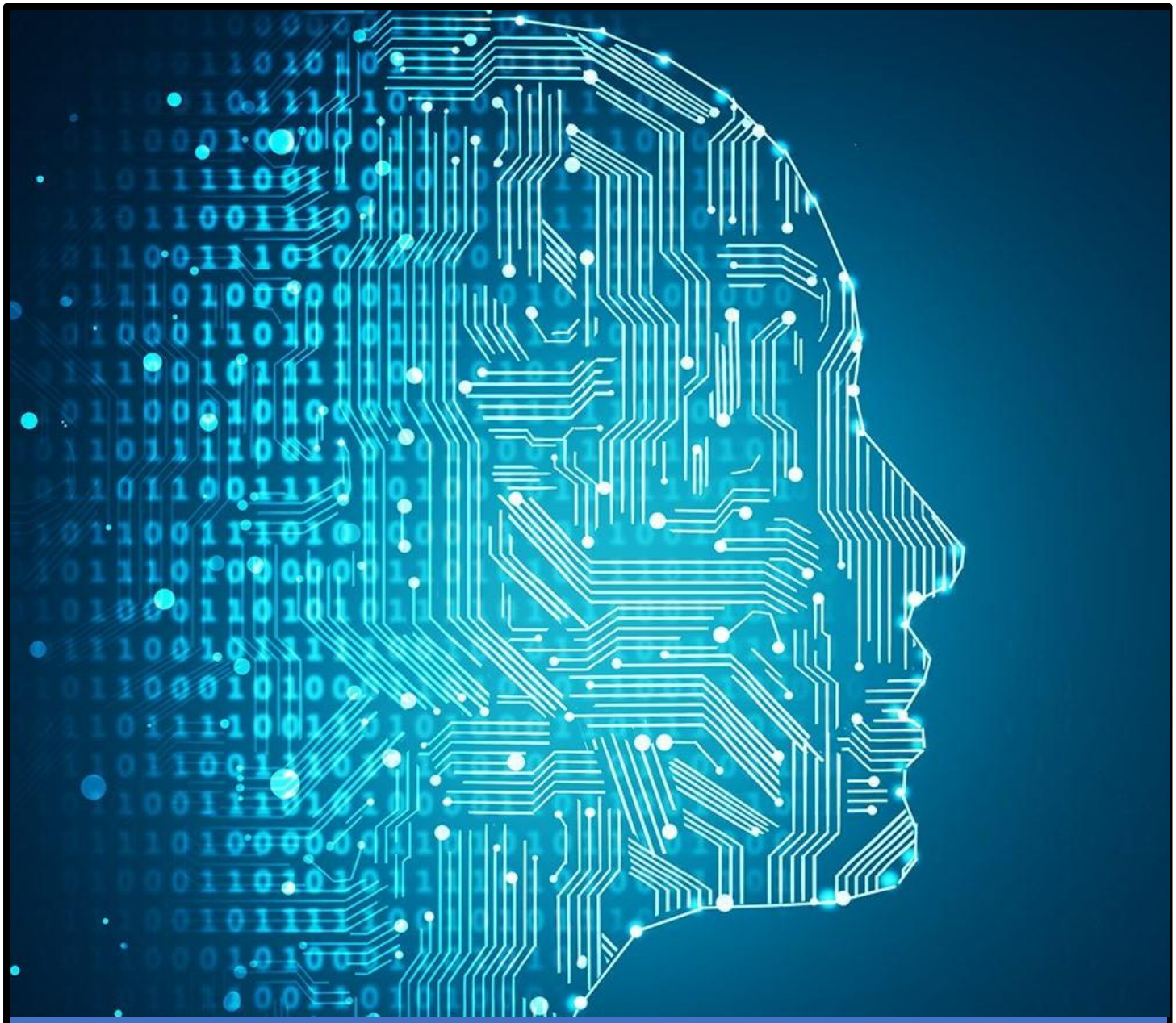


Fraud Detection via Predictive Analytics

A Machine Learning project to detect fraudulent transactions

Khizar Tahir

Objective:

To use Vesta corporation's real-world e-commerce transaction data along with identity data of transactors to create multiple machine learning models and use the best to accurately predict fraudulent transactions so false positives and false negatives are reduced.

Data:

For this project, there are primarily two sets of data that come from Vesta corporation. The first is online transaction data that has hundreds of columns for a mix of numerical and categorical variables including a transaction ID, transaction amount, product code, payment card information, address, etc. Transactions data also contains our variable of interest, the 'isFraud' binary variable that is 0 when a transaction is genuine, and 1 if fraudulent. The second dataset contained identity information of the transactor. The information was masked so as to avoid actual identification of personnel. This dataset had information about the network, digital signature, device and browser information, etc. these two datasets are joined by a common column Transaction ID.

Both these datasets were further broken down into a training and a test dataset. The training dataset included the isFraud column for training our models and the test data had the isFraud column removed for testing. As a result, we had 4 datasets: train_identity, train_transaction, test_identity, and test_transaction.

Data cleaning/Preparation:

Keeping in mind the vastness of our datasets, we begin to clean it and make it suitable for our use. We do some initial filtering of columns based on domain knowledge and the variables we want to study. From the transactions data, we keep the transaction ID, isFraud, amount, product code, card information as the variables we want to analyze. As a starting step, we retain all columns of the identity data as we suspect they define the outcome more.

Next, we join the training transaction data with the training identity data such that all columns of identity data are retained and only the specified columns of transaction data are chosen. Rows from the two datasets are joined on the transaction ID column. After this step, our resulting dataset has about 144,000 rows. We then calculate the total missing values in each column and remove those variables with more than approximately 5% missing values. There were variables with most data missing. Those are removed from analysis as first filteration as there needs to be sufficient data for imputation. The variables left are then separated as numerical or categorical. Missing numerical variable values present now are imputed with median values, missing categorical variable values are imputed with 'na' for not available.

We observed that some categorical variables had values that were not exactly the same but meant the same. This was mostly in the browser name and version column and the operating software of device column. We coded them such that similar values fell under one category. For example, various versions of Chrome browser all were grouped under Chrome instead of being treated separately.

