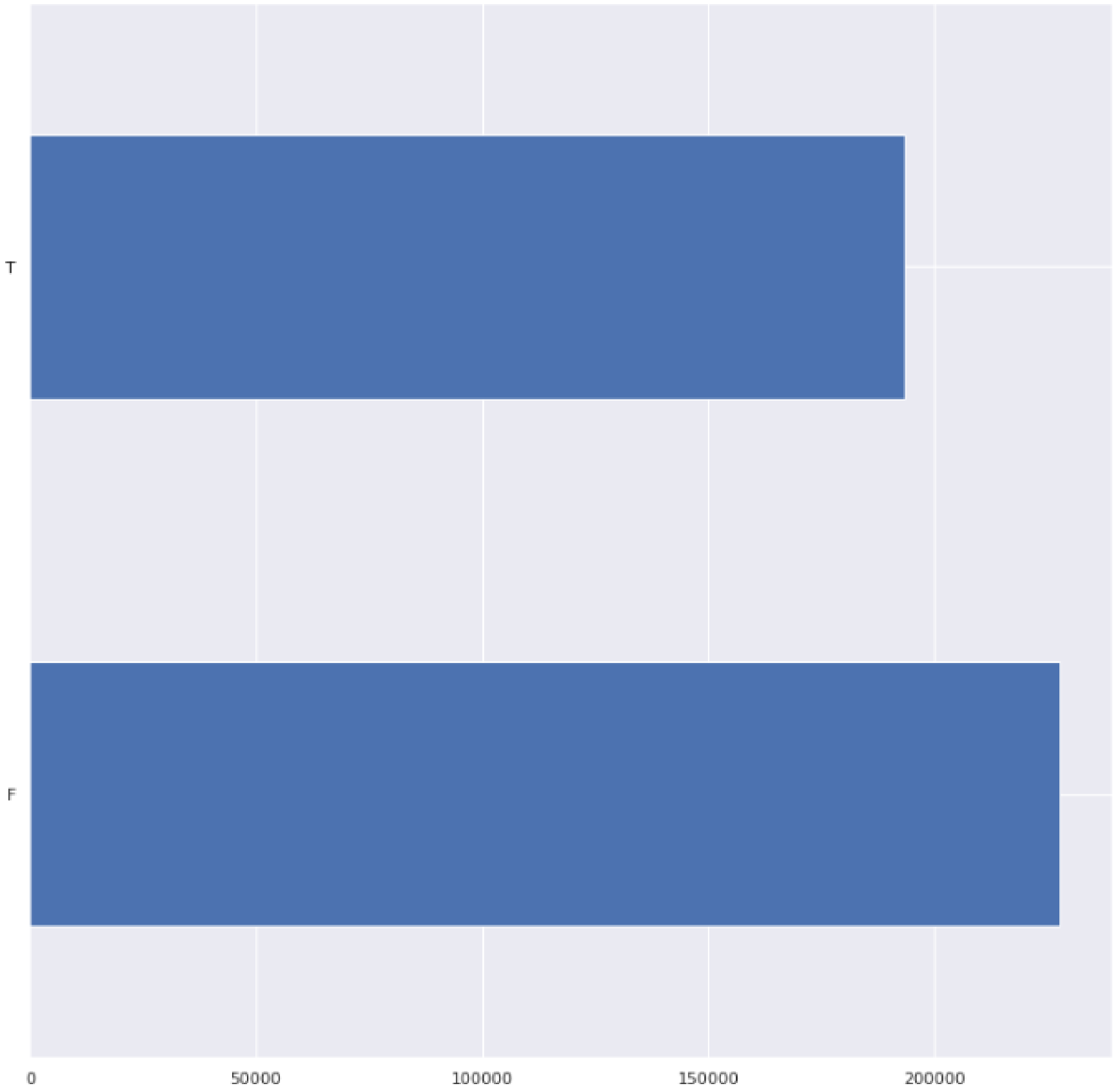


```
missing = df['M6'].isnull()
missing.head()
```

```
0    False
1    False
2    False
3     True
4    False
Name: M6, dtype: bool
```



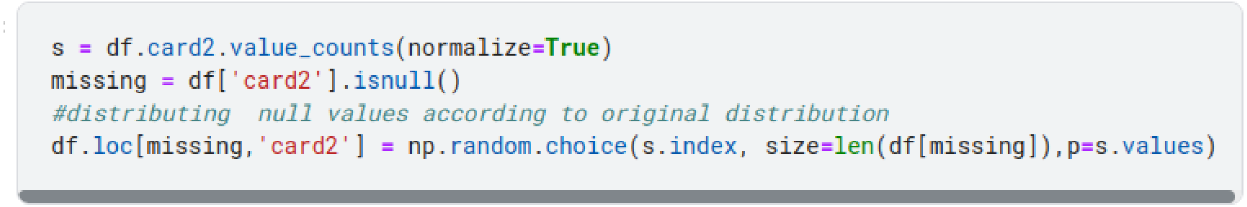
```
#distributing F,T in null values according to original distribution
df.loc[missing,'M6'] = np.random.choice(s.index, size=len(df[missing]),p=s.values)
```


T
F
0
50000
100000
150000
200000

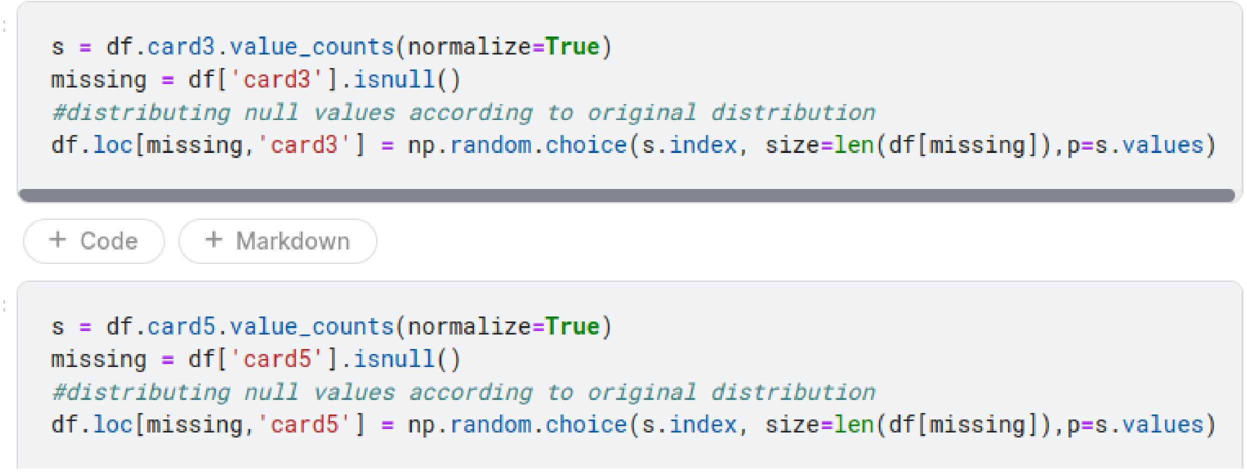
3.2 Numerical columns

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.

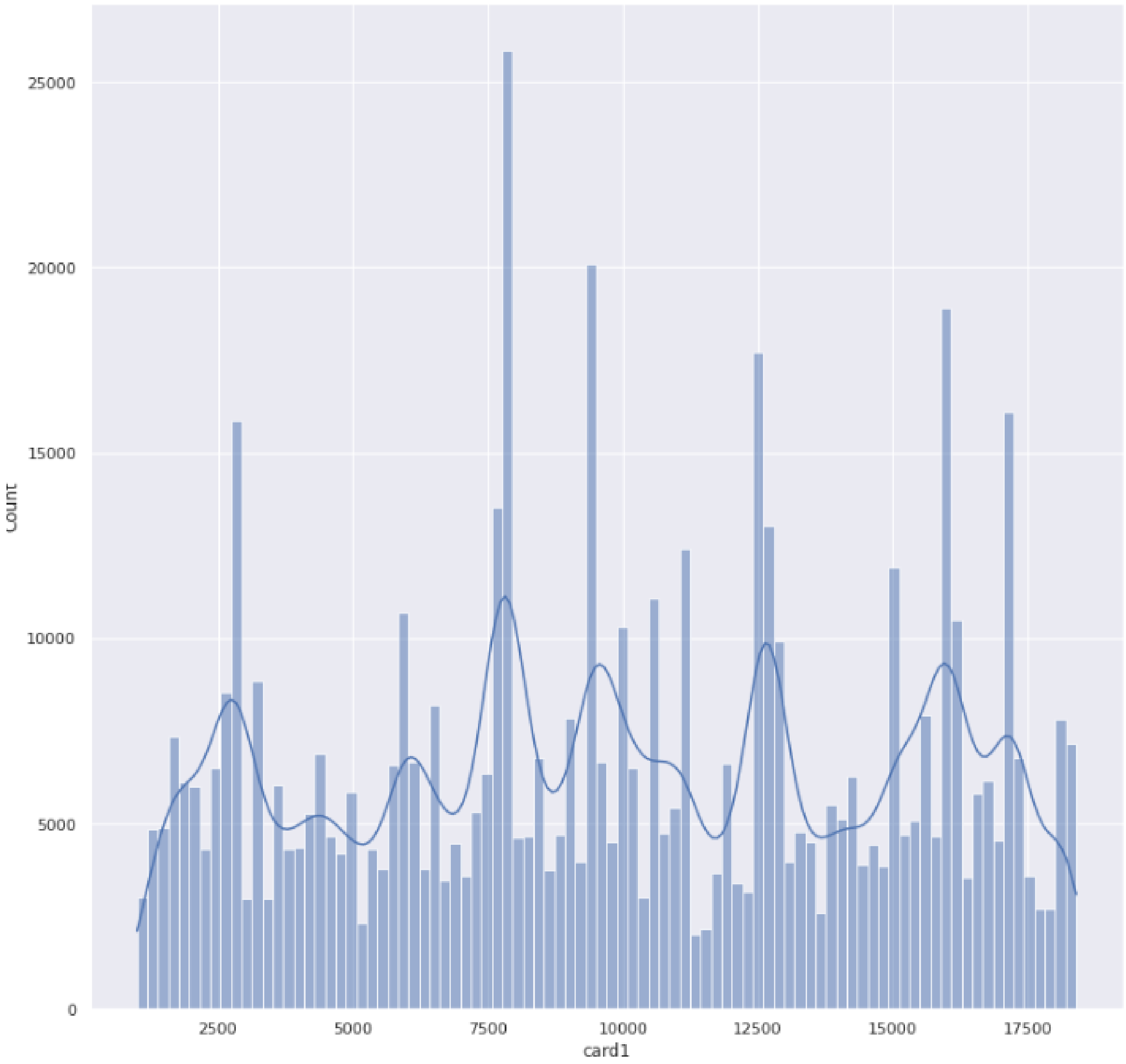


```
s = df.card2.value_counts(normalize=True)
missing = df['card2'].isnull()
#distributing  null values according to original distribution
df.loc[missing,'card2'] = np.random.choice(s.index, size=len(df[missing]),p=s.values)
```

