It’s a fraud

AI 511 Machine Learning

Team : Mod Good

Yash Koushik (IMT2020033)

Murali Jayam (IMT2020035)

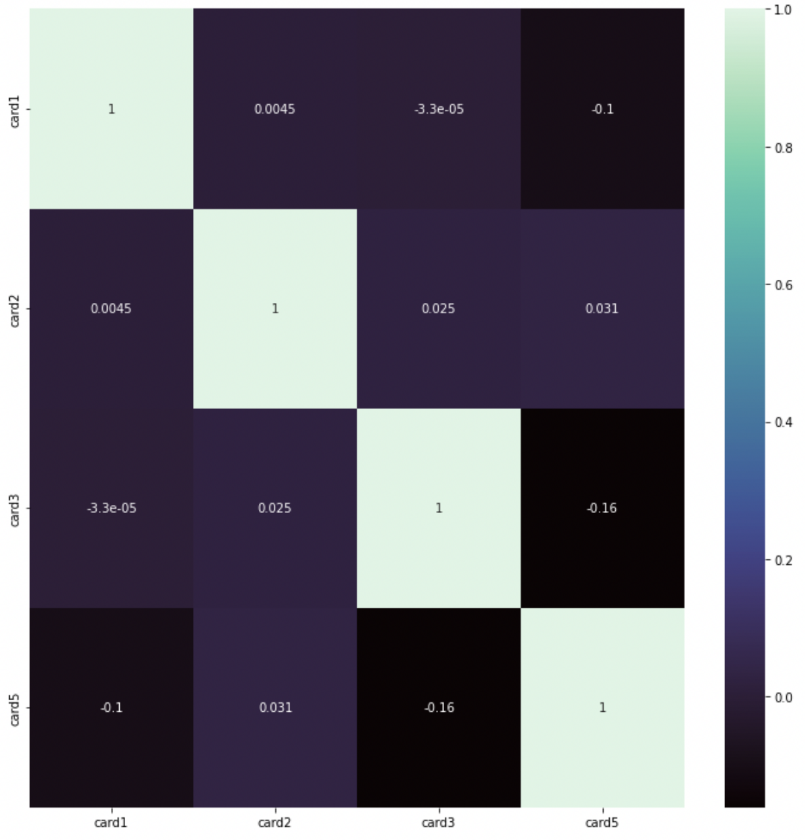
Task

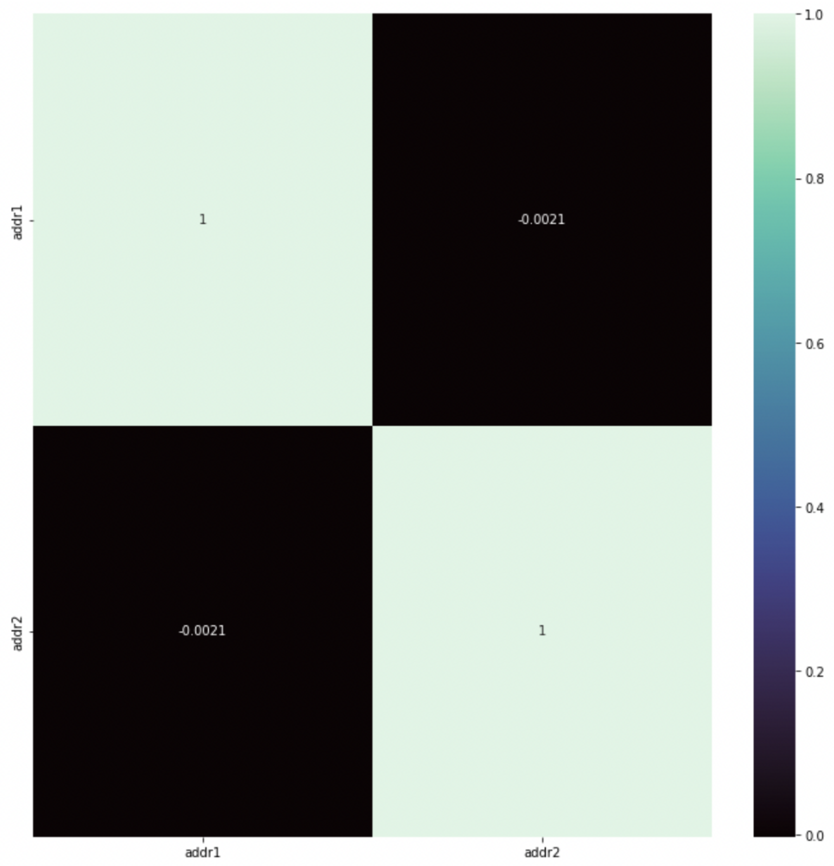
In this Project we are predicting whether an online transaction is fraudulent or not.

