It’s a Fraud

Team Prem

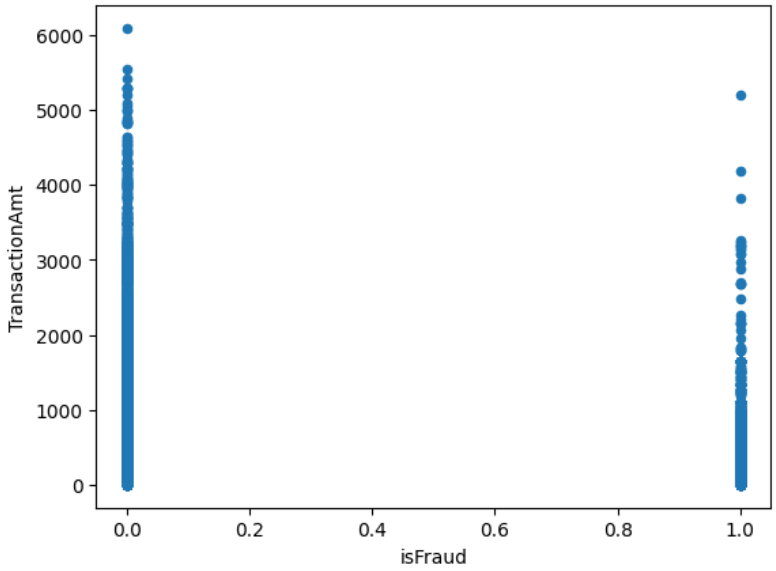
About the dataset

About the columns (features) of the data:

- `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 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 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),digital signature (UA/browser/os/version, etc) associated with transactions are also present

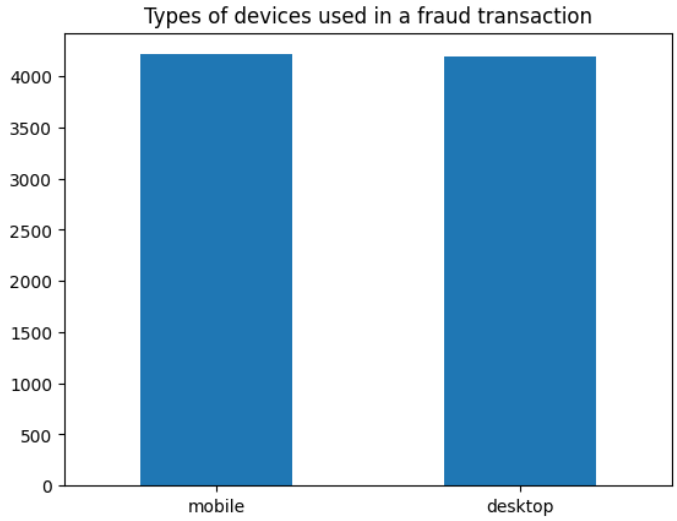
EDA

Transaction Amount



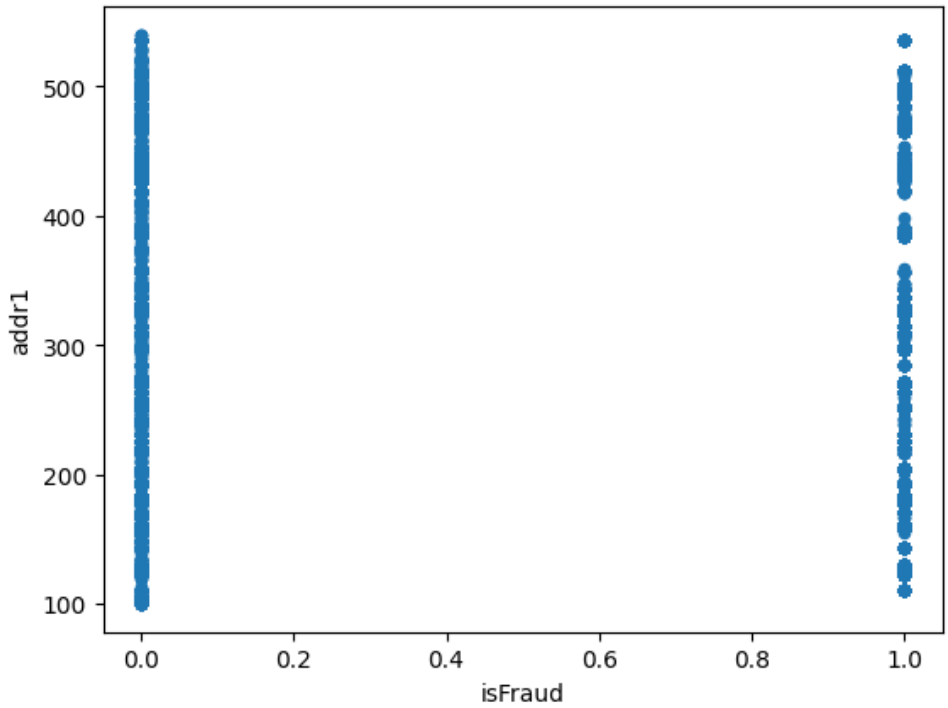
The training data shows that maximum frauds happen up-to transaction amount of 4000. There are less chances of fraud at higher amount but still it can happen at any amount, yet we can infer that the transactions of more than 4000 could be genuine but not with 100% surety !!