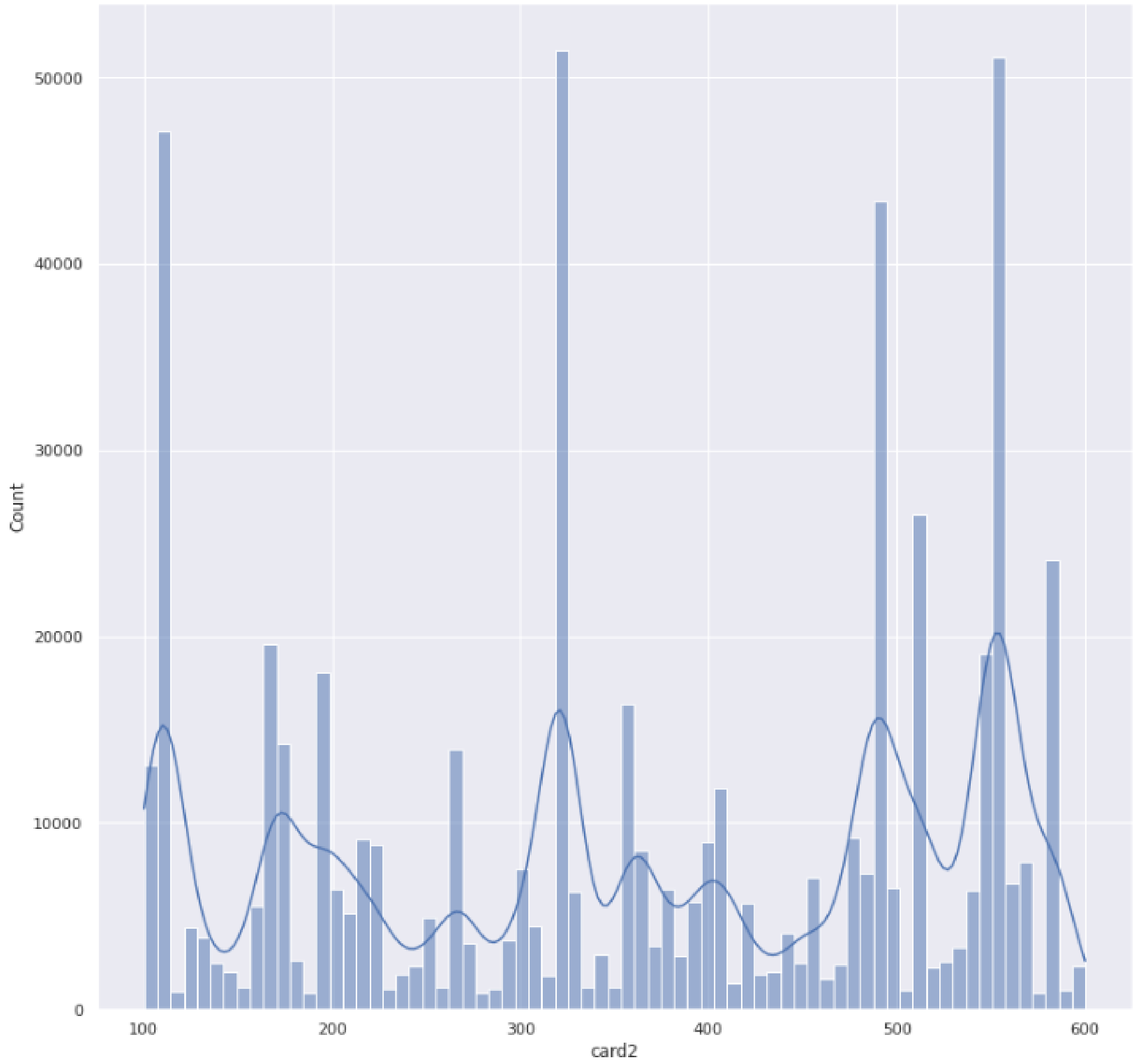


```
s = df.card3.value_counts(normalize=True)
missing = df['card3'].isnull()
#distributing null values according to original distribution
df.loc[missing,'card3'] = np.random.choice(s.index, size=len(df[missing]),p=s.values)
```

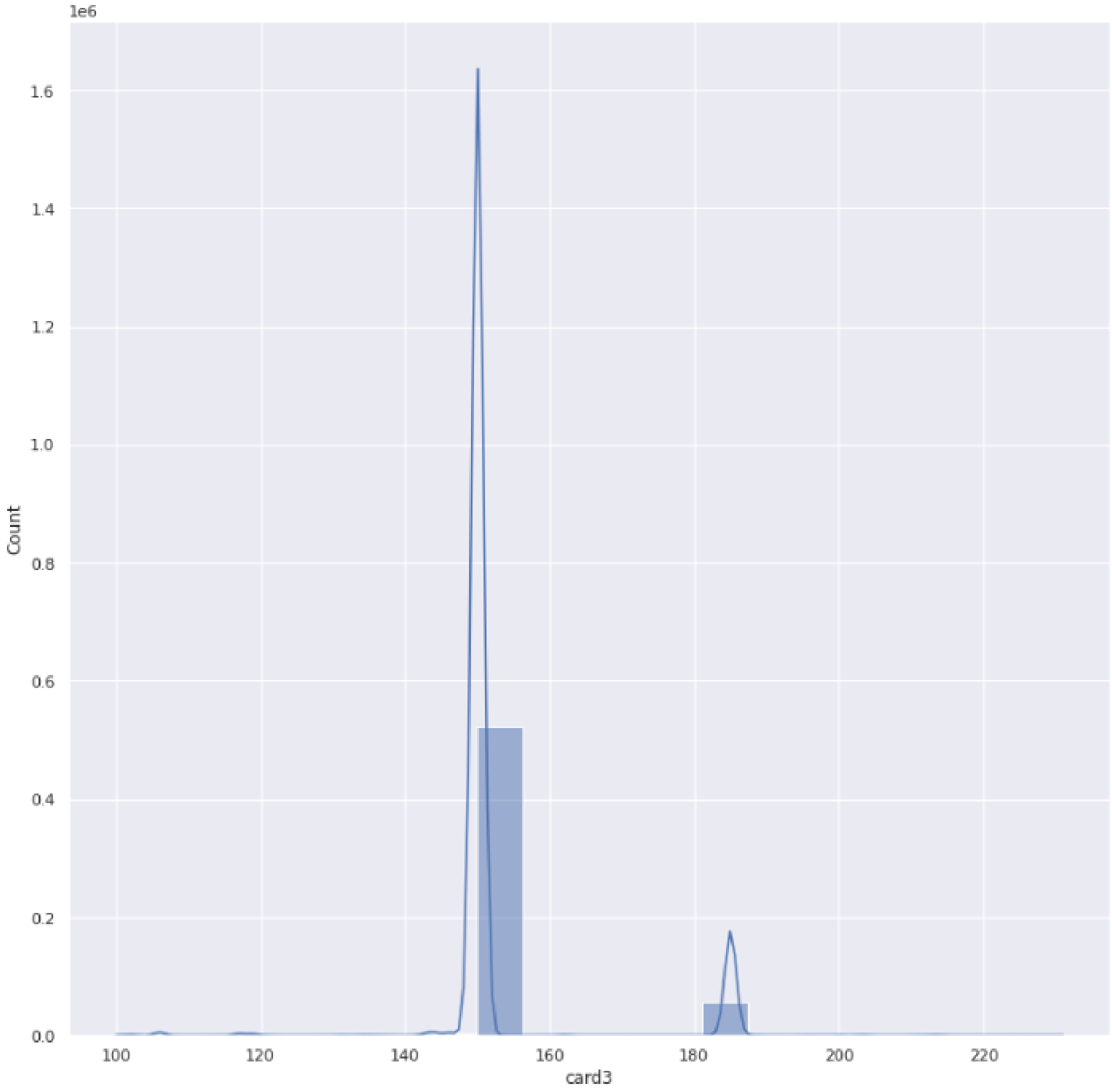
```
s = df.card5.value_counts(normalize=True)
missing = df['card5'].isnull()
#distributing null values according to original distribution
df.loc[missing,'card5'] = np.random.choice(s.index, size=len(df[missing]),p=s.values)
```


Count
25000
20000
15000
10000
5000
0
2500
5000
7500
10000
12500
15000
17500
card1