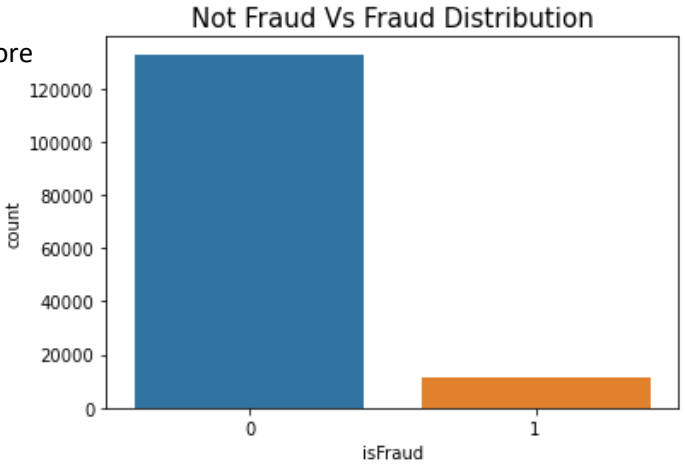
After all these steps, our data is for some exploratory data analysis.

Exploratory Data Analysis:

In this section, we explore and visualize the data that we have to see if we can identify some trends and generate insights.

First, we plot a bar graph counting the number of times the transaction came out fraudulent versus not fraudulent.

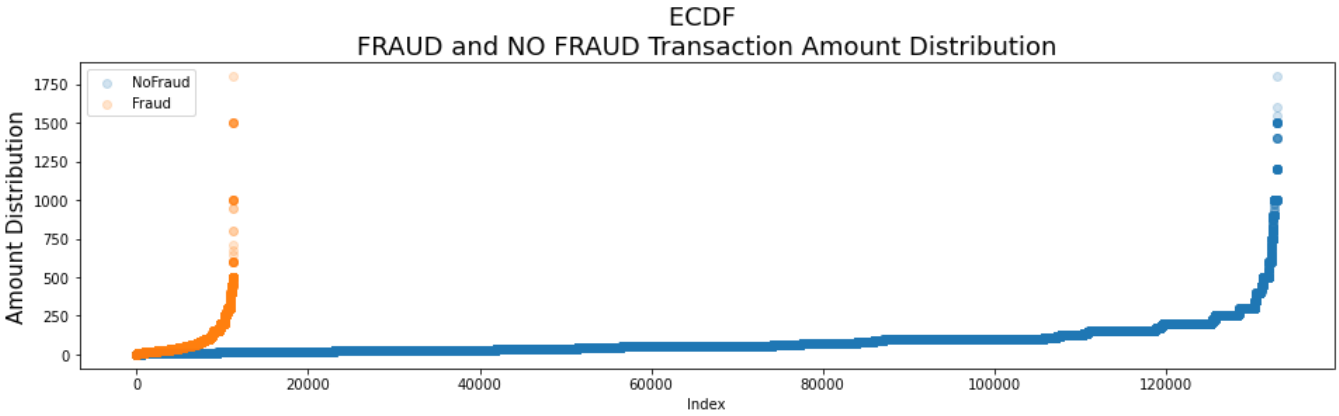
The graph on the right shows that a great majority of transactions are not fraudulent. Only some are. In fact, more than 90% are genuine transactions in our training dataset.



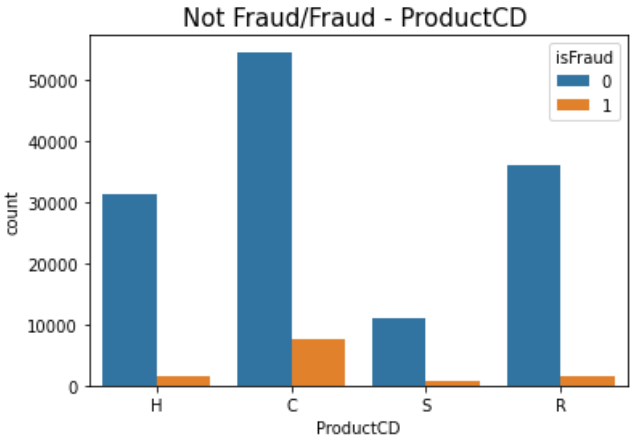
To get an idea of the distribution of the transaction amounts across the two categories, we plot the graph that superimposes the amount distribution plots for the two categories.

We observe from graph below that most fraudulent transactions are for low amounts – darker orange cluster around less amounts. This is probably because fraudulent transactors know that going for a big transaction will draw more attention.

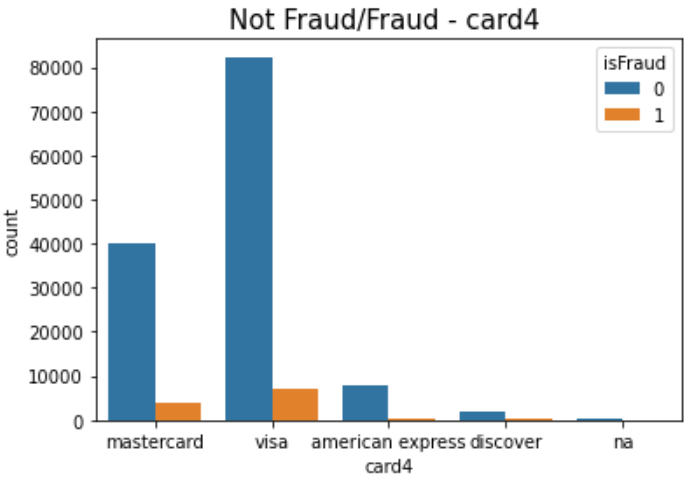
We can also observe that transaction amount is not a sole determinant of fraud because most transactions for any amount are still mostly non fraudulent. This variable works in conjunction with other determinants.

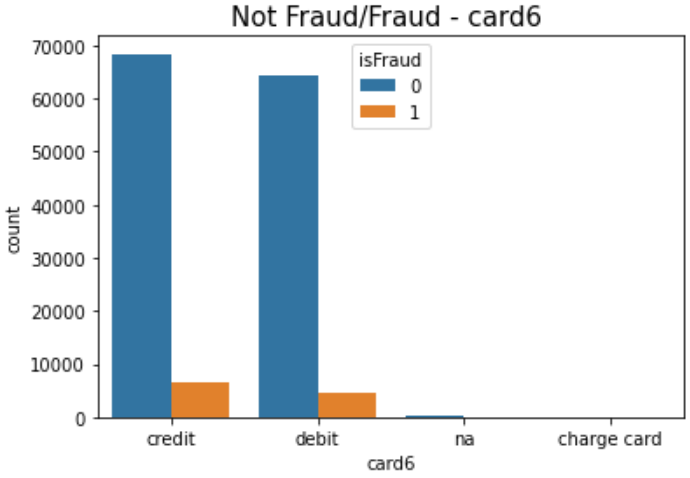


From the count plot of fraud versus the product code shown below, we can see that product C has had way more fraudulent transactions than other products. It is definitely more purchased too but fraud is alarmingly high.