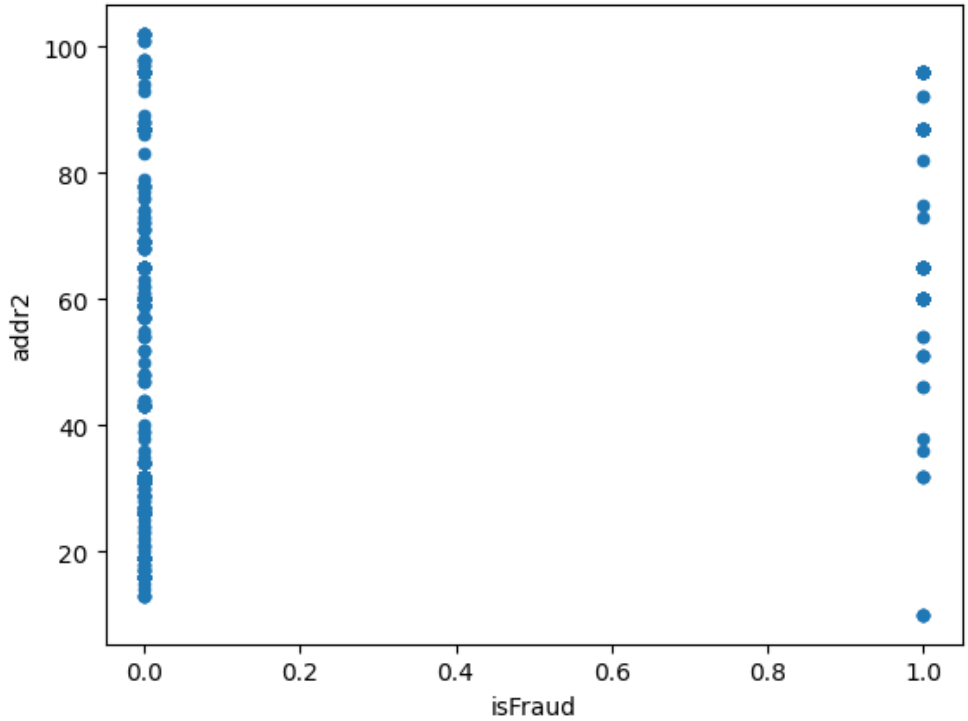
Device Type



We can clearly see that the device type does not say anything about the transaction. One can fraud from desktop as well as mobile. So this feature is not of much use.