Preprocessing:

- There are 110 columns which have less than 10000 NaN values. We will collect the names of these columns and drop the corresponding rows where these column values are NaN.
- There are 12 columns which have 90% values as NaN, so we have dropped these columns.
- We have observed that 96.5% of our dataset contains the transaction not being fraud and 3.5% transactions being fraud.
- There is no correlation between any of the cards columns and card4 and card6 have categorical values, therefore did not showup in
  The heatmap.

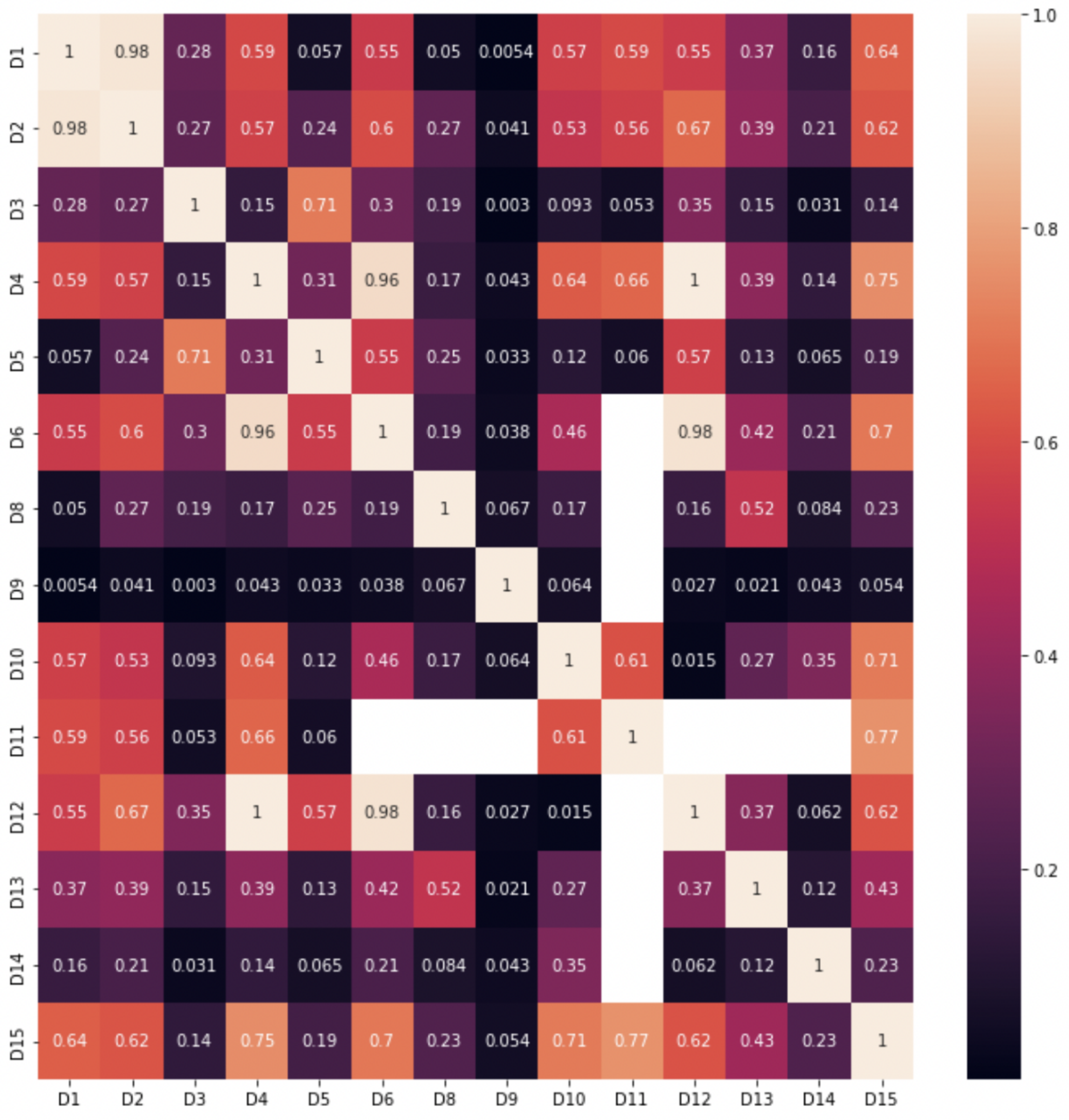
- addr1 and addr2 contain NaN values and these are not correlated. Hence we replace these NaN values with the mean.