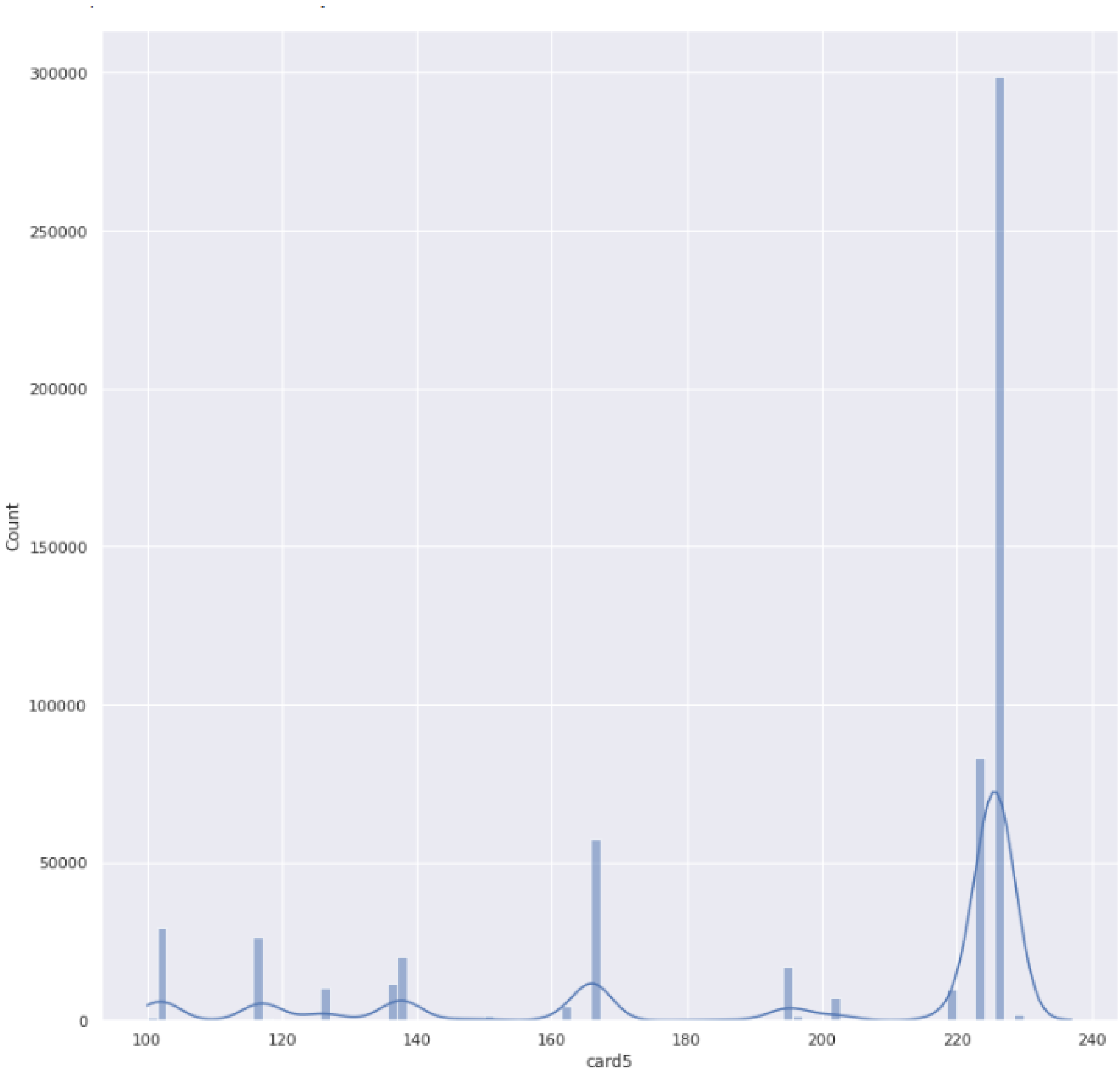
card1


50000
40000
30000
20000
10000
0
Count
100
200
300
400
500
600
card2

card2


1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Count
100
120
140
160
180
200
220
card3

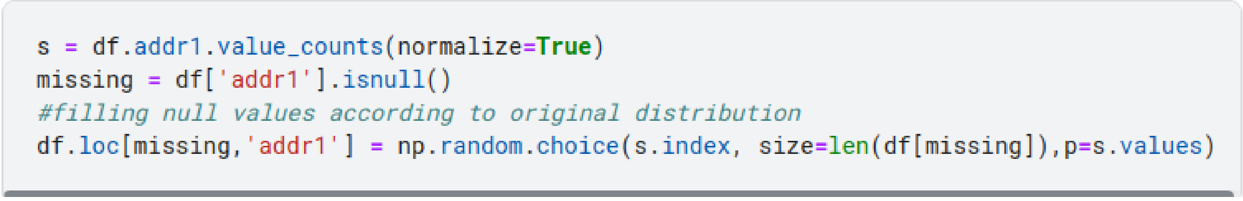
card3


300000
250000
200000
150000
100000
50000
0
Count
100
120
140
160
180
200
220
240
card5