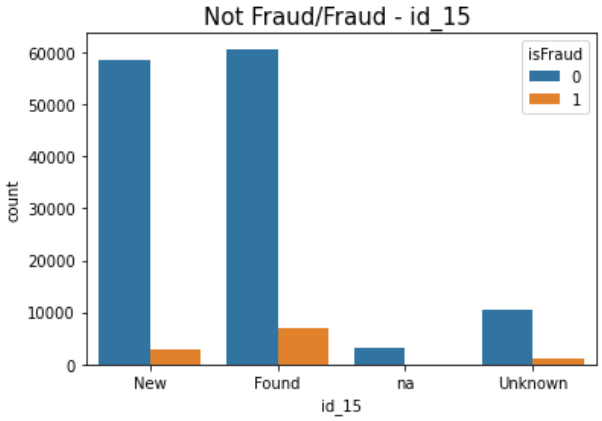
The count plots of cards and their categories below show Visa card as the most popular payment card with the highest fraud transactions followed by mastercard. Other options are not as popular. Credit cards are slightly more used than debit and also have a slightly higher fraudulent transactions count.

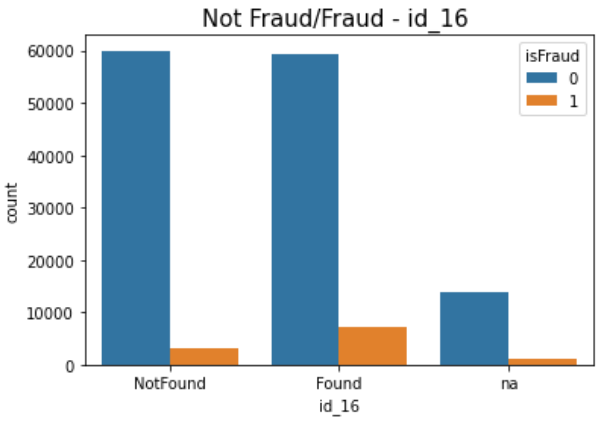


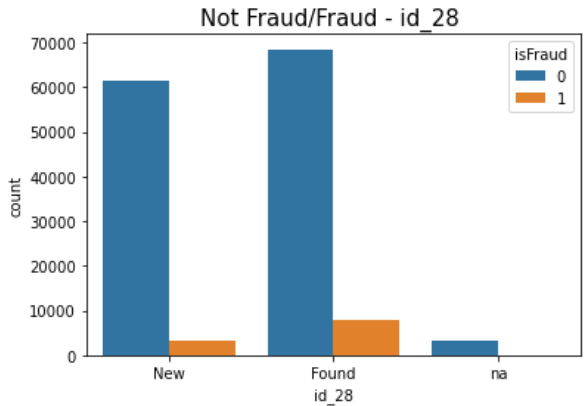


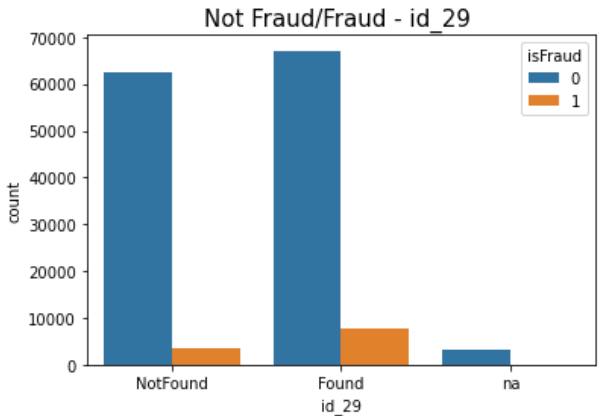
We suspect id_15 to be whether the transactor has a new account or not, and id_16 to be if any history of transaction is found or not. Both these variables perfectly match with very similar distributions. For when id_15 is New, id_16 is always Not found and vice versa. Id_28 and id_29 also have similar relation. Their distributions are below.

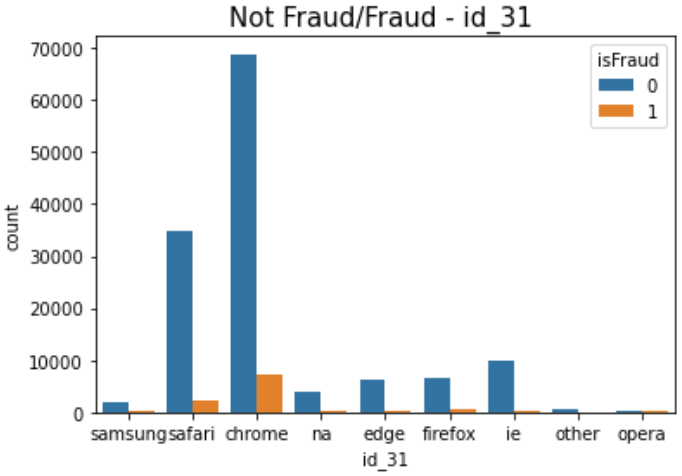
We observe that most fraudulent activities occurred when there was a history of transactor's transactions. This may be because malicious users find it easier to scam when someone's data is already in the system. They have limited information about a new user making it hard to conduct fraud.