- The dist1 column has erratic data. Hence we replace the NaN values with mode.(Most frequently occuring)
- The following pairs of columns have high correlation.
    - D4, D6
    - D1, D2
    - D12, D6
    - D4, D12

  So, we wish to drop D6, D12 and D2.

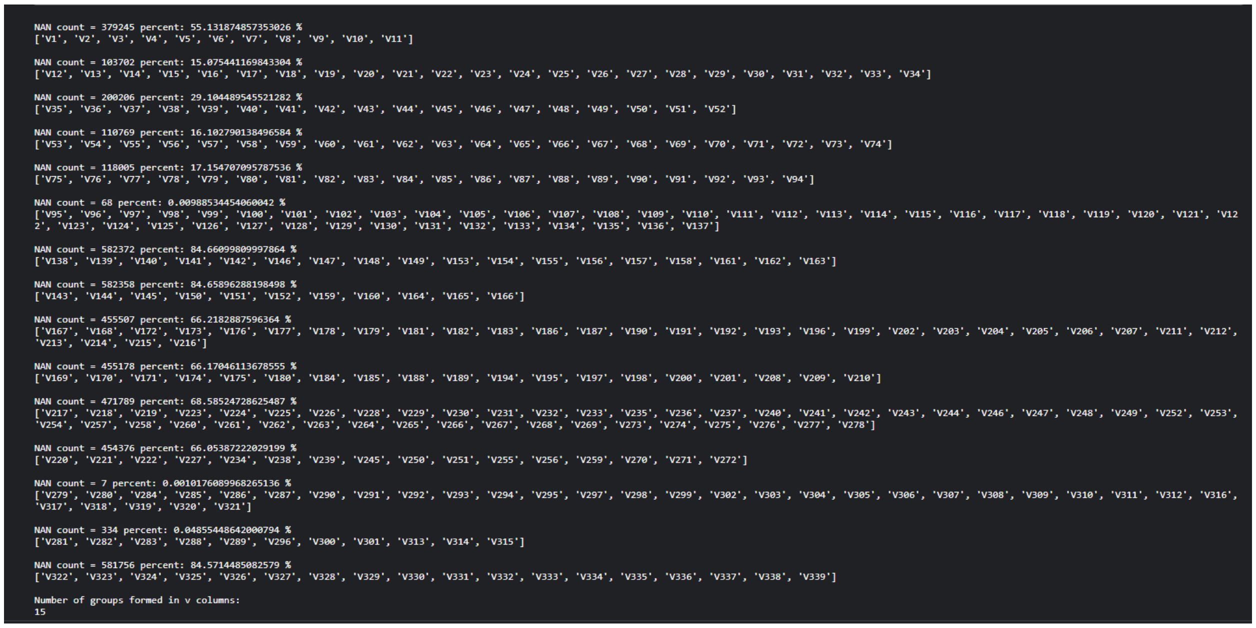


- There are 339, V columns (V1 - V339). Plotted heatmaps of 339 columns making multiple groups (Group 1 - Group 15).
    - Group the columns based on number of missing values
    - For each group:

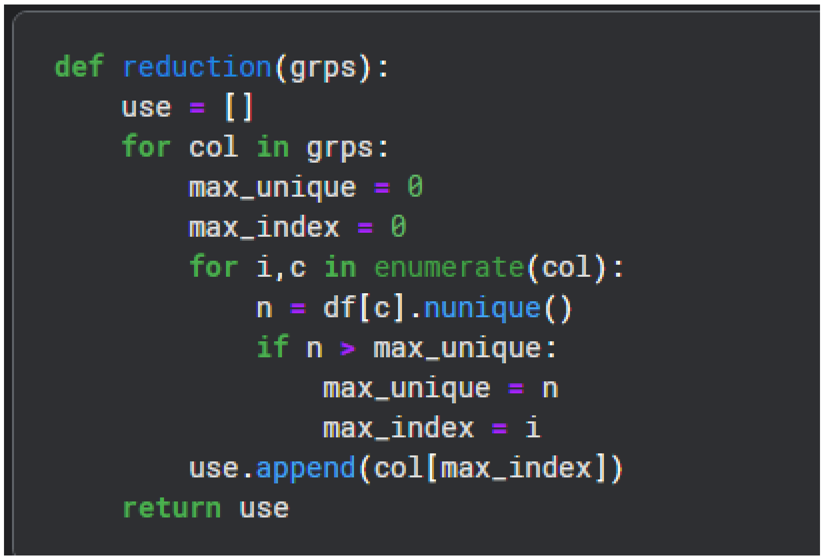
      For each column in that group find the correlation with other columns and take only columns with correlation coefficient > 0.75

    - Take the largest list with common elements as a subgroup.Each group contains several subgroups.

- Now from each subgroup choose the column with the most number of unique values.