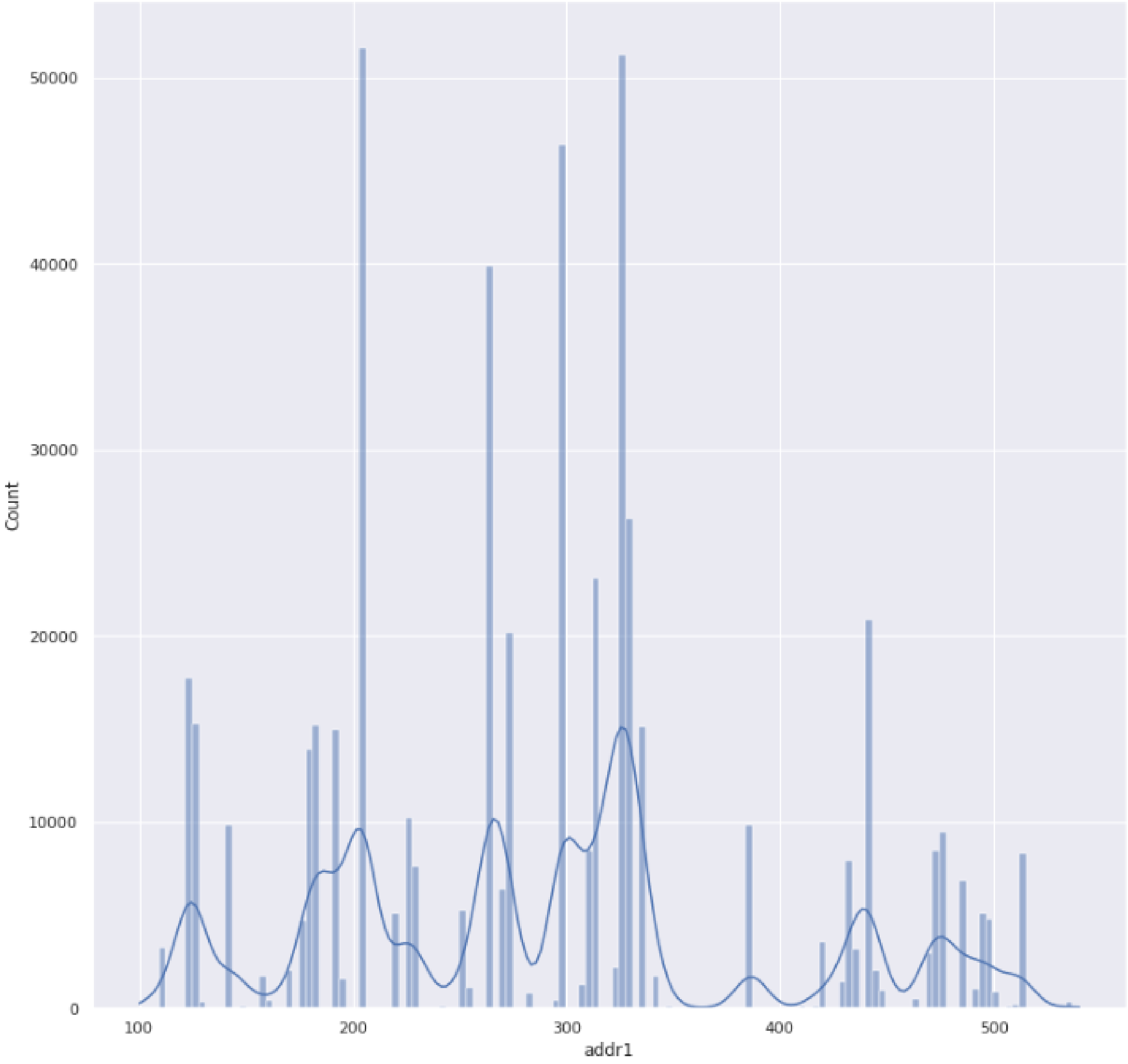
card5

3.2.2 addr1 and addr2

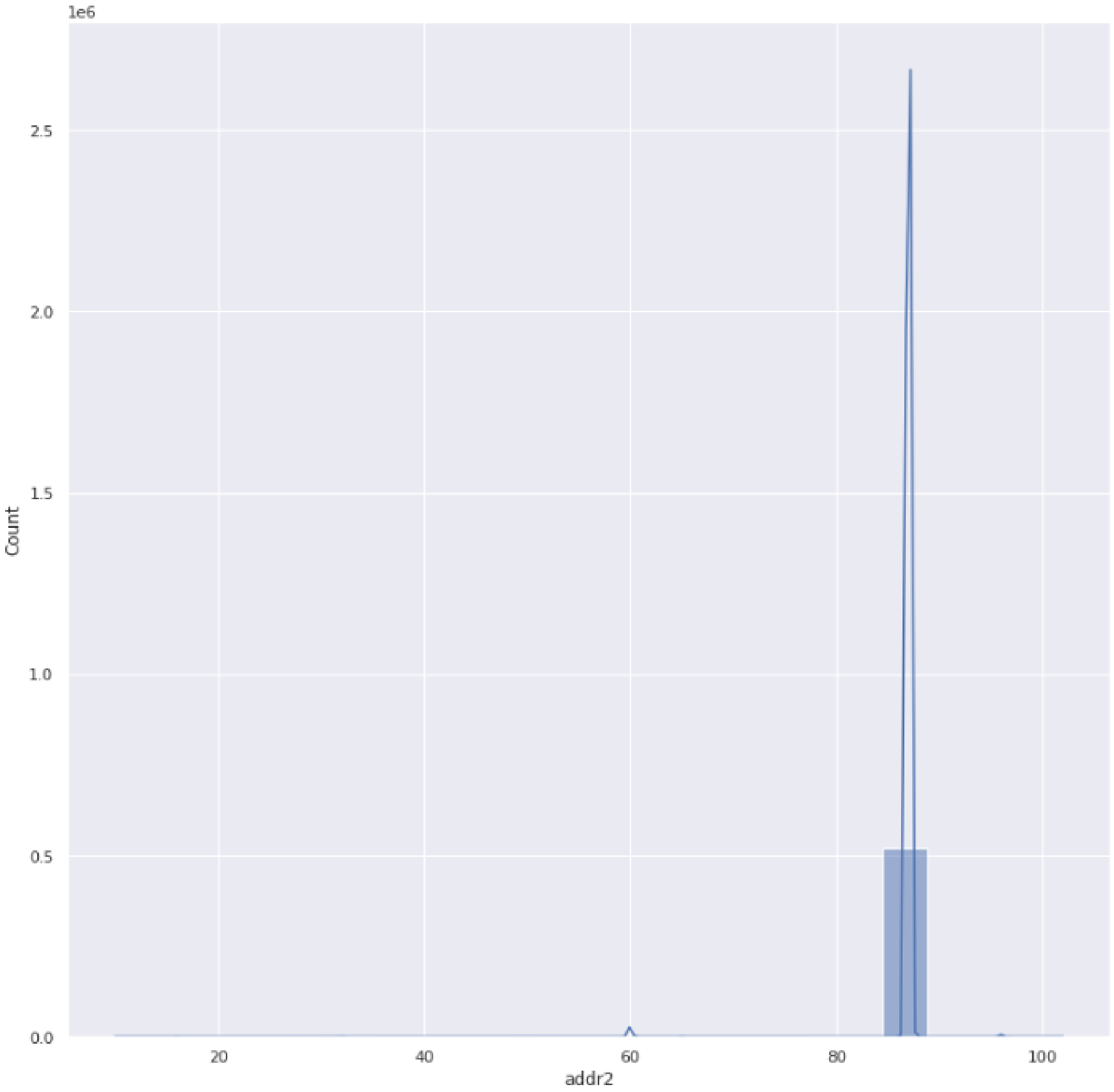
Both addr1 and addr2 columns have 65706 null values.we will fill the null value by the distribution of the numerical values as you can see in the below graphs. Here numerical value represents the categories.



```
s = df.addr1.value_counts(normalize=True)
missing = df['addr1'].isnull()
#filling null values according to original distribution
df.loc[missing,'addr1'] = np.random.choice(s.index, size=len(df[missing]),p=s.values)
```



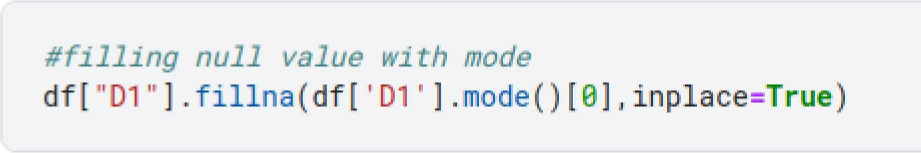
addr1



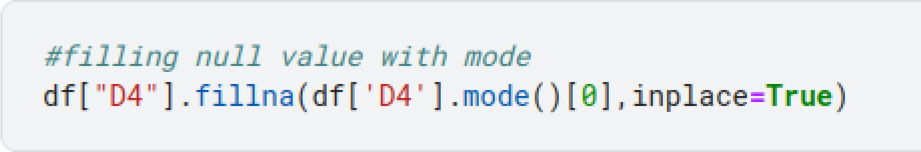
addr2

3.2.3 D1, D4, D10, D15

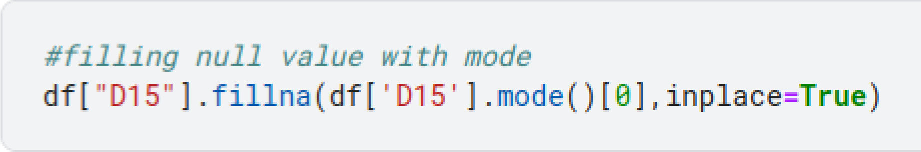
D1, D4, D10 and D15 columns have 1269,168922,76022,89133 null values respectively.we will fill the null value by the mode of each column.



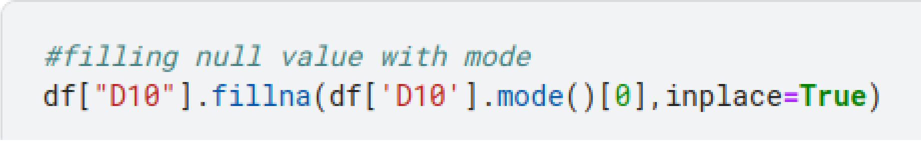
```
#filling null value with mode
df["D1"].fillna(df['D1'].mode()[0],inplace=True)
```



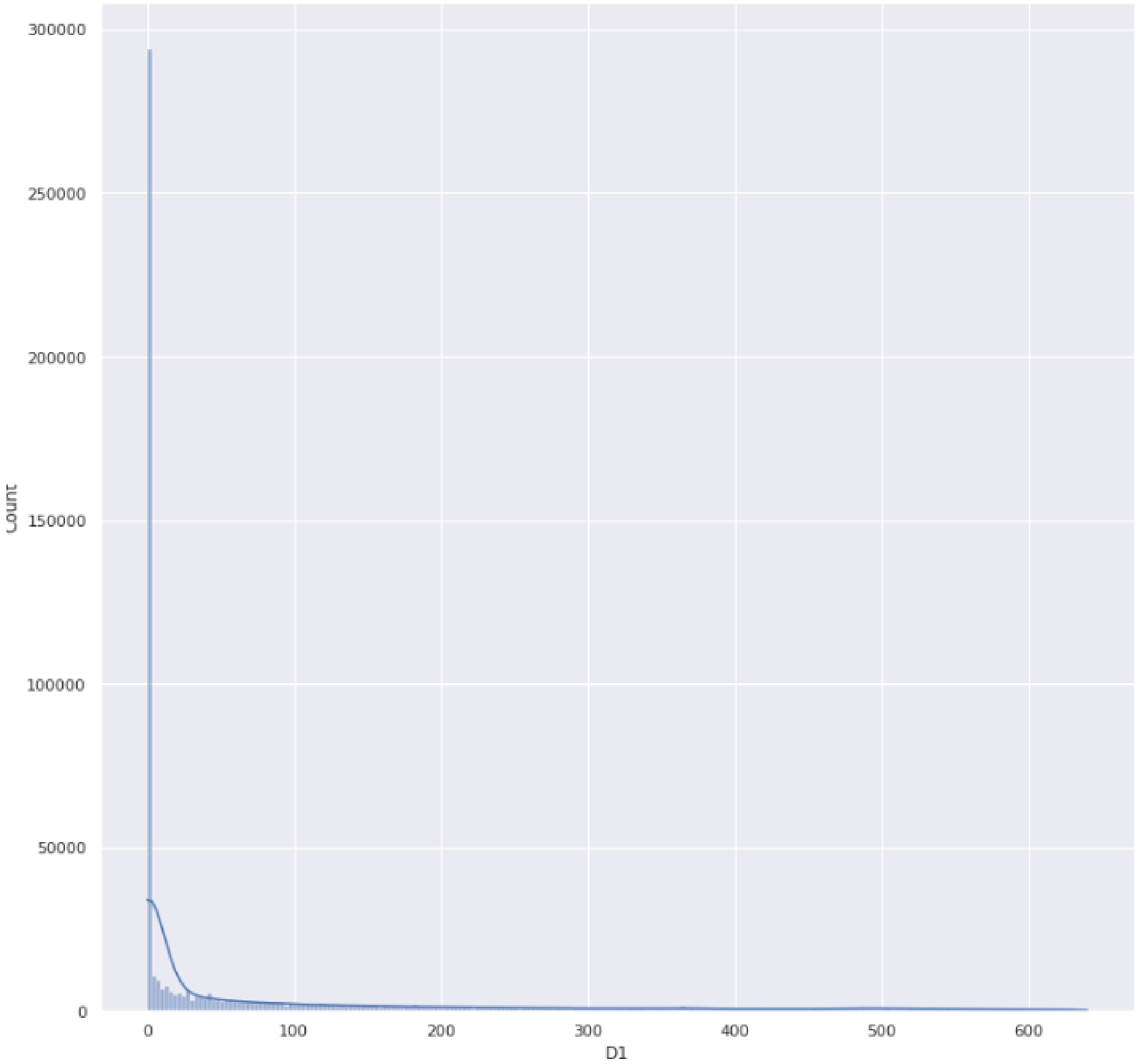
```
#filling null value with mode
df["D4"].fillna(df['D4'].mode()[0],inplace=True)
```