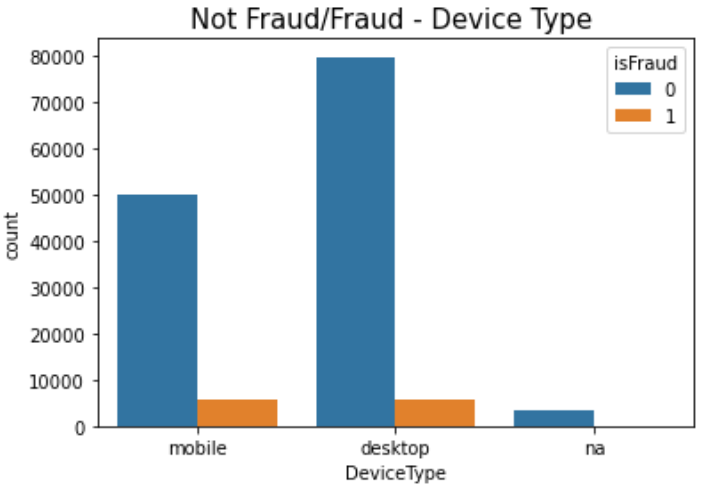


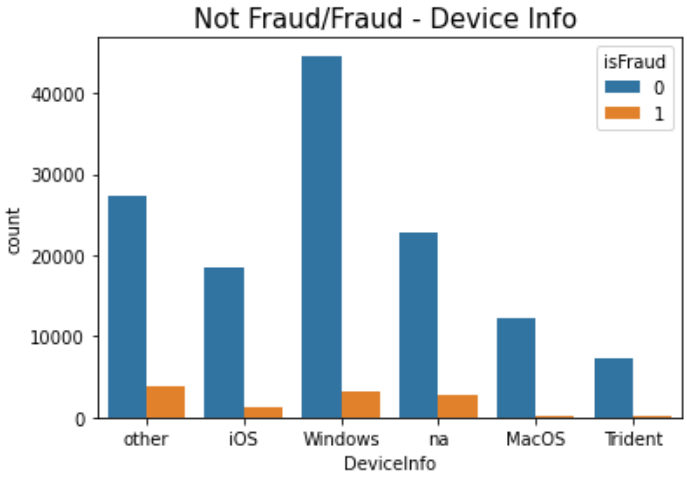




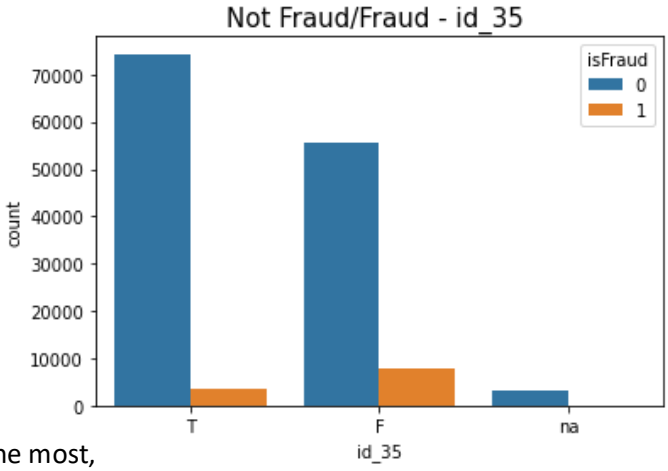
A count plot of browser used is shown on the left. We see that chrome is the most popular browser with also the highest fraudulent activities, followed by safari. Others are not as popular.

A plot of fraud with respect to device type is shown on the right. There are 2 categories. While there are more transactions through the desktop than mobile, the fraudulent activity counts are roughly the same.





A count plot of operating system used is shown on the left. We see that Windows is the most popular browser with decent number of fraudulent activities. We notice that there are significantly many operating systems that are not from among the most popular like Windows, iOS, or MacOS. These other systems have a very high fraudulent activity number. Many 'missing' systems details also have a high fraud rate.



Id_35 – id_38 are categorical variables with True (T) False (F) values. Although it is hard to understand what they may mean, the graph of id_35 on the right is interesting. It has high fraudulent transactions when F.

Insight:

Most of our graphs show that the fraudulent activities occur with the same attributes as the most frequent genuine transactions. Meaning, if Chrome is the most popular browser, most fraud activities are through Chrome, if product C is sold the most, most malicious transactions are for product C. It may just be due to numbers but it is also possible that malicious

users emulate real, most common transactions for fraud. They probably know that if they go for a huge amount, use a new payment method, or browser, they are likely to draw attention and get caught.

Creating a Predictive Model:

Now that we have a good understanding of our variables, we create our model. Our aim is not to build the best model in the first attempt. Rather, we will build several models and chose the best based of performance.

Preliminary steps:

First, we understand that our dataset is considerable in size and running machine learning models may consume too much time. So, we subset the data by randomly removing some rows to retain approximately half of training data.

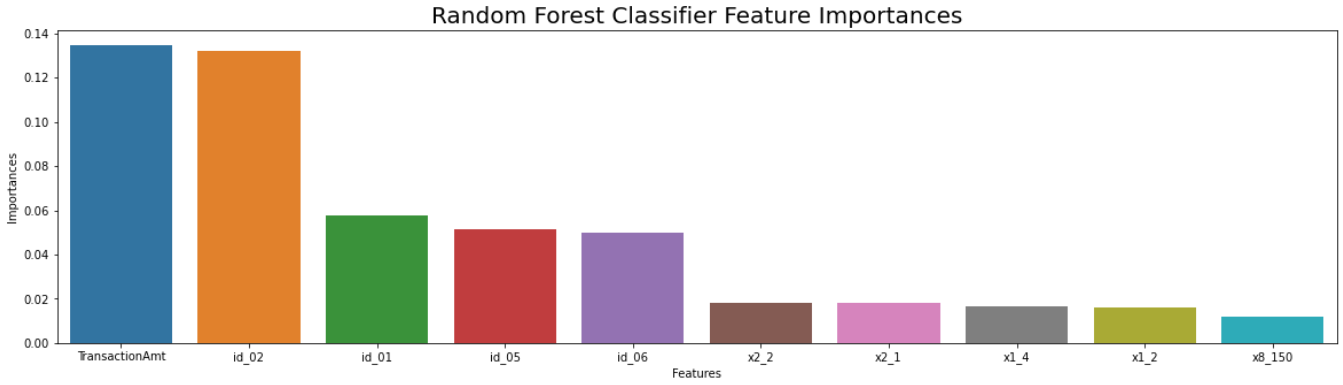
Secondly, we realize that our columns may have very different values, values accordingly to different scales or measurements. So, we scale our data as a standardizing procedure. This would enable fair and accurate comparisons and allow for a more robust model. We scale numeric and categorical variables separately and appropriately.

Thirdly, we split our training data (with isFraud column) into training and test. This way, we train our model and can test on data and get the model's accuracy as well. We use about 67% data as training, 33% for testing.

Random Forest Classifier:

Our first model is a random forest classifier. We create it by building 300 trees. After fitting the model on split training data, we check its prediction accuracy on the same data which was used to build it first. The accuracy comes out as 99.9%. We then check its prediction accuracy on our split **test** data. The accuracy comes out as **94.6%** which is very high.

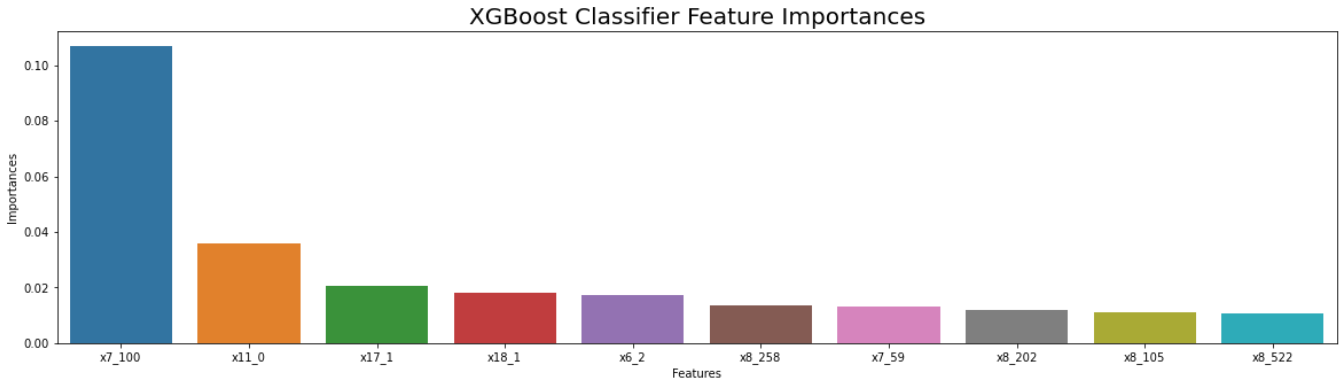
Below is a bar graph of feature importance as determined by our random forest classifier model. We can see that transaction amount and id_02 are the greatest determinants of outcome. Id_01, id_05, id_06 also contribute but not as much.