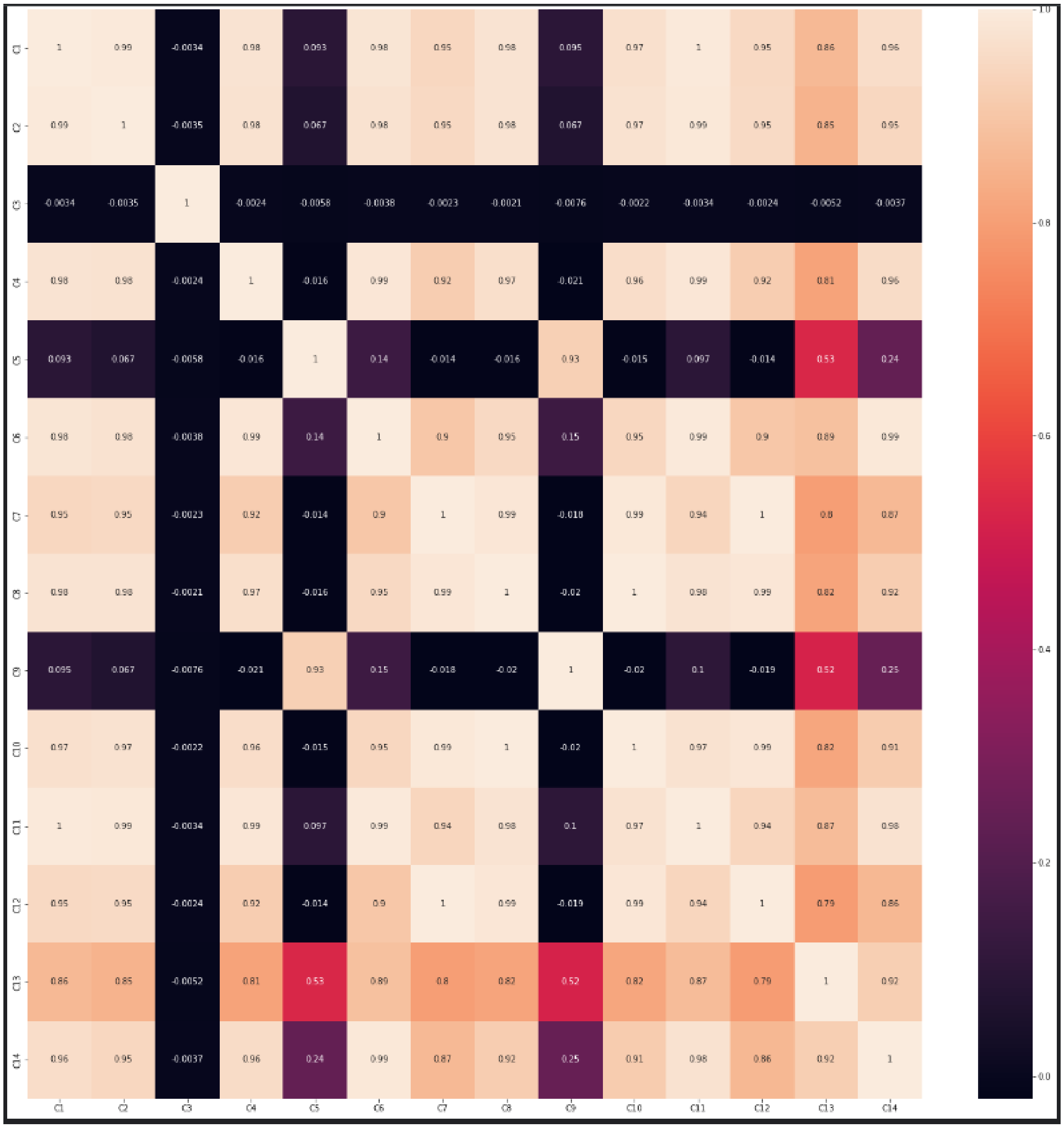
```
NAN count = 379245 percent: 55.131874857353026 %
['V1', 'V2', 'V3', 'V4', 'V5', 'V6', 'V7', 'V8', 'V9', 'V10', 'V11']

NAN count = 103702 percent: 15.075441169843304 %
['V12', 'V13', 'V14', 'V15', 'V16', 'V17', 'V18', 'V19', 'V20', 'V21', 'V22', 'V23', 'V24', 'V25', 'V26', 'V27', 'V28', 'V29', 'V30', 'V31', 'V32', 'V33', 'V34']

NAN count = 200206 percent: 29.104489545521282 %
['V35', 'V36', 'V37', 'V38', 'V39', 'V40', 'V41', 'V42', 'V43', 'V44', 'V45', 'V46', 'V47', 'V48', 'V49', 'V50', 'V51', 'V52']

NAN count = 110769 percent: 16.102790138496584 %
['V53', 'V54', 'V55', 'V56', 'V57', 'V58', 'V59', 'V60', 'V61', 'V62', 'V63', 'V64', 'V65', 'V66', 'V67', 'V68', 'V69', 'V70', 'V71', 'V72', 'V73', 'V74']

NAN count = 118005 percent: 17.154707095787536 %
['V75', 'V76', 'V77', 'V78', 'V79', 'V80', 'V81', 'V82', 'V83', 'V84', 'V85', 'V86', 'V87', 'V88', 'V89', 'V90', 'V91', 'V92', 'V93', 'V94']

NAN count = 68 percent: 0.00988534454060042 %
['V95', 'V96', 'V97', 'V98', 'V99', 'V100', 'V101', 'V102', 'V103', 'V104', 'V105', 'V106', 'V107', 'V108', 'V109', 'V110', 'V111', 'V112', 'V113', 'V114', 'V115', 'V116', 'V117', 'V118', 'V119', 'V120', 'V121', 'V12
2', 'V123', 'V124', 'V125', 'V126', 'V127', 'V128', 'V129', 'V130', 'V131', 'V132', 'V133', 'V134', 'V135', 'V136', 'V137']

NAN count = 582372 percent: 84.66099809997864 %
['V138', 'V139', 'V140', 'V141', 'V142', 'V146', 'V147', 'V148', 'V149', 'V153', 'V154', 'V155', 'V156', 'V157', 'V158', 'V161', 'V162', 'V163']

NAN count = 582358 percent: 84.65896288198498 %
['V143', 'V144', 'V145', 'V150', 'V151', 'V152', 'V159', 'V160', 'V164', 'V165', 'V166']

NAN count = 455507 percent: 66.2182887596364 %
['V167', 'V168', 'V172', 'V173', 'V176', 'V177', 'V178', 'V179', 'V181', 'V182', 'V183', 'V186', 'V187', 'V190', 'V191', 'V192', 'V193', 'V196', 'V199', 'V202', 'V203', 'V204', 'V205', 'V206', 'V207', 'V211', 'V212',
'V213', 'V214', 'V215', 'V216']

NAN count = 455178 percent: 66.17046113678555 %
['V169', 'V170', 'V171', 'V174', 'V175', 'V180', 'V184', 'V185', 'V188', 'V189', 'V194', 'V195', 'V197', 'V198', 'V200', 'V201', 'V208', 'V209', 'V210']

NAN count = 471789 percent: 68.58524728625487 %
['V217', 'V218', 'V219', 'V223', 'V224', 'V225', 'V226', 'V228', 'V229', 'V230', 'V231', 'V232', 'V233', 'V235', 'V236', 'V237', 'V240', 'V241', 'V242', 'V243', 'V244', 'V246', 'V247', 'V248', 'V249', 'V252', 'V253',
'V254', 'V257', 'V258', 'V260', 'V261', 'V262', 'V263', 'V264', 'V265', 'V266', 'V267', 'V268', 'V269', 'V273', 'V274', 'V275', 'V276', 'V277', 'V278']

NAN count = 454376 percent: 66.05387222029199 %
['V220', 'V221', 'V222', 'V227', 'V234', 'V238', 'V239', 'V245', 'V250', 'V251', 'V255', 'V256', 'V259', 'V270', 'V271', 'V272']

NAN count = 7 percent: 0.0010176089968265136 %
['V279', 'V280', 'V284', 'V285', 'V286', 'V287', 'V290', 'V291', 'V292', 'V293', 'V294', 'V295', 'V297', 'V298', 'V299', 'V302', 'V303', 'V304', 'V305', 'V306', 'V307', 'V308', 'V309', 'V310', 'V311', 'V312', 'V316',
'V317', 'V318', 'V319', 'V320', 'V321']

NAN count = 334 percent: 0.04855448642000794 %
['V281', 'V282', 'V283', 'V288', 'V289', 'V296', 'V300', 'V301', 'V313', 'V314', 'V315']

NAN count = 581756 percent: 84.5714485082579 %
['V322', 'V323', 'V324', 'V325', 'V326', 'V327', 'V328', 'V329', 'V330', 'V331', 'V332', 'V333', 'V334', 'V335', 'V336', 'V337', 'V338', 'V339']

Number of groups formed in v columns:
15
```