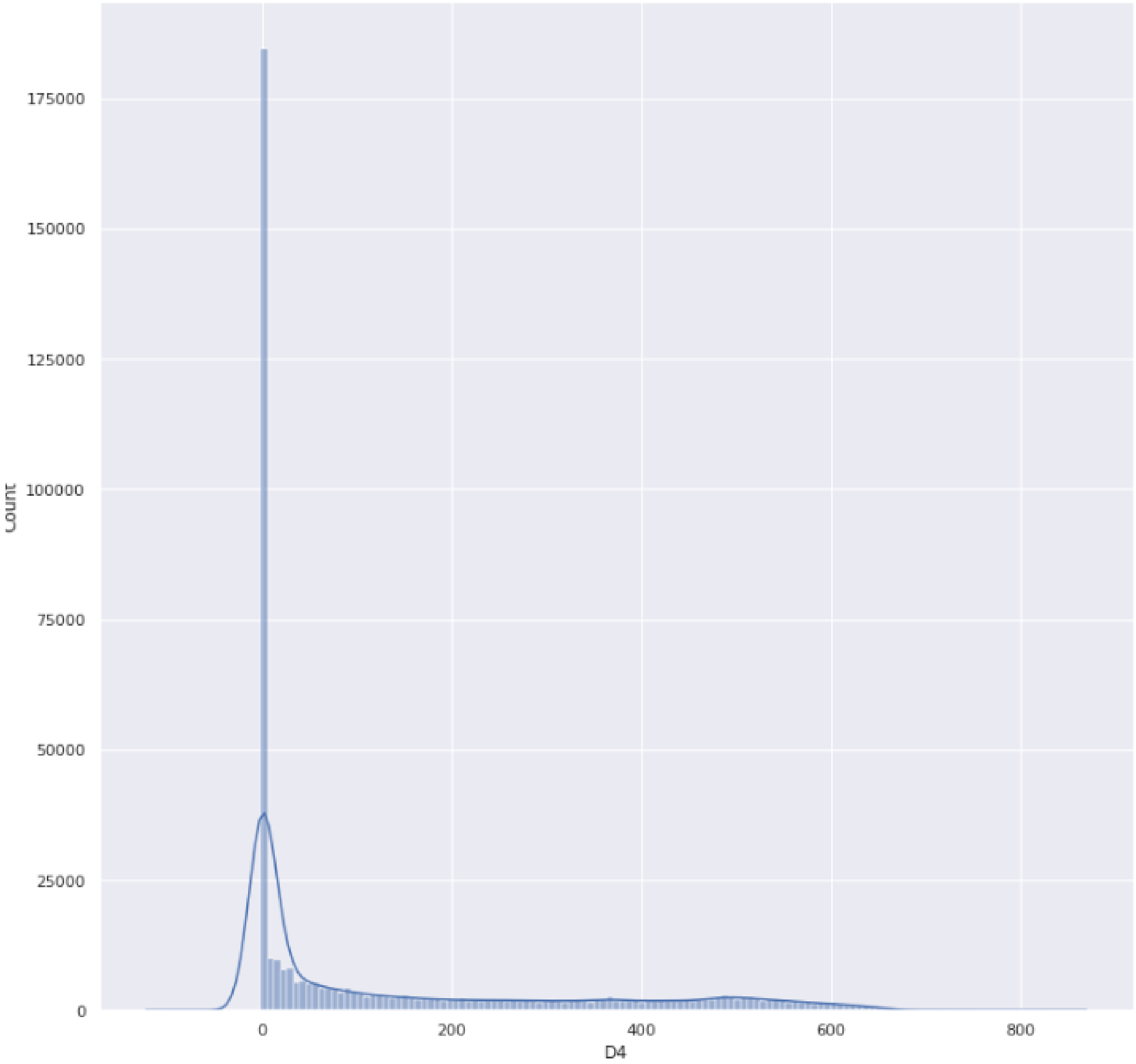
```
#filling null value with mode
df["D15"].fillna(df['D15'].mode()[0],inplace=True)
```



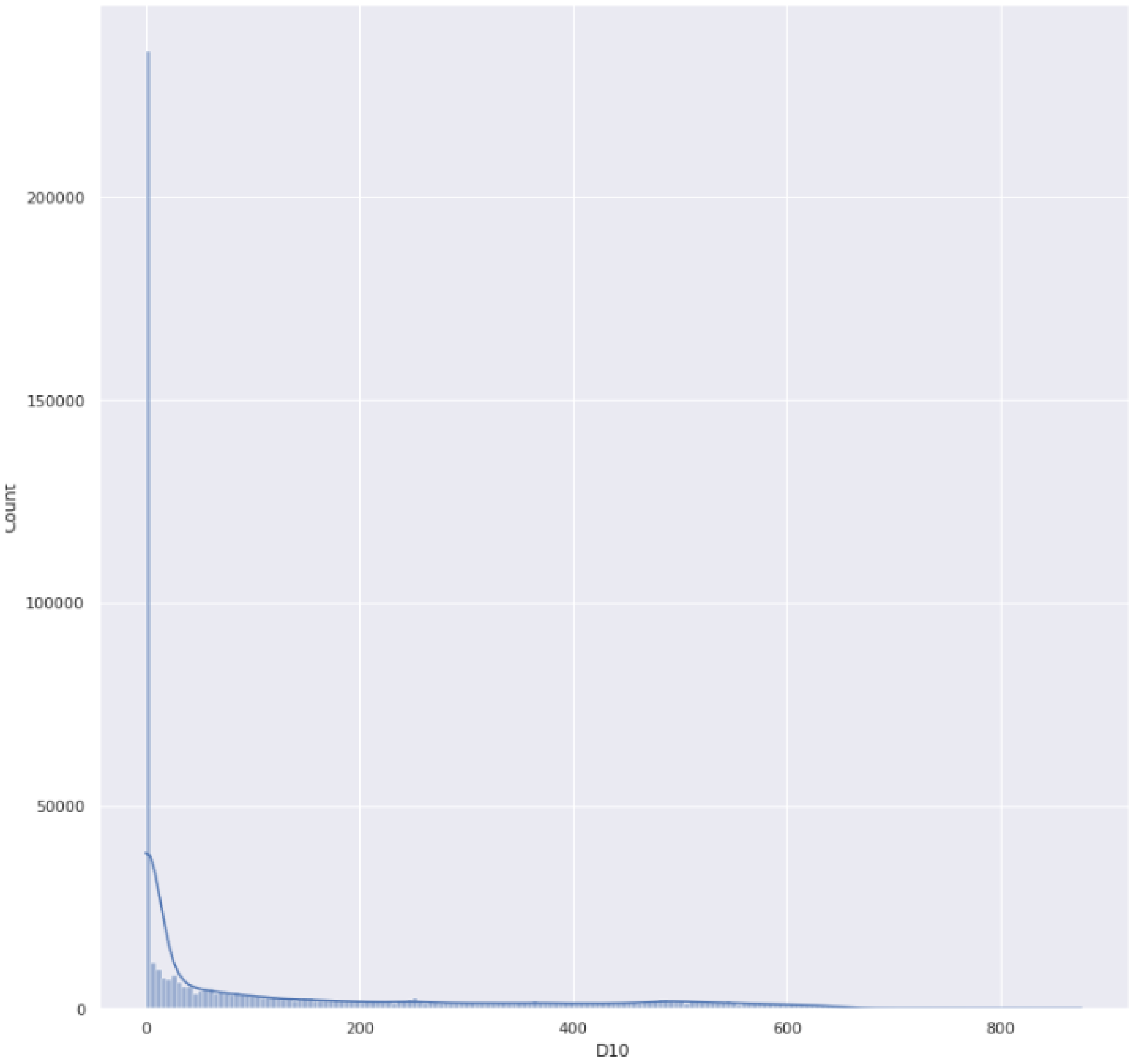
```
#filling null value with mode
df["D10"].fillna(df['D10'].mode()[0],inplace=True)
```