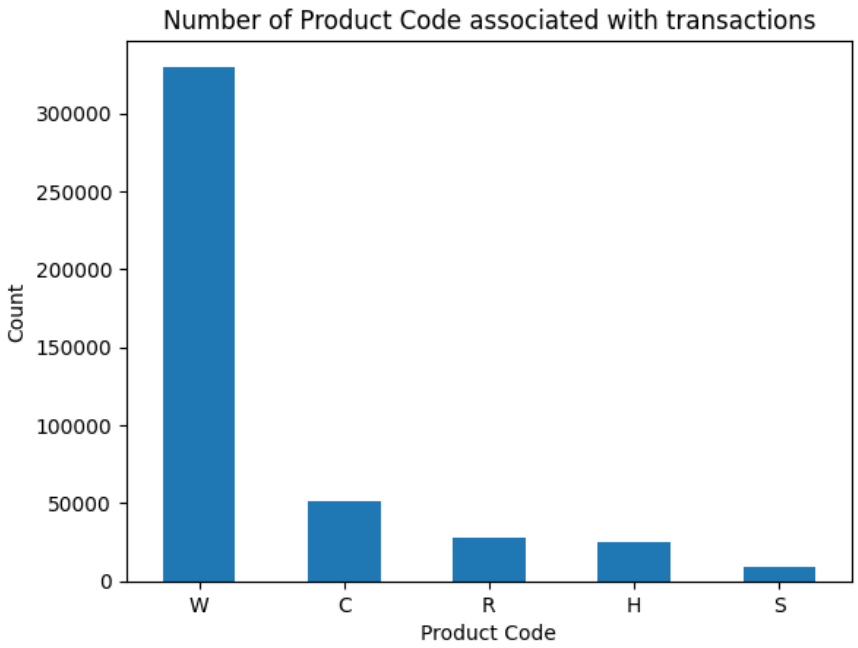
Billing Address

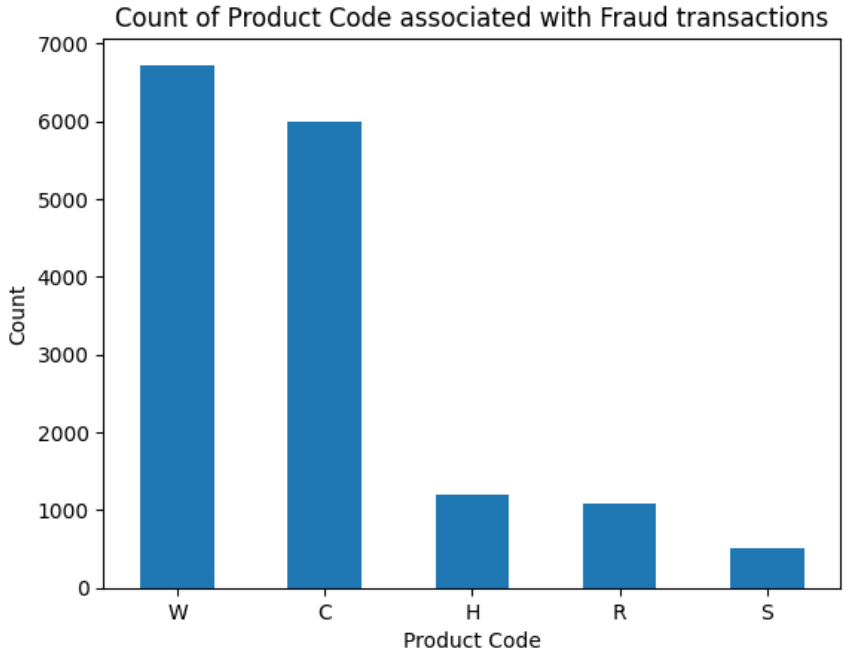




We cannot say much about the transaction by just looking at the billing region (addr1) or billing country (addr2) as frauds are scattered all over the regions.