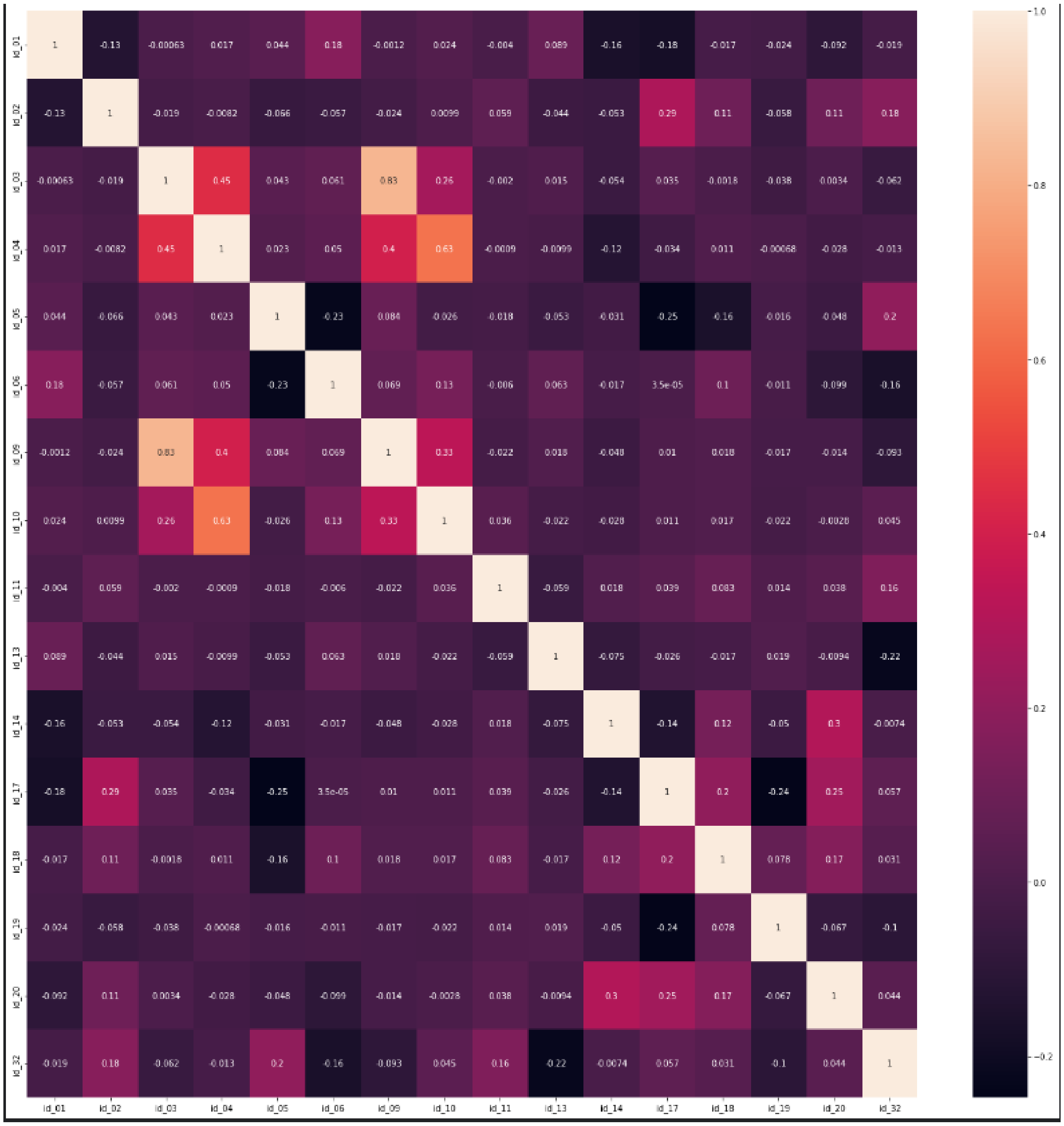
```python
def reduction(grps):
    use = []
    for col in grps:
        max_unique = 0
        max_index = 0
        for i,c in enumerate(col):
            n = df[c].nunique()
            if n > max_unique:
                max_unique = n
                max_index = i
        use.append(col[max_index])
    return use
```