XGBoost Classifier:

Our second model is an XGBoost classifier. We create it by building 300 trees. After fitting the model on split training data, we check its prediction accuracy on the same data which was used to build it first. The accuracy comes out as 97.2%. We then check its prediction accuracy on our split **test** data. The accuracy comes out as **94.5%** which is very high as well.

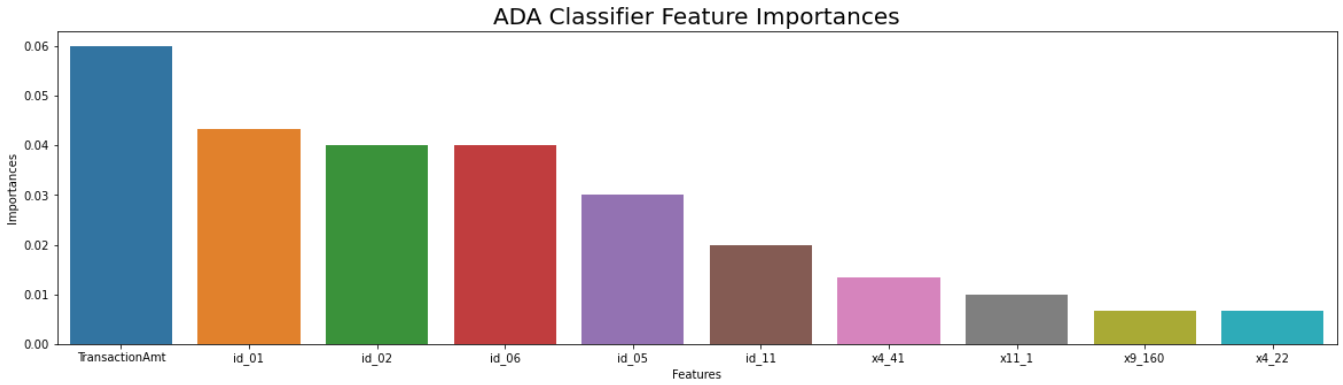
Below is a bar graph of feature importance as determined by our XGBoost classifier model. We can see that x7_100 which is a categorical scaled variable is determined to be the most important factor.



AdaBoost Classifier:

Our third model is an AdaBoost classifier. We create it by building 300 trees. After fitting the model on split training data, we check its prediction accuracy on the same data which was used to build it first. The accuracy comes out as 92.7%. We then check its prediction accuracy on our split **test** data. The accuracy comes out as **92.6%** which is good.

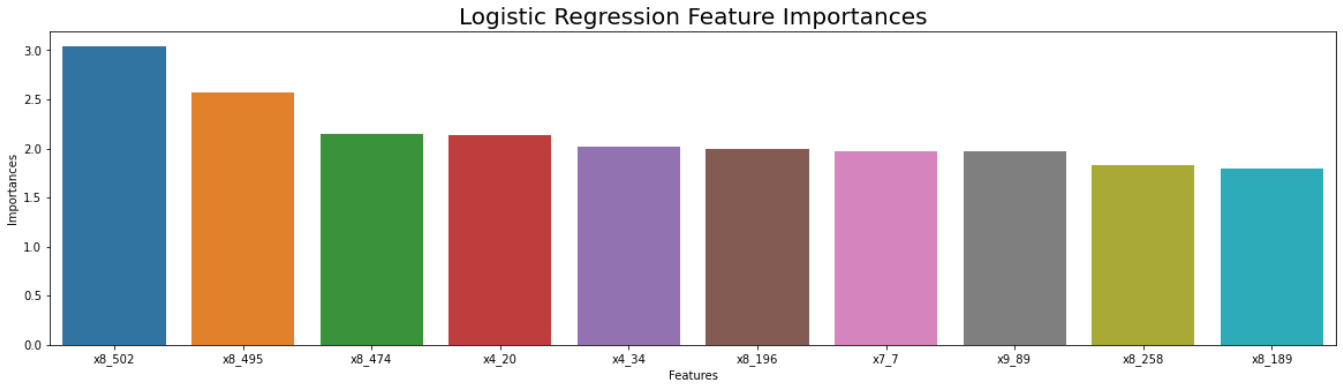
Below is a bar graph of feature importance as determined by our AdaBoost classifier model. We can see that transaction amount is the most important factor. Id_01, id_02, id_06, and id_05 also contribute substantially and are not that far behind.



Logistic Regression:

Our fourth and final model is logistic regression. After fitting the model on split training data, we check its prediction accuracy on the same data which was used to build it first. The accuracy comes out as 92.6%. We then check its prediction accuracy on our split **test** data. The accuracy comes out as **92.4%** which is good.

Below is a bar graph of feature importance as determined by our logistic regression model. We can see that x8_502 and x8_495 which are categorical scaled variables are determined to be the most important factors. Unlike other models made earlier, logistic regression considers a lot of variables to be significant.



Random Forest Classifier on Full training data:

Looking at the results of our four models, we decide to move ahead with the random forest classifier model that had the highest prediction accuracy on split test data. The model not only fit extremely well on the split training set, but also generalized well on the split test set.

We now split the data and run the same model on the whole dataset available to us – earlier we only ran on a subset. The random forest classifier on full data shows 99.9% prediction accuracy on data it was trained on, and shows **95.3%** prediction accuracy on the split test dataset. This is really good! This model performed even better on the full data because it had more data to train itself on.

Predicting on Final Test Data:

Now that we have our final model performing really well, we move onto the last step – predicting on data we cannot validate ourselves. Before running our prediction, we take the test_identity and test_transaction data through the same pipeline as our train datasets: data cleaning, filtering, feature selection, data imputation, scaling, etc.

Once the data is ready and in the same state as our training data was, we run our prediction. At the end, we create a file containing two columns only: transaction ID and isFraud containing our predictions.