300000
250000
200000
150000
100000
50000
0
Count
0
100
200
300
400
500
600
D1

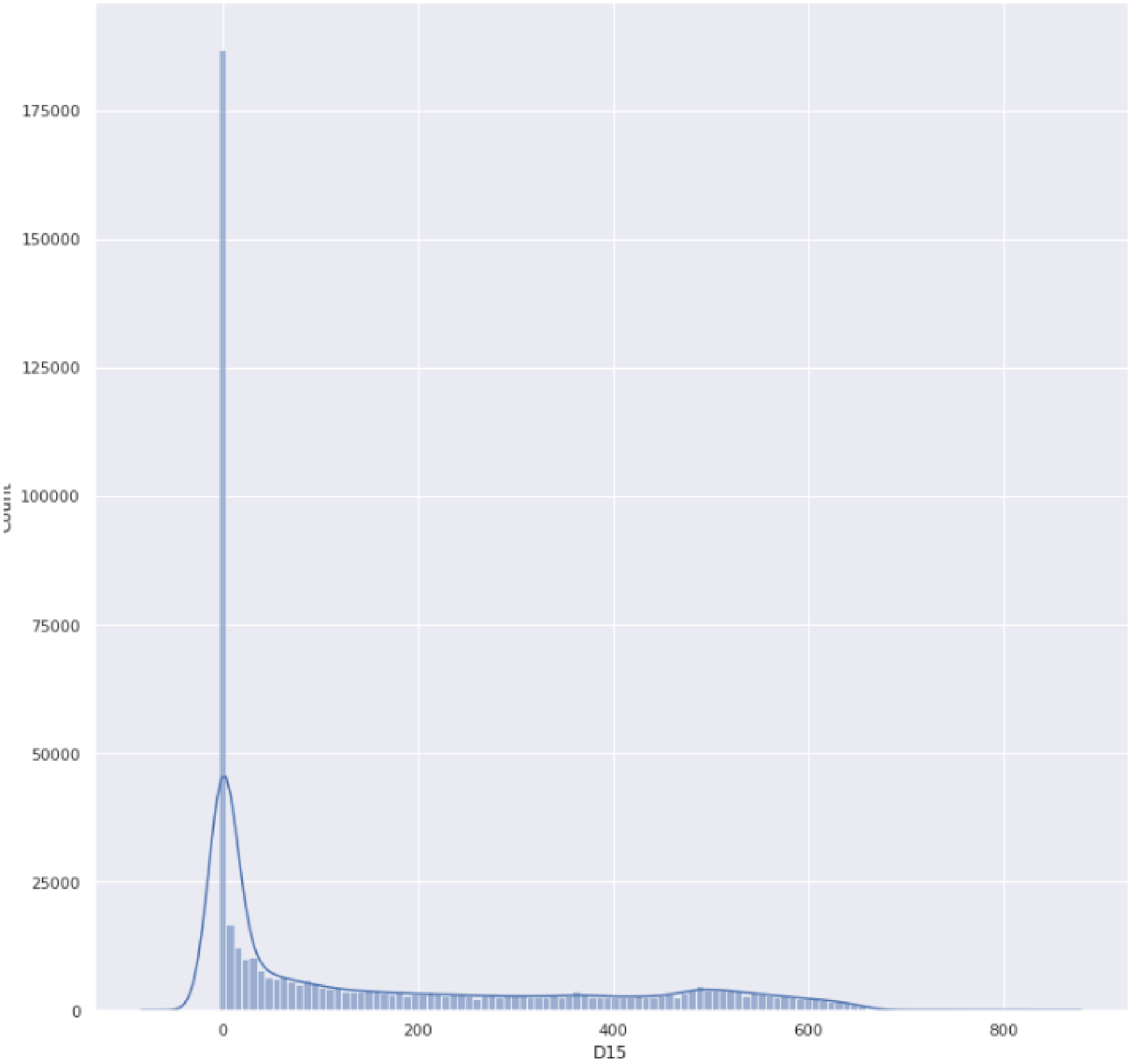
D1


Count
175000
150000
125000
100000
75000
50000
25000
0
0
200
400
600
800
D4

D4



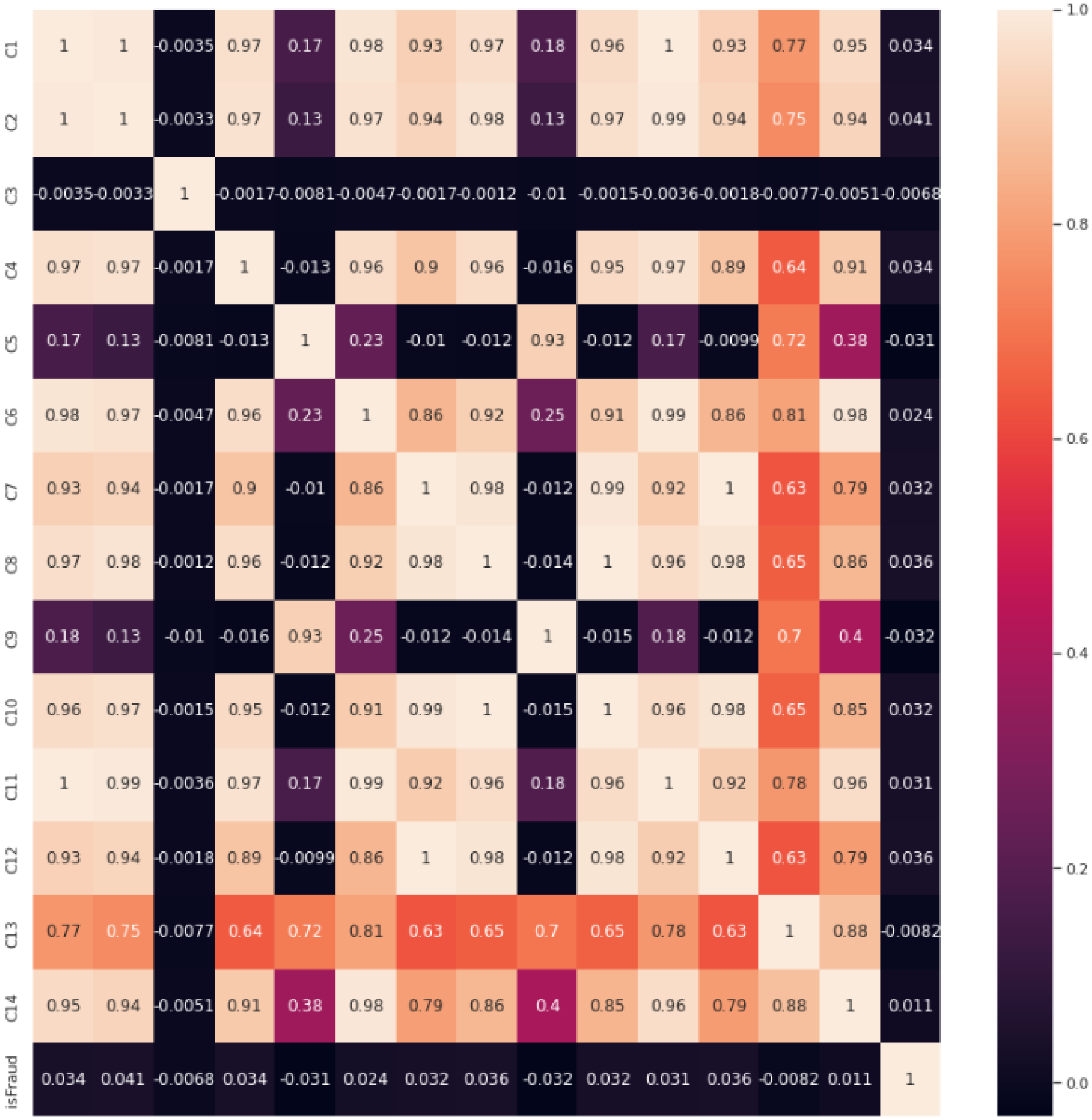
D10


175000
150000
125000
100000
75000
50000
25000
0
Count
0
200
400
600
800
D15

D15

3.2.4 C1-C14 columns

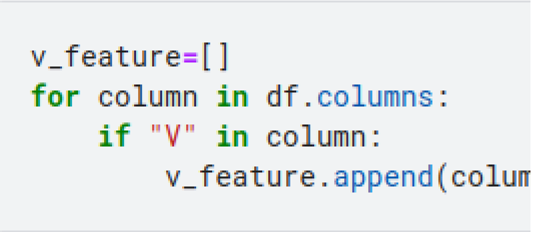
c1-c14 have no null values. Some of the columns have high correlations as we can see in the graph below. we will drop the columns which have a correlation greater than 90%. Ex. C2, C4, C6, C7, C8, C9, C10, C11, C12, C14



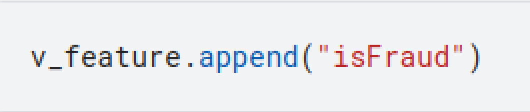
C1	1	1	-0.0035	0.97	0.17	0.98	0.93	0.97	0.18	0.96	1	0.93	0.77	0.95	0.034
C2	1	1	-0.0033	0.97	0.13	0.97	0.94	0.98	0.13	0.97	0.99	0.94	0.75	0.94	0.041
C3	-0.0035	-0.0033	1	-0.0017	-0.0081	-0.0047	-0.0017	-0.0012	-0.01	-0.0015	-0.0036	-0.0018	-0.0077	-0.0051	-0.0068
C4	0.97	0.97	-0.0017	1	-0.013	0.96	0.9	0.96	-0.016	0.95	0.97	0.89	0.64	0.91	0.034
C5	0.17	0.13	-0.0081	-0.013	1	0.23	-0.01	-0.012	0.93	-0.012	0.17	-0.0099	0.72	0.38	-0.031
C6	0.98	0.97	-0.0047	0.96	0.23	1	0.86	0.92	0.25	0.91	0.99	0.86	0.81	0.98	0.024
C7	0.93	0.94	-0.0017	0.9	-0.01	0.86	1	0.98	-0.012	0.99	0.92	1	0.63	0.79	0.032
C8	0.97	0.98	-0.0012	0.96	-0.012	0.92	0.98	1	-0.014	1	0.96	0.98	0.65	0.86	0.036
C9	0.18	0.13	-0.01	-0.016	0.93	0.25	-0.012	-0.014	1	-0.015	0.18	-0.012	0.7	0.4	-0.032
C10	0.96	0.97	-0.0015	0.95	-0.012	0.91	0.99	1	-0.015	1	0.96	0.98	0.65	0.85	0.032
C11	1	0.99	-0.0036	0.97	0.17	0.99	0.92	0.96	0.18	0.96	1	0.92	0.78	0.96	0.031
C12	0.93	0.94	-0.0018	0.89	-0.0099	0.86	1	0.98	-0.012	0.98	0.92	1	0.63	0.79	0.036
C13	0.77	0.75	-0.0077	0.64	0.72	0.81	0.63	0.65	0.7	0.65	0.78	0.63	1	0.88	-0.0082
C14	0.95	0.94	-0.0051	0.91	0.38	0.98	0.79	0.86	0.4	0.85	0.96	0.79	0.88	1	0.011
isFraud	0.034	0.041	-0.0068	0.034	-0.031	0.024	0.032	0.036	-0.032	0.032	0.031	0.036	-0.0082	0.011	1

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

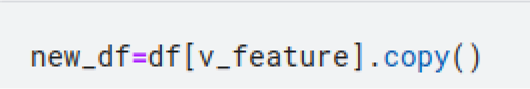
V12-V34 has 76073 null values.
V35-V52 has 168969 null values.
V53-V74 has a 77096 null value.
V75-V94 has an 89164 null value.
V95-V137 has 314 null values.
V279, V280, V284-V287, V290-V295, V297-V299, V302-V312, V316-V321 have 12 null values.
V281-V283, V288-V289, V296, V300, V301, and V313-V315 has 1269 null values.



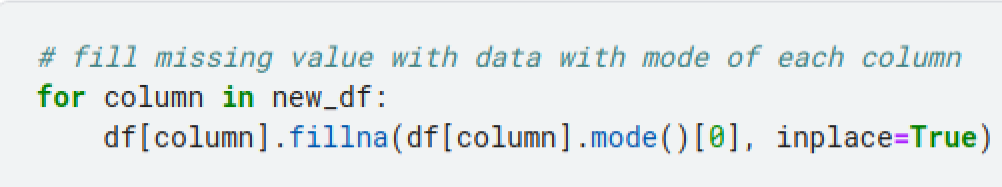
```
v_feature=[]
for column in df.columns:
    if "V" in column:
        v_feature.append(colum
```



```
v_feature.append("isFraud")
```

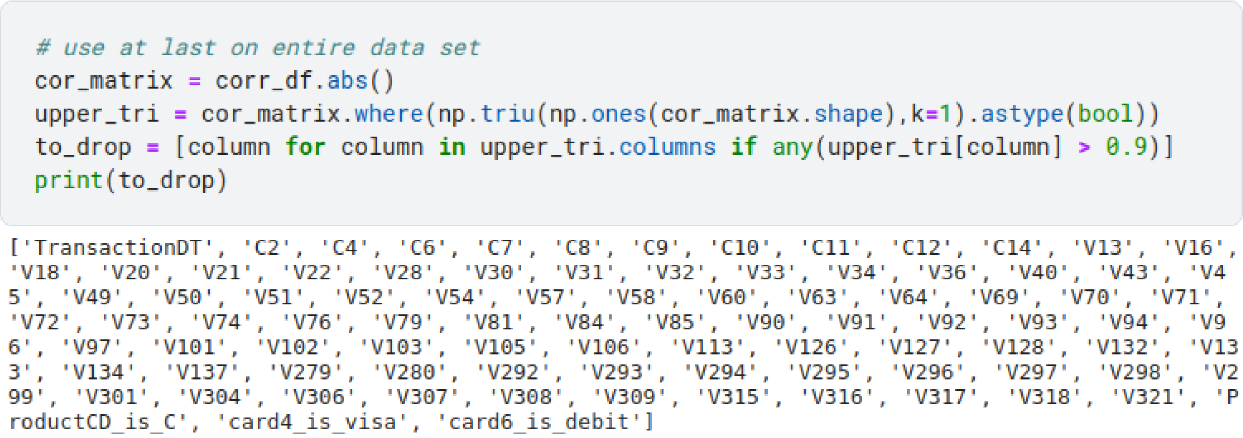


```
new_df=df[v_feature].copy()
```



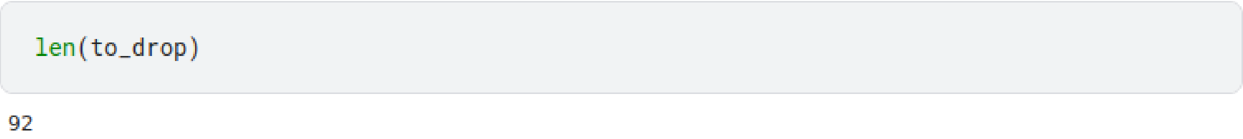
```
# fill missing value with data with mode of each column
for column in new_df:
    df[column].fillna(df[column].mode()[0], inplace=True)
```