- Used the correlation matrix to reduce the c columns, by looking at the correlation between the columns.

- ○ Except for C3, C5, C9 and C13 every other c column is highly correlated, so we drop them all.



- In id columns, for the numerical columns we again used the correlation matrix, similar to the before step, and for the categorical columns we did label encoding and as far as the null values are concerned, we imputed them for the numerical columns and we replaced them with mode for the categorical (changed to label encoding numbers) columns.



- For all the categorical columns, if the unique values are too high, then we used label encoding. If they are too low then we used one hot encoding.

Fitting different models:

Logistic Regression:

Parameters:

- ➢ **C: float, default =** 1.0

  Inverse of regularization strength; must be a positive float. Like in support vector machines, smaller values specify stronger regularization.

- ➢ **Penalty {‘l1’, ‘l2’, ‘elastic net’, None}, default =** ’l2’

  Specify the norm of the penalty:

- ★ **Tuned hyperparameters:** C = 100, Penalty: l2

  Accuracy: 56.2

Naive Bayes:

Parameters:

- ➢ **Var_smoothing:** float, default = 1e-9

  Portion of the largest variance of all features that is added to variances for calculation stability.

- ★ **Tuned Parameters:** 1e-9

Accuracy: 60

KNN:

Parameters:

➢ **n_neighbors** (default = 5)

Number of neighbors to use by default for kneighbors queries.

➢ **weights** (default = uniform)

Weight function used in prediction. Possible values:

- ‘uniform’ : uniform weights. All points in each neighborhood are weighted equally.
- ‘distance’ : weight points by the inverse of their distance. in this case, closer neighbors of a query point will have a greater influence than neighbors which are further away.
- [callable] : a user-defined function which accepts an array of distances, and returns an array of the same shape containing the weights.

★ Tuned parameters: n_neigbors = 6, weight= uniform

Accuracy = 61.5

XGBoost:

Parameters:

➢ **Max_depth:** (default = 6)

Maximum depth of a tree. Increasing this value will make the model more complex and more likely to overfit. 0 indicates no limit on depth. Beware that XGBoost aggressively consumes memory when training a deep tree, exact tree method requires non-zero value.