Important notes:

A quick analysis of our prediction showed that our model determined almost all transactions as Not Fraud. Following points need to be considered for this:

- Our first EDA graph showed more than 90% of transactions to Not be fraud. The classes in our data were very imbalanced. By just classifying transactions to 0, our model achieves greater than 90% accuracy. This is great but may not be very practical.
- One solution to this is to Bootstrap the data while over-sampling the rare class and under-sampling the common class. Another solution is to use a "weighted" random forest where miss-classification of the minority class is assigned a higher cost.
- It is also possible that our model is overfitting on the training dataset. Although we tested on split test too but that was a way smaller dataset than our final test data to predict. For future work, other models could also be used to see if they generalize on test data any better than this.
- Many variables were dropped because their data was missing, or we did not have enough information about them. Having more information on variables could improve our analysis.

Conclusion:

From this project, we demonstrated the steps required to successfully and iteratively build a machine learning/predictive model. The insight about fraudulent users trying to emulate real, common transactions is something EDA showed us. We showed how various predictive models are built, evaluated, compared, and chosen. At last, we were able to predict fraudulent activities which is of great importance to Vesta corporation and could potentially save a lot of money and hassle.

Appendix:

```
import numpy as np

import pandas as pd

import joblib

from sklearn.model_selection import train_test_split

from sklearn import preprocessing

from sklearn.ensemble import RandomForestClassifier

from sklearn.pipeline import make_pipeline

from sklearn.model_selection import GridSearchCV,cross_val_score,cross_val_predict

from sklearn.metrics import mean_squared_error, r2_score

from sklearn.compose import ColumnTransformer, make_column_transformer

from sklearn.impute import SimpleImputer

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import LabelEncoder

from xgboost import XGBClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.linear_model import LogisticRegression

import matplotlib.pyplot as plt

import seaborn as sns

#from sklearn.externals import joblib

train_transaction = pd.read_csv("C:/RIT Courses/BANA 780 - Advanced Bus. Analytics/Problem 4/train_transaction.csv")

train_identity = pd.read_csv("C:/RIT Courses/BANA 780 - Advanced Bus. Analytics/Problem 4/train_identity.csv")

test_transaction = pd.read_csv("C:/RIT Courses/BANA 780 - Advanced Bus. Analytics/Problem 4/test_transaction.csv")

test_identity = pd.read_csv("C:/RIT Courses/BANA 780 - Advanced Bus. Analytics/Problem 4/test_identity.csv")

train_transaction.shape, test_transaction.shape, train_identity.shape, test_identity.shape

train_merge = pd.merge(train_transaction[['TransactionID','isFraud', 'TransactionAmt', 'ProductCD', 'card4', 'card6']],

                       train_identity,

                       on='TransactionID',

                       how='right')

print(train_merge.shape)

train_merge.isna().sum()

train_merge = train_merge.drop(columns=['id_03', 'id_04', 'id_07', 'id_08', 'id_09', 'id_10', 'id_14', 'id_18', 'id_21', 'id_22', 'id_23', 'id_24', 'id_25', 'id_26', 'id_27', 'id_30', 'id_32', 'id_33', 'id_34'])

categorical_features = ['ProductCD', 'card4', 'card6', 'id_12', 'id_13', 'id_15', 'id_16', 'id_17', 'id_19', 'id_20', 'id_28', 'id_29', 'id_31', 'id_35', 'id_36', 'id_37', 'id_38', 'DeviceType', 'DeviceInfo']

numeric_features = ['TransactionAmt', 'id_01', 'id_02','id_05','id_06','id_11']


for col in numeric_features:


train_merge[col].fillna(train_merge[col].median(), inplace=True)



for col in categorical_features:

    train_merge[col].fillna('na', inplace=True)

train_merge.loc[train_merge['id_31'].str.contains('samsung', case=False), 'id_31'] = 'samsung'

train_merge.loc[train_merge['id_31'].str.contains('chrome', case=False)& train_merge['id_31'].str.contains('android', case=False), 'id_31'] = 'chrome'

train_merge.loc[train_merge['id_31'].str.contains('chrome', case=False), 'id_31'] = 'chrome'

train_merge.loc[train_merge['id_31'].str.contains('ie', case=False), 'id_31'] = 'ie'

train_merge.loc[train_merge['id_31'].str.contains('edge', case=False), 'id_31'] = 'edge'

train_merge.loc[train_merge['id_31'].str.contains('firefox', case=False), 'id_31'] = 'firefox'

#result3.loc[result3['id_31'].str.contains('Android', case=False) | result3['id_31'].str.contains('android', case=False), 'id_31'] = 'android'

train_merge.loc[train_merge['id_31'].str.contains('opera', case=False), 'id_31'] = 'opera'

train_merge.loc[train_merge['id_31'].str.contains('safari', case=False), 'id_31'] = 'safari'

#for other

train_merge.loc[~train_merge['id_31'].isin(['samsung','chrome','ie','edge','firefox','opera','safari','na']), 'id_31'] = 'other'

train_merge.loc[train_merge['DeviceInfo'].str.contains('iOS', case=False), 'DeviceInfo'] = 'iOS'

train_merge.loc[train_merge['DeviceInfo'].str.contains('Trident', case=False), 'DeviceInfo'] = 'Trident'

train_merge.loc[train_merge['DeviceInfo'].str.contains('Windows', case=False), 'DeviceInfo'] = 'Windows'

train_merge.loc[train_merge['DeviceInfo'].str.contains('MacOS', case=False), 'DeviceInfo'] = 'MacOS'

#for other

train_merge.loc[~train_merge['DeviceInfo'].isin(['iOS','Trident','Windows','MacOS','na']), 'DeviceInfo'] = 'other'

sns.countplot(x="isFraud", data=train_merge).set_title('Not Fraud Vs Fraud Distribution', fontsize = 15)

fraud_result = train_merge.loc[train_merge['isFraud'] == 1]
```

```
notfraud_result = train_merge.loc[train_merge['isFraud'] == 0]

plt.figure(figsize=(15,6))

plt.subplot(121)

sns.distplot(fraud_result["TransactionAmt"])


plt.subplot(122)

sns.distplot(notfraud_result["TransactionAmt"])

plt.figure(figsize=(16,4))




plt.subplot()

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 0].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 0]['TransactionAmt'].values),