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'



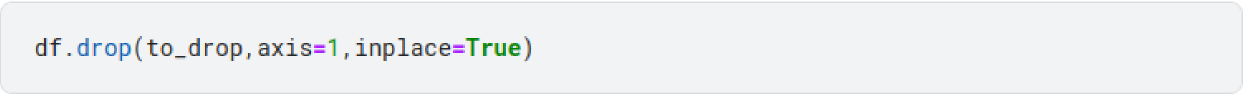
```
# use at last on entire data set
cor_matrix = corr_df.abs()
upper_tri = cor_matrix.where(np.triu(np.ones(cor_matrix.shape),k=1).astype(bool))
to_drop = [column for column in upper_tri.columns if any(upper_tri[column] > 0.9)]
print(to_drop)
```

```
['TransactionDT', 'C2', 'C4', 'C6', 'C7', 'C8', 'C9', 'C10', 'C11', 'C12', 'C14', 'V13', 'V16',
'V18', 'V20', 'V21', 'V22', 'V28', 'V30', 'V31', 'V32', 'V33', 'V34', 'V36', 'V40', 'V43', 'V4
5', 'V49', 'V50', 'V51', 'V52', 'V54', 'V57', 'V58', 'V60', 'V63', 'V64', 'V69', 'V70', 'V71',
'V72', 'V73', 'V74', 'V76', 'V79', 'V81', 'V84', 'V85', 'V90', 'V91', 'V92', 'V93', 'V94', 'V9
6', 'V97', 'V101', 'V102', 'V103', 'V105', 'V106', 'V113', 'V126', 'V127', 'V128', 'V132', 'V13
3', 'V134', 'V137', 'V279', 'V280', 'V292', 'V293', 'V294', 'V295', 'V296', 'V297', 'V298', 'V2
99', 'V301', 'V304', 'V306', 'V307', 'V308', 'V309', 'V315', 'V316', 'V317', 'V318', 'V321', 'P
roductCD_is_C', 'card4_is_visa', 'card6_is_debit']
```



```
len(to_drop)
```

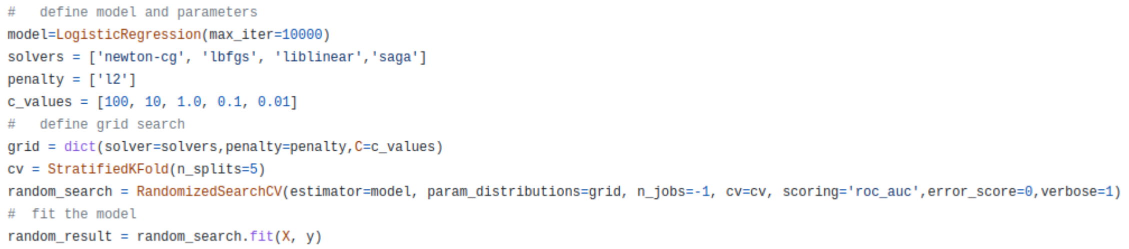
```
92
```



```
df.drop(to_drop,axis=1,inplace=True)
```

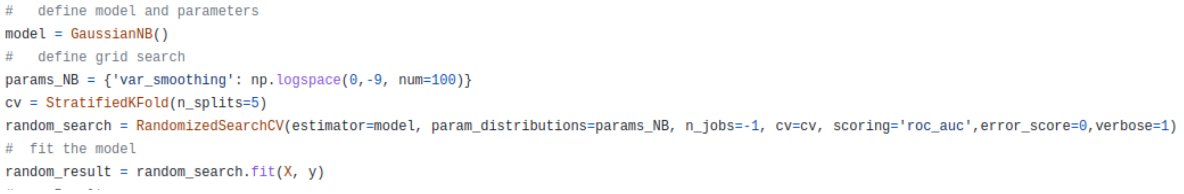
4. Training Models Evaluation

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.



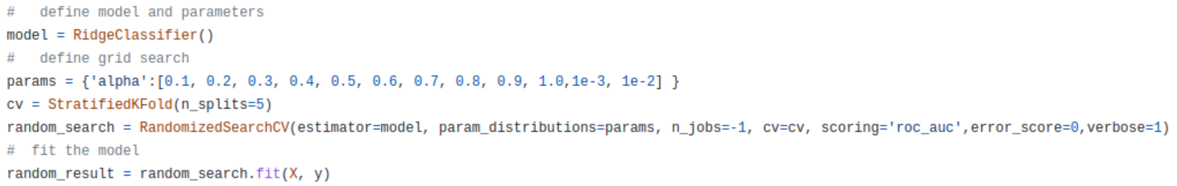
```
#   define model and parameters
model=LogisticRegression(max_iter=10000)
solvers = ['newton-cg', 'lbfgs', 'liblinear','saga']
penalty = ['l2']
c_values = [100, 10, 1.0, 0.1, 0.01]
#   define grid search
grid = dict(solver=solvers,penalty=penalty,C=c_values)
cv = StratifiedKFold(n_splits=5)
random_search = RandomizedSearchCV(estimator=model, param_distributions=grid, n_jobs=-1, cv=cv, scoring='roc_auc',error_score=0,verbose=1)
#  fit the model
random_result = random_search.fit(X, y)
```

Logistic



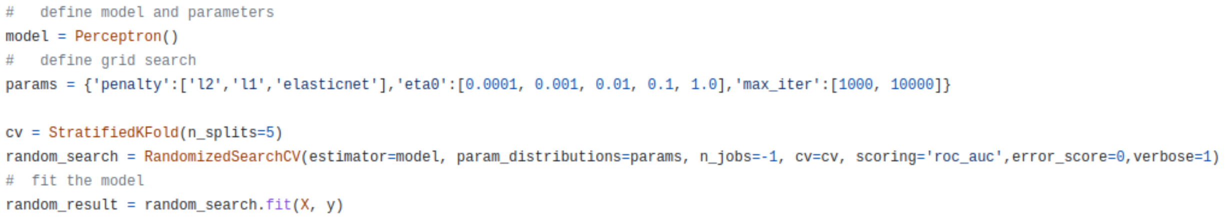
```
#   define model and parameters
model = GaussianNB()
#   define grid search
params_NB = {'var_smoothing': np.logspace(0,-9, num=100)}
cv = StratifiedKFold(n_splits=5)
random_search = RandomizedSearchCV(estimator=model, param_distributions=params_NB, n_jobs=-1, cv=cv, scoring='roc_auc',error_score=0,verbose=1)
#  fit the model
random_result = random_search.fit(X, y)
```

Naive Bayes



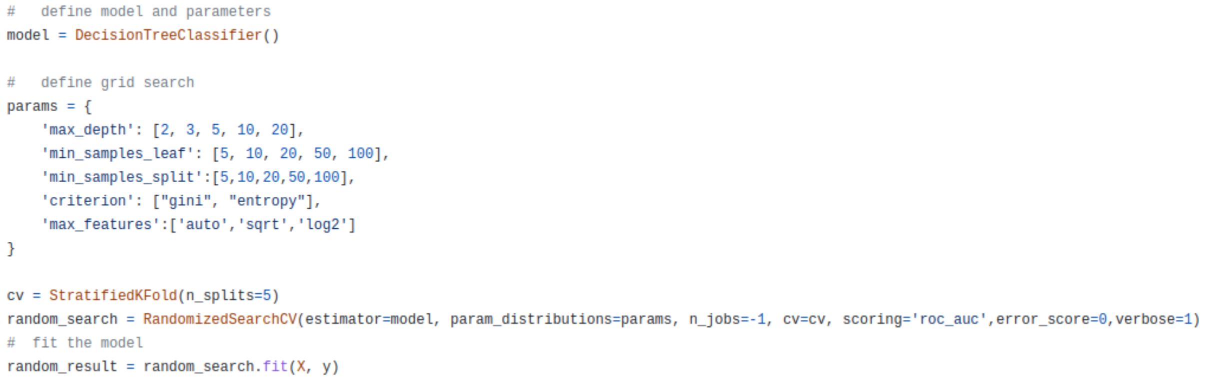
```
#   define model and parameters
model = RidgeClassifier()
#   define grid search
params = {'alpha':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0,1e-3, 1e-2] }
cv = StratifiedKFold(n_splits=5)
random_search = RandomizedSearchCV(estimator=model, param_distributions=params, n_jobs=-1, cv=cv, scoring='roc_auc',error_score=0,verbose=1)
#  fit the model
random_result = random_search.fit(X, y)
```