Product CD



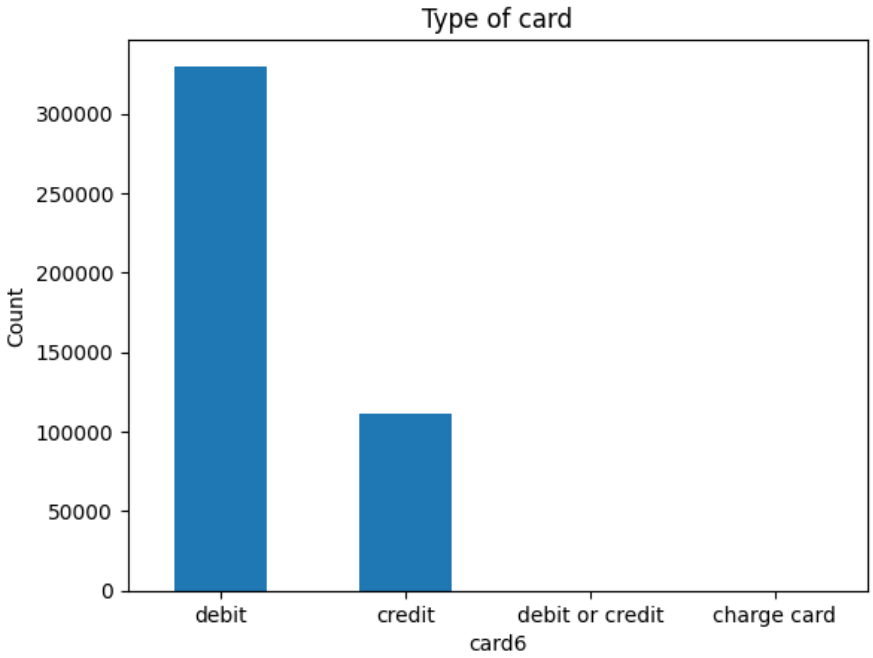


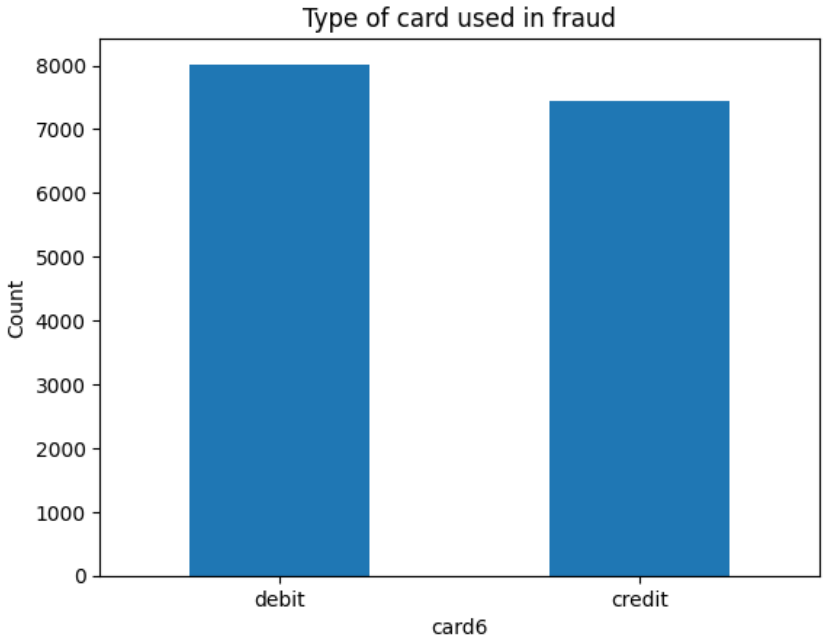
Although we can see that products with ProductCD = 'W' has the most number of frauds, this is clear that out of number of products with code 'C' it has maximum percentage of fraud transactions.

There are about 50,000 products with code 'C' and about 6000 of them are fraud which makes it around 12%.

And the other codes roughly make it around 5-6%.