            label='NoFraud', alpha=.2)

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 1].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 1]['TransactionAmt'].values),

            label='Fraud', alpha=.2)

g4= plt.title("ECDF \nFRAUD and NO FRAUD Transaction Amount Distribution", fontsize=18)

g4 = plt.xlabel("Index")

g4 = plt.ylabel("Amount Distribution", fontsize=15)

g4 = plt.legend()

sns.countplot(x="ProductCD", hue="isFraud", data=train_merge).set_title('Not Fraud/Fraud - ProductCD ', fontsize = 15)

sns.countplot(x="card4", hue="isFraud", data=train_merge).set_title('Not Fraud/Fraud - card4 ', fontsize = 15)

sns.countplot(x="card6", hue="isFraud", data=train_merge).set_title('Not Fraud/Fraud - card6 ', fontsize = 15)

plt.figure(figsize=(16,4))




plt.subplot()

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 0].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 0]['id_01'].values),

            label='NoFraud', alpha=.2)

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 1].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 1]['id_01'].values),

            label='Fraud', alpha=.2)

g4= plt.title("Id_01 Distribution Not Fraud Vs Fraud", fontsize=18)

g4 = plt.xlabel("Index")

g4 = plt.ylabel("id_01 Distribution", fontsize=15)

g4 = plt.legend()

plt.figure(figsize=(16,4))




plt.subplot()

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 0].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 0]['id_02'].values),

            label='NoFraud', alpha=.2)

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 1].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 1]['id_02'].values),

            label='Fraud', alpha=.2)

g4= plt.title("Id_02 Distribution Not Fraud Vs Fraud", fontsize=18)

g4 = plt.xlabel("Index")

g4 = plt.ylabel("id_02 Distribution", fontsize=15)

g4 = plt.legend()

plt.figure(figsize=(16,4))




plt.subplot()

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 0].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 0]['id_05'].values),

            label='NoFraud', alpha=.2)

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 1].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 1]['id_05'].values),

            label='Fraud', alpha=.2)

g4= plt.title("Id_05 Distribution Not Fraud Vs Fraud", fontsize=18)

g4 = plt.xlabel("Index")

g4 = plt.ylabel("id_05 Distribution", fontsize=15)

g4 = plt.legend()

plt.figure(figsize=(16,4))




plt.subplot()

g4 = plt.scatter(range(train_merge[train_merge['isFraud'] == 0].shape[0]),


np.sort(train_merge[train_merge['isFraud'] == 0]['id_06'].values),

            label='NoFraud', alpha=.2)
```

```
g4 =
plt.scatter(range(train_merge[train_m
erge['isFraud'] == 1].shape[0]),


np.sort(train_merge[train_merge['isFr
aud'] == 1]['id_06'].values),

            label='Fraud', alpha=.2)

g4= plt.title("Id_06 Distribution Not
Fraud Vs Fraud", fontsize=18)

g4 = plt.xlabel("Index")

g4 = plt.ylabel("id_06 Distribution",
fontsize=15)

g4 = plt.legend()

plt.figure(figsize=(16,4))




plt.subplot()

g4 =
plt.scatter(range(train_merge[train_m
erge['isFraud'] == 0].shape[0]),


np.sort(train_merge[train_merge['isFr
aud'] == 0]['id_11'].values),

            label='NoFraud', alpha=.2)

g4 =
plt.scatter(range(train_merge[train_m
erge['isFraud'] == 1].shape[0]),


np.sort(train_merge[train_merge['isFr
aud'] == 1]['id_11'].values),

            label='Fraud', alpha=.2)

g4= plt.title("Id_11 Distribution Not
Fraud Vs Fraud", fontsize=18)

g4 = plt.xlabel("Index")

g4 = plt.ylabel("id_11 Distribution",
fontsize=15)

g4 = plt.legend()

sns.countplot(x="id_12",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_12 ',
fontsize = 15)

sns.countplot(x="id_15",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_15 ',
fontsize = 15)

sns.countplot(x="id_16",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_16 ',
fontsize = 15)

sns.countplot(x="id_28",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_28 ',
fontsize = 15)

sns.countplot(x="id_29",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_29 ',
fontsize = 15)

sns.countplot(x="id_31",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_31 ',
fontsize = 15)

sns.countplot(x="id_35",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_35 ',
fontsize = 15)

sns.countplot(x="id_36",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_36 ',
fontsize = 15)

sns.countplot(x="id_37",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_37 ',
fontsize = 15)

sns.countplot(x="id_38",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - id_38 ',
fontsize = 15)

sns.countplot(x="DeviceType",
hue="isFraud",
data=train_merge).set_title('Not
Fraud/Fraud - Device Type ', fontsize =
15)

sns.countplot(x="DeviceInfo",
hue="isFraud",data=train_merge).set_
title('Not Fraud/Fraud - Device Info ',
fontsize = 15)

labelencoder = LabelEncoder()


for col in categorical_features:

    train_merge[col] =
labelencoder.fit_transform(train_merg
e[col].astype(str))

#drop a bunch of rows to make this
more quick to compute

np.random.seed(10)

remove_n = 70000

drop_indices =
np.random.choice(train_merge.index,
remove_n, replace=False)

train_subset =
train_merge.drop(drop_indices)

train_subset.shape

train_subset =
train_subset.drop(columns =
['TransactionID'])

features = train_subset.drop(columns
= ['isFraud'])

Y = train_subset.isFraud

categorical_transformer =
Pipeline(steps=[

    ('onehot',
preprocessing.OneHotEncoder(handle
_unknown='ignore'))])


numeric_transformer =
Pipeline(steps=[

    ('scaler',
preprocessing.StandardScaler())])


preprocessor = ColumnTransformer(

    transformers=[

        ('num', numeric_transformer,
numeric_features),

        ('cat', categorical_transformer,
categorical_features)])

features_scaled =
preprocessor.fit_transform(features)

#X = data.drop('points', axis=1)

X_train, X_test, y_train, y_test =
train_test_split(features, Y,


test_size=0.33,


random_state=42)

rfc = Pipeline(steps=[('preprocessor',
preprocessor),

                      ('rfclassifier',
RandomForestClassifier(n_estimators=
300))])
```

```
rfc.fit(X_train, y_train)

rfc.score(X_train, y_train)

rfc.score(X_test, y_test)

cat_features =