Ridge Classifier



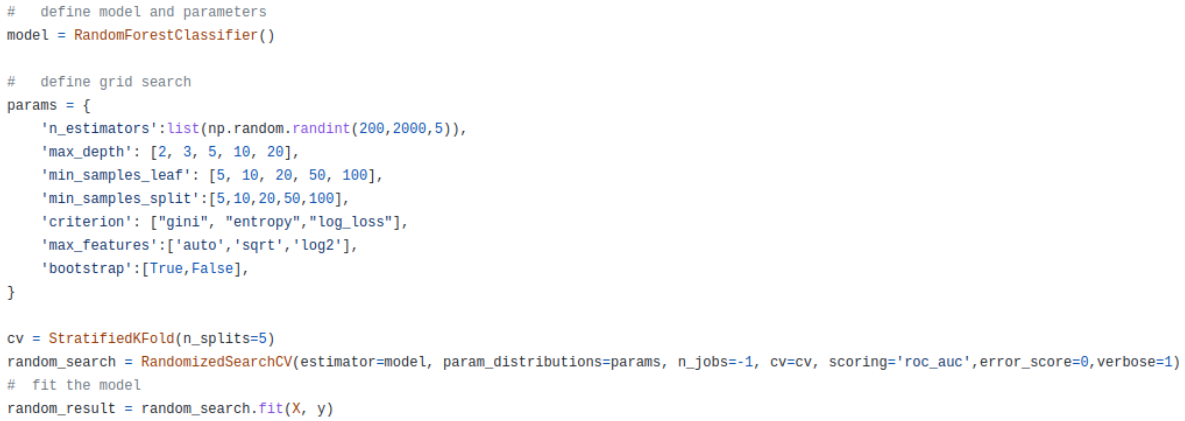
```
#   define model and parameters
model = Perceptron()
#   define grid search
params = {'penalty':['l2','l1','elasticnet'],'eta0':[0.0001, 0.001, 0.01, 0.1, 1.0],'max_iter':[1000, 10000]}

cv = StratifiedKFold(n_splits=5)
random_search = RandomizedSearchCV(estimator=model, param_distributions=params, n_jobs=-1, cv=cv, scoring='roc_auc',error_score=0,verbose=1)
#  fit the model
random_result = random_search.fit(X, y)
```

Perceptron



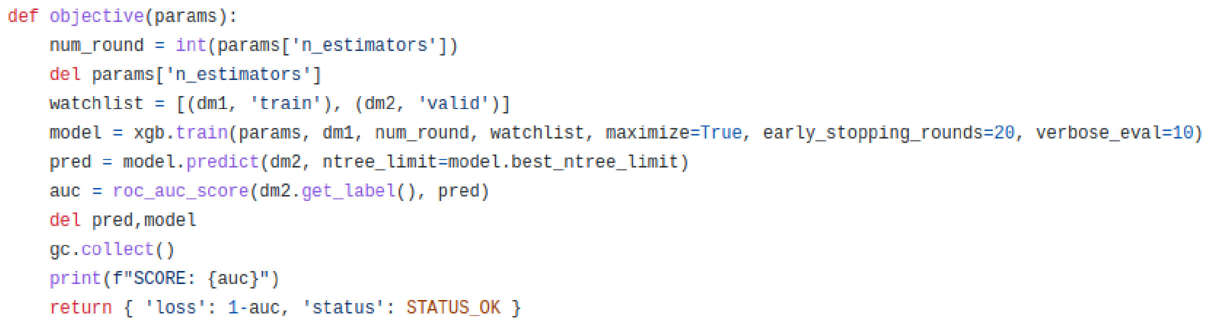
```
#   define model and parameters
model = DecisionTreeClassifier()

#   define grid search
params = {
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'min_samples_split':[5,10,20,50,100],
    'criterion': ["gini", "entropy"],
    'max_features':['auto','sqrt','log2']
}

cv = StratifiedKFold(n_splits=5)
random_search = RandomizedSearchCV(estimator=model, param_distributions=params, n_jobs=-1, cv=cv, scoring='roc_auc',error_score=0,verbose=1)
#  fit the model
random_result = random_search.fit(X, y)
```

Decision Tree

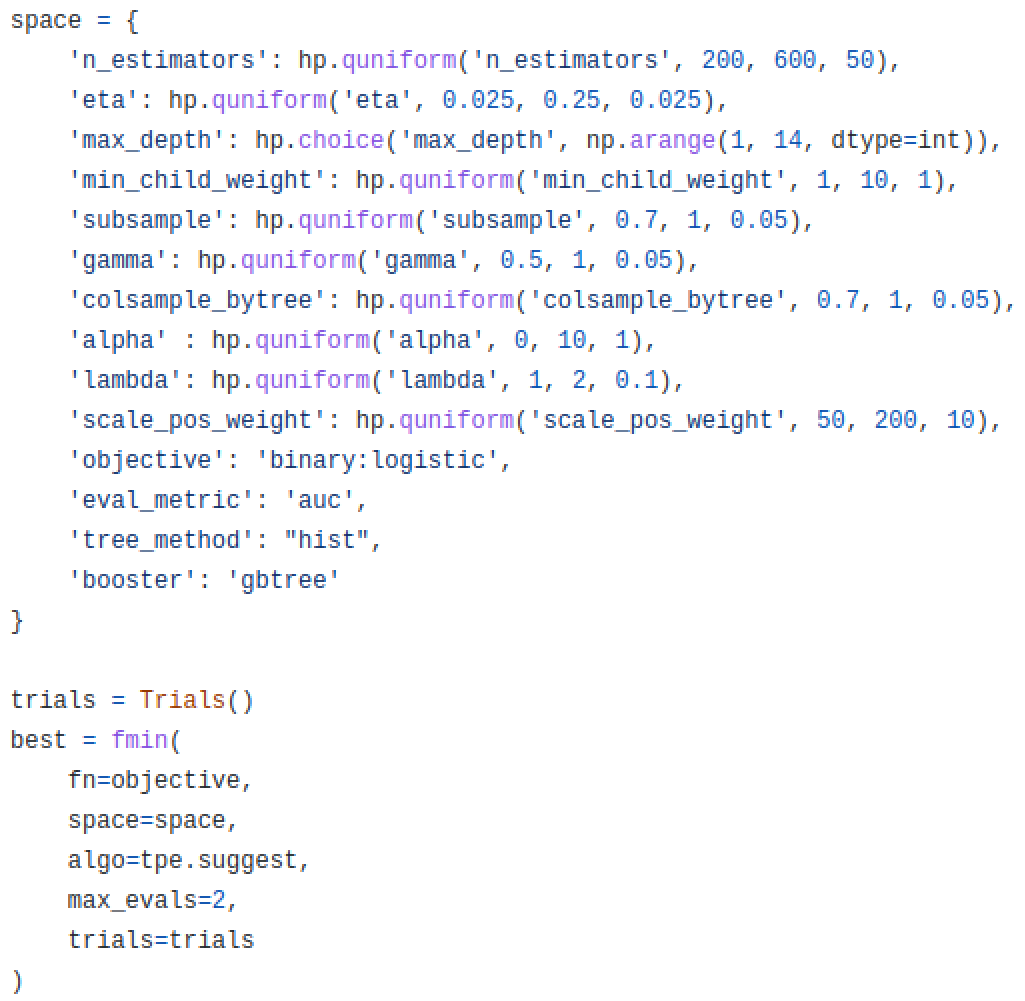


```
#   define model and parameters
model = RandomForestClassifier()

#   define grid search
params = {
    'n_estimators':list(np.random.randint(200,2000,5)),
    'max_depth': [2, 3, 5, 10, 20],
    'min_samples_leaf': [5, 10, 20, 50, 100],
    'min_samples_split':[5,10,20,50,100],
    'criterion': ["gini", "entropy","log_loss"],
    'max_features':['auto','sqrt','log2'],
    'bootstrap':[True,False],
}

cv = StratifiedKFold(n_splits=5)
random_search = RandomizedSearchCV(estimator=model, param_distributions=params, n_jobs=-1, cv=cv, scoring='roc_auc',error_score=0,verbose=1)
#  fit the model
random_result = random_search.fit(X, y)
```

Random Forest



```
def objective(params):
    num_round = int(params['n_estimators'])
    del params['n_estimators']
    watchlist = [(dm1, 'train'), (dm2, 'valid')]
    model = xgb.train(params, dm1, num_round, watchlist, maximize=True, early_stopping_rounds=20, verbose_eval=10)
    pred = model.predict(dm2, ntree_limit=model.best_ntree_limit)
    auc = roc_auc_score(dm2.get_label(), pred)
    del pred,model
    gc.collect()
    print(f"SCORE: {auc}")
    return { 'loss': 1-auc, 'status': STATUS_OK }
```



```
space = {
    'n_estimators': hp.quniform('n_estimators', 200, 600, 50),
    'eta': hp.quniform('eta', 0.025, 0.25, 0.025),
    'max_depth': hp.choice('max_depth', np.arange(1, 14, dtype=int)),
    'min_child_weight': hp.quniform('min_child_weight', 1, 10, 1),
    'subsample': hp.quniform('subsample', 0.7, 1, 0.05),
    'gamma': hp.quniform('gamma', 0.5, 1, 0.05),
    'colsample_bytree': hp.quniform('colsample_bytree', 0.7, 1, 0.05),
    'alpha' : hp.quniform('alpha', 0, 10, 1),
    'lambda': hp.quniform('lambda', 1, 2, 0.1),
    'scale_pos_weight': hp.quniform('scale_pos_weight', 50, 200, 10),
    'objective': 'binary:logistic',
    'eval_metric': 'auc',
    'tree_method': "hist",
    'booster': 'gbtree'
}

trials = Trials()
best = fmin(
    fn=objective,
    space=space,
    algo=tpe.suggest,
    max_evals=2,
    trials=trials
)
```

XGboost

Model	Hyper Parameters	Value	Model Score
Logistic regression classifier	max_iter	1000	0.57204
	solver	libliner	
	penalty	l2	
	c	0.01	
Naive Bayes	var_smoothing	0.0533669923120631	0.70151
Ridge Classifier	alpha	1	0.53441
Perceptron	penalty	l1	0.52756
	max_iter	1000	
	eta0	0.0001	
Decision Tree	min_sample_split	5	0.62807
	min_samples_leaf	50	
	max_feature	sqrt	
	max_depth	20	
	criterion	gini	
Random Forest	n_estimators	1635	0.56897
	min_samples_split	5	
	min_samples_leaf	100	
	max_feature	sqrt	
	max_depth	10	
	criterion	entropy	
	bootstrap	TRUE	
Xgboost	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/