Card

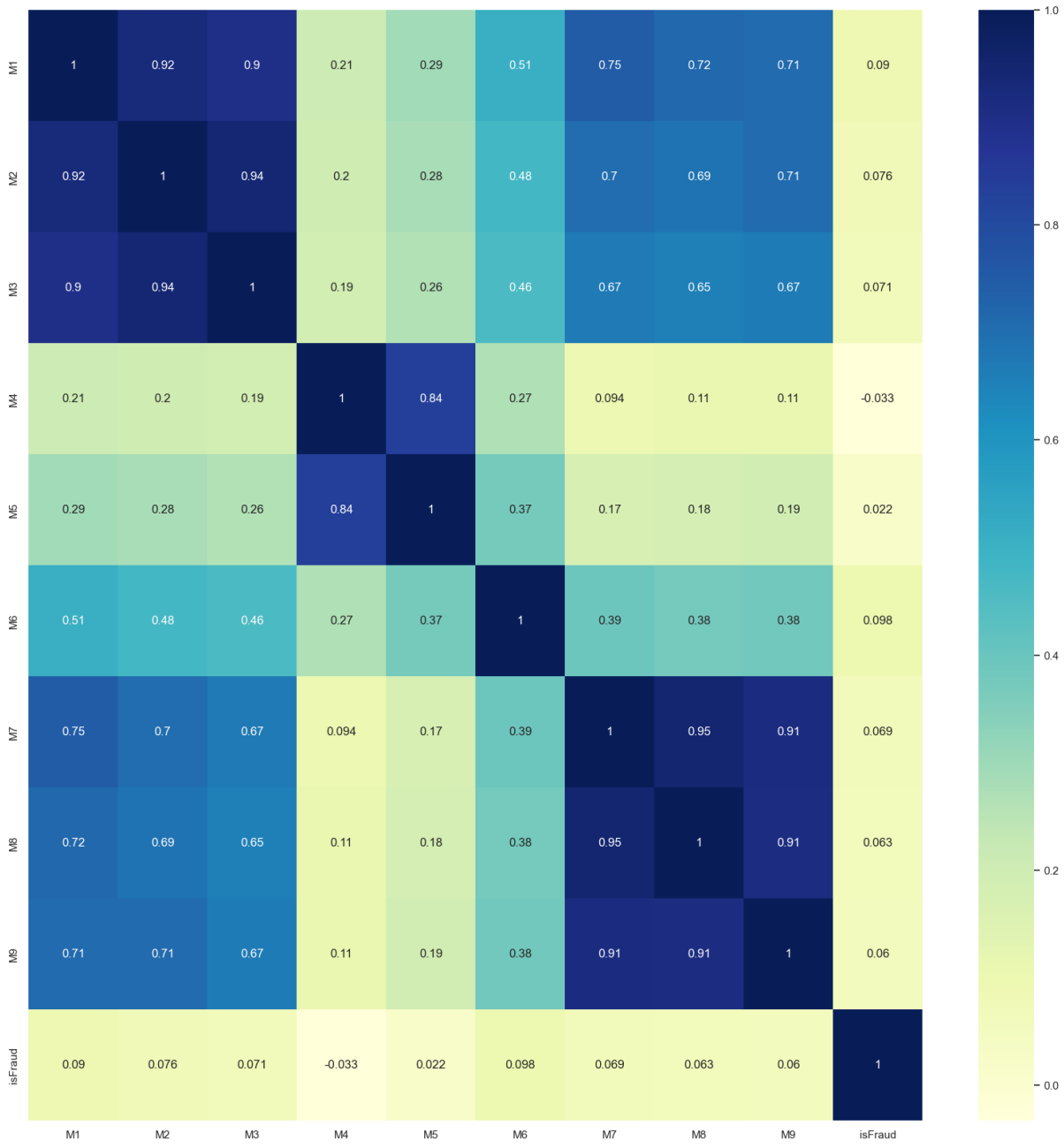




If you look at the overall data then credit card is used very less in compare to debit card, but when it comes to fraud credit card are used almost same as debit card.

So, we can conclude that there are bigger chance of transaction being fraud when credit card is used (7%) than debit card (2.5%).

M Columns



Here M2, M3, M8 and M9 are highly correlated.

So, we can keep only one of them and remove the others.

M2 is most correlated with isFraud among the four.

D Columns

Here D2, D7, D12 and D6 are highly correlated.

So, we can keep only one of them and remove the others.

As D6 has the most number of unique values we will keep D6.

Pre-processing

1. Device Info

This feature consisted of the Operating software used by the device (like iOS, etc.) or the brand name with different notation (like Motorola was represented by ‘Moto’, ‘moto’, ‘Moto G’).

We added a new feature named ‘device_name’ which had all the brands and OS represented in different style integrated to one common name.

Here is the snippet of what exactly is done.

```
train.loc[train['device_name'].str.contains('SM', na=False), 'device_name'] = 'Samsung'
train.loc[train['device_name'].str.contains('SAMSUNG', na=False), 'device_name'] = 'Samsung'
train.loc[train['device_name'].str.contains('GT-', na=False), 'device_name'] = 'Samsung'
train.loc[train['device_name'].str.contains('Moto G', na=False), 'device_name'] = 'Motorola'
train.loc[train['device_name'].str.contains('Moto', na=False), 'device_name'] = 'Motorola'
train.loc[train['device_name'].str.contains('moto', na=False), 'device_name'] = 'Motorola'
train.loc[train['device_name'].str.contains('LG-', na=False), 'device_name'] = 'LG'
train.loc[train['device_name'].str.contains('rv:', na=False), 'device_name'] = 'RV'
train.loc[train['device_name'].str.contains('HUAWEI', na=False), 'device_name'] = 'Huawei'
train.loc[train['device_name'].str.contains('ALE-', na=False), 'device_name'] = 'Huawei'
train.loc[train['device_name'].str.contains('-L', na=False), 'device_name'] = 'Huawei'
train.loc[train['device_name'].str.contains('Blade', na=False), 'device_name'] = 'ZTE'
train.loc[train['device_name'].str.contains('BLADE', na=False), 'device_name'] = 'ZTE'
train.loc[train['device_name'].str.contains('Linux', na=False), 'device_name'] = 'Linux'
train.loc[train['device_name'].str.contains('XT', na=False), 'device_name'] = 'Sony'
train.loc[train['device_name'].str.contains('HTC', na=False), 'device_name'] = 'HTC'
train.loc[train['device_name'].str.contains('ASUS', na=False), 'device_name'] = 'Asus'
```

2. Transaction ID and Transaction DT

We removed both ID and date and time feature.

Transaction ID is unique for all the transactions and it does not make any sense to the dataset as this cannot be used to distinguish any transaction.