➢ **N_estimators:** (default = 100)

The number of runs XGBoost will try to learn.

➢ **Learning rate:** (default = 0.3)

The learning rate is the shrinkage you do at every step you are making.

★ **Tuned hyperparameters:** colsample_bylevel = 1, colsample_bytree = 0.8,

Gpu_id = -1, learning_rate = 0.02, max_depth = 12, n_estimators = 800,

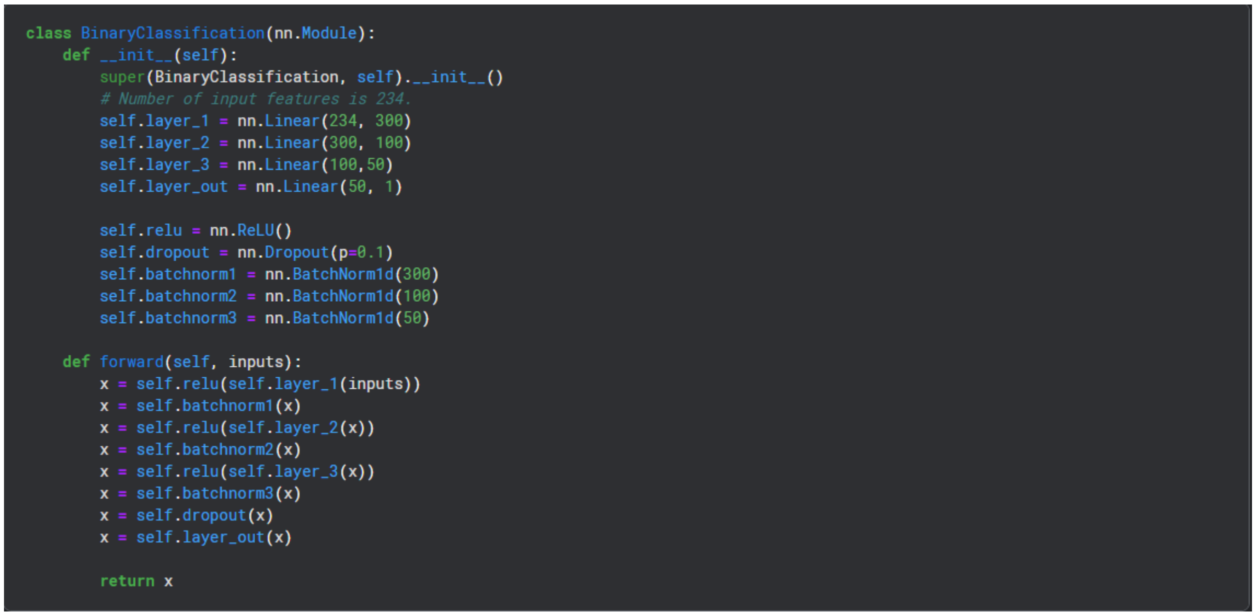
Random_state = 0, subsample = 0.3, tree_method = 'gpu_hist'

Accuracy: 88.2 (hard limit), 96.4 (soft limit)

Neural Network:

Number of layers: 3

Activation function used in each layer: Relu



```python
class BinaryClassification(nn.Module):
    def __init__(self):
        super(BinaryClassification, self).__init__()
        # Number of input features is 234.
        self.layer_1 = nn.Linear(234, 300)
        self.layer_2 = nn.Linear(300, 100)
        self.layer_3 = nn.Linear(100,50)
        self.layer_out = nn.Linear(50, 1)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(p=0.1)
        self.batchnorm1 = nn.BatchNorm1d(300)
        self.batchnorm2 = nn.BatchNorm1d(100)
        self.batchnorm3 = nn.BatchNorm1d(50)

    def forward(self, inputs):
        x = self.relu(self.layer_1(inputs))
        x = self.batchnorm1(x)
        x = self.relu(self.layer_2(x))
        x = self.batchnorm2(x)
        x = self.relu(self.layer_3(x))
        x = self.batchnorm3(x)
        x = self.dropout(x)
        x = self.layer_out(x)

        return x
```

Accuracy: 83.8