rfc.named_steps['preprocessor'].transf
ormers_[1][1][0].get_feature_names()

cat_features

feature_names =
np.append(['TransactionAmt', 'id_01',
'id_02','id_05','id_06','id_11'],
cat_features)

feature_names

featimparray =
rfc.named_steps['rfclassifier'].feature_i
mportances_

feat_importances =
pd.Series(featimparray,
index=feature_names)

feat_importances.nlargest(20).plot(kin
d='barh')

rfc_features =
pd.DataFrame({'Features':
feature_names, 'Importances':
featimparray})

rfc_top20 =
rfc_features.sort_values('Importances',
ascending=False).head(10)

plt.figure(figsize=(20, 5))

sns.barplot(x="Features",
y="Importances",
data=rfc_top20).set_title('Random
Forest Classifier Feature Importances',
fontsize = 20)

xgbc = Pipeline(steps=[('preprocessor',
preprocessor),

                ('xgbc',
XGBClassifier(n_estimators=300))])

xgbc.fit(X_train, y_train)

xgbc.score(X_train, y_train)

xgbc.score(X_test, y_test)

featimparray_xgbc =
xgbc.named_steps['xgbc'].feature_imp
ortances_

feat_importances_xgbc =
pd.Series(featimparray_xgbc,
index=feature_names)

feat_importances_xgbc.nlargest(20).pl
ot(kind='barh')

xgbc_features =
pd.DataFrame({'Features':
feature_names, 'Importances':
featimparray_xgbc})

xgbc_top20 =
xgbc_features.sort_values('Importance
s',ascending=False).head(10)

plt.figure(figsize=(20, 5))

sns.barplot(x="Features",
y="Importances",
data=xgbc_top20).set_title('XGBoost
Classifier Feature Importances',
fontsize = 20)

adac = Pipeline(steps=[('preprocessor',
preprocessor),

                ('adac',
AdaBoostClassifier(n_estimators=300))
])

adac.fit(X_train, y_train)

adac.score(X_train, y_train)

adac.score(X_test, y_test)

featimparray_adac =
adac.named_steps['adac'].feature_imp
ortances_

feat_importances_adac =
pd.Series(featimparray_adac,
index=feature_names)

feat_importances_adac.nlargest(20).pl
ot(kind='barh')

adac_features =
pd.DataFrame({'Features':
feature_names, 'Importances':
featimparray_adac})

adac_top20 =
adac_features.sort_values('Importance
s',ascending=False).head(10)

plt.figure(figsize=(20, 5))

sns.barplot(x="Features",
y="Importances",
data=adac_top20).set_title('ADA
Classifier Feature Importances',
fontsize = 20)

lr = Pipeline(steps=[('preprocessor',
preprocessor),

                ('lr',
LogisticRegression(max_iter=10000))])

lr.fit(X_train, y_train)

lr.score(X_train, y_train)

lr.score(X_test, y_test)

featimparray_lr =
lr.named_steps['lr'].coef_[0]

feat_importances_lr =
pd.Series(featimparray_lr,
index=feature_names)

feat_importances_lr.nlargest(20).plot(
kind='barh')

lr_features =
pd.DataFrame({'Features':
feature_names, 'Importances':
featimparray_lr})

lr_top20 =
lr_features.sort_values('Importances',a
scending=False).head(10)

plt.figure(figsize=(20, 5))

sns.barplot(x="Features",
y="Importances",
data=lr_top20).set_title('Logistic
Regression Feature Importances',
fontsize = 20)

train_merge2 =
train_merge.drop(columns =
['TransactionID'])

features_full =
train_merge2.drop(columns =
['isFraud'])

Y_full = train_merge2.isFraud

len(features_full)

features_scaled_full =
preprocessor.fit_transform(features_f
ull)

features_scaled_full

X_train_full, X_test_full, y_train_full,
y_test_full =
train_test_split(features_full, Y_full,


test_size=0.33,


random_state=42)

X_train_full

rfc_full =
Pipeline(steps=[('preprocessor',
preprocessor),
```

```
                ('rfclassifier',
RandomForestClassifier(n_estimators=
300))])

rfc_full.fit(X_train_full, y_train_full)

rfc_full.score(X_train_full, y_train_full)

rfc_full.score(X_test_full, y_test_full)

test_merge =
pd.merge(test_transaction[['Transactio
nID', 'TransactionAmt', 'ProductCD',
'card4', 'card6']],

          test_identity,

          on='TransactionID',

          how='right')

test_merge.head()

test_merge.isna().sum()

test_merge =
test_merge.drop(columns=['id_03',
'id_04', 'id_07', 'id_08', 'id_09', 'id_10',
'id_14', 'id_18', 'id_21', 'id_22', 'id_23',
'id_24', 'id_25', 'id_26', 'id_27', 'id_30',
'id_32', 'id_33', 'id_34'])

for col in numeric_features:


test_merge[col].fillna(test_merge[col].
median(), inplace=True)



for col in categorical_features:

   test_merge[col].fillna('na',
inplace=True)

test_merge.loc[test_merge['id_31'].str.
contains('samsung', case=False),
'id_31'] = 'samsung'

test_merge.loc[test_merge['id_31'].str.
contains('chrome', case=False)&
test_merge['id_31'].str.contains('andro
id', case=False), 'id_31'] = 'chrome'

test_merge.loc[test_merge['id_31'].str.
contains('chrome', case=False), 'id_31']
= 'chrome'

test_merge.loc[test_merge['id_31'].str.
contains('ie', case=False), 'id_31'] = 'ie'

test_merge.loc[test_merge['id_31'].str.
contains('edge', case=False), 'id_31'] =
'edge'

test_merge.loc[test_merge['id_31'].str.
contains('firefox', case=False), 'id_31']
= 'firefox'

#result3.loc[result3['id_31'].str.contain
s('Android', case=False) |
result3['id_31'].str.contains('android',
case=False), 'id_31'] = 'android'

test_merge.loc[test_merge['id_31'].str.
contains('opera', case=False), 'id_31'] =
'opera'

test_merge.loc[test_merge['id_31'].str.
contains('safari', case=False), 'id_31'] =
'safari'

#for other

test_merge.loc[~test_merge['id_31'].is
in(['samsung','chrome','ie','edge','firefo
x','opera','safari','na']), 'id_31'] =
'other'

test_merge.loc[test_merge['DeviceInfo
'].str.contains('iOS', case=False),
'DeviceInfo'] = 'iOS'

test_merge.loc[test_merge['DeviceInfo
'].str.contains('Trident', case=False),
'DeviceInfo'] = 'Trident'

test_merge.loc[test_merge['DeviceInfo
'].str.contains('Windows', case=False),
'DeviceInfo'] = 'Windows'

test_merge.loc[test_merge['DeviceInfo
'].str.contains('MacOS', case=False),
'DeviceInfo'] = 'MacOS'

#for other

test_merge.loc[~test_merge['DeviceInf
o'].isin(['iOS','Trident','Windows','Mac
OS','na']), 'DeviceInfo'] = 'other'

predict_features =
test_merge.drop(columns =
['TransactionID'])

predict_features

final_prediction =
rfc_full.predict(predict_features)

final_df =
test_merge[['TransactionID']]

final_df['isFraud'] =
final_prediction.tolist()

final_df

final_df.to_csv('C:/RIT Courses/BANA
780 - Advanced Bus. Analytics/Problem
4/IEEE-CIS Fraud Detection submission
- KhizarTahir.csv', index=False)
```