Date and time just represent the timings of the transaction so we thought of not keeping it.

3. Column Removal

We removed the V columns and categorical features with excessive NULL values (i.e., >80%). The id, C, M and D columns were removed based on correlation. The correlation matrix was used to visualize the highly correlated values as shown in EDA above.

This is the snippet of how correlation was used to see which columns are highly correlated and remove those columns.

```
D_features = [ 'D1','D2','D3','D4','D5','D6','D7','D8','D9','D10','D11','D12','D13','D14','D15']
```

```
corr_D = train[D_features].corr()
plt.subplots(figsize = (20, 20))
sns.heatmap(corr_D,annot=True, cmap = 'YlGnBu')
sns.set(font_scale=1)
```

4. Replacing NULL values

We replaced the null values of all the remaining numerical features after column removal by its median value and categorical features by its mode. We used median in place of mean for numerical columns because we had very little knowledge about the outliers in the data. If the outliers are less then only mean is recommended otherwise median should be used.

Below is the snippet of how we replaced the null values.

```
main_df["D1"].fillna(main_df["D1"].median(), inplace=True)
main_df["D3"].fillna(main_df["D3"].median(), inplace=True)
main_df["D4"].fillna(main_df["D4"].median(), inplace=True)
main_df["D5"].fillna(main_df["D5"].median(), inplace=True)
main_df["D6"].fillna(main_df["D6"].median(), inplace=True)
main_df["D10"].fillna(main_df["D10"].median(), inplace=True)
main_df["D11"].fillna(main_df["D11"].median(), inplace=True)
main_df["D15"].fillna(main_df["D15"].median(), inplace=True)
```

```
main_df["P_emaildomain"].fillna("gmail.com", inplace=True)
```

```
gmail.com is the mode of P_emaildomain
```

5. Categorical Encoding

We performed both one-hot encoding and categorical encoding. For features with less unique values (like ‘Product CD’, ‘card4’, ‘card6’, etc.) one-hot encoding is done. But for features with large number of unique values like 60, feature encoding was performed so that the number of features does not increase drastically. Features like ‘id_30’, ‘P_emaildomain’ and ‘R_emaildomain’ had 60 unique values.

```
column_names_to_one_hot = ["ProductCD", "card4", "card6", "M1", "M2", "
```

```
main_df = pd.get_dummies(main_df, columns=column_names_to_one_hot)
```

```
main_df['id_30'] = le.fit_transform(main_df['id_30'])
main_df['R_emaildomain'] = le.fit_transform(main_df['R_emaildomain'])
main_df['P_emaildomain'] = le.fit_transform(main_df['P_emaildomain'])
```

6. Re-sampling

Our dataset is highly imbalance i.e., we have about 96.5% non-fraud data and only 3.5% fraud data. This can create a bias in the model towards non-fraud and can also lead to overfitting.

To overcome this issue, we did random oversampling and random under sampling in different ratios. Over sampling was done such that the fraud to non-fraud ratio becomes 0.1 and then under sampling was done such that the same ratio becomes 0.5.

```
over = RandomOverSampler(sampling_strategy=0.1)
under = RandomUnderSampler(sampling_strategy=0.5)
```

```
pipeline = Pipeline(steps=[('o', over), ('u', under)])
train_X_re, train_y_re = pipeline.fit_resample(X_train, y_train)
```

Models

Model Tuned	Hyper parameters	Model score
Logistic regression	Max_iter = 10000	0.693
Naïve bayes’	Var_smoothing = 1e-12	0.698
KNN	N_neighbours = 3	0.69
Decision tree	-	0.797
Random forest	n_estimators = 2000	0.809
XGBoost	n_estimatores = 2000 learning_rate = 0.2 max_depth = 12 gamma = 0.1	0.895
Neural networks	-	0.508

Hyper parameter tuning

Hyper parameter tuning was done by applying grid search cv on different models. Below is the snippet of one of the tests done on XGBoost model.

```
param_test2 = {
'n_estimators' : [2000, 2500, 3000],
'max_depth': [8, 10, 12],
'min_child_weight': [1, 2]
}

gsearch_2 = GridSearchCV(estimator=xgb_classifier, param_grid=param_test2, scoring='roc_auc', cv=3)

Best parameters: {'max_depth': 12, 'min_child_weight': 1, 'n_estimators': 2000}
```

Conclusion

After all the pre-processing and model tuning we got the best score with xgboost model.

Public score: 0.8909

Private score: 0.89512