Fase 1 - AA2 - Credit Fraud Detection

Pacotes necessários para a solução

In [32]:

```
import pandas as pd # python's library for data manipulation and preprocessing
import numpy as np  # python's library for number crunching

import matplotlib            # python's library for visualisation
import matplotlib.pyplot as plt

import seaborn as sns        # also python's library for visualisations
color = sns.color_palette()
sns.set_style('darkgrid')

import os

from IPython.display import Image #Package que permite fazer o import de imagens

from random import randint
from scipy.stats import randint as sp_randint

from time import time
from datetime import datetime

import sklearn                #python's machine learning library
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from sklearn.model_selection import train_test_split, StratifiedShuffleSplit, Randor
from sklearn.feature_selection import SelectKBest, VarianceThreshold, chi2

# import warnings filter
from warnings import simplefilter

%matplotlib inline
```

In [33]:

```
# permite ignorar warnings
simplefilter(action='ignore', category=FutureWarning)
```

Carregamento dos dados

O primeiro passo a realizar é a leitura do dataset que se encontra no formato .csv.

In [31]:

```
#train = pd.read_csv("../../datasets/Catch-the-fraudster/small-dataset.csv")
train = pd.read_csv("/Users/vitorpeixoto/Documents/MIEI/AA2/projeto_dataset/train_v2
```

Para verificar o correto carregamento do ficheiro, imprimimos as primeiras 5 linhas e a informação sobre as colunas que compõem este dataset.

In [34]:

```
train.head()
```

Out[34]:

	id	timestamp	product_id	product_department	product_category	card_id	user_id	C
0	0	1413851531856	f3845767	1fbe01fe	28905ebd	ecad2386	a99f214a	3
1	1	1413851817483	f3845767	1fbe01fe	28905ebd	ecad2386	a99f214a	3
2	2	1413852597526	f3845767	1fbe01fe	28905ebd	ecad2386	a99f214a	3
3	3	1413851283020	f3845767	1fbe01fe	28905ebd	ecad2386	a99f214a	3
4	4	1413849935779	9166c161	fe8cc448	0569f928	ecad2386	a99f214a	3

Temos também ao nosso dispor um dataset de dados de teste, para podermos fazer predições sobre a sua classificação.

In [129]:

```
#test = pd.read_csv("../../datasets/Catch-the-fraudster/test_v2.csv")
test = pd.read_csv("/Users/vitorpeixoto/Documents/MIEI/AA2/projeto_dataset/test_v2.c
```

In [130]:

```
test.head()
```

Out[130]:

	id	timestamp	product_id	product_department	product_category	card_id	use
0	32263877	1414540656054	c4e18dd6	85f751fd	50e219e0	92f5800b	a99f
1	32263886	1414540614666	968765cd	6399eda6	f028772b	ecad2386	a99f
2	32263890	1414540692012	7e091613	e151e245	f028772b	ecad2386	a99f
3	32263895	1414540720045	7e091613	e151e245	f028772b	ecad2386	a99f
4	32263896	1414540641750	c4e18dd6	85f751fd	50e219e0	73206397	cc6c

Modelo preditivo de base

Um 'baseline model' tem uma importância significativa na construção de modelos mais complexos. A precisão de um modelo de base ajuda a escolher uma estratégia de modelação para o problema que temos. Se a performance do modelo de base for fraca, temos de avançar para uma estratégia mais complexa que evite os exatos problemas encontrados no modelo de base.

Começamos por dividir os preditores da variável de resposta:

In [131]:

```
x = train.loc[:, train.columns != 'isfraud']
y = train.loc[:, train.columns == 'isfraud']
```

Depois, dividimos o dataset em dados de treino e dados de validação, num rácio de 80/20.

In [132]:

```
train_x, val_x, train_y, val_y = train_test_split(x, y, test_size=.20)
```

In [133]:

```
print(train_x.shape)
print(val_x.shape)
print(train_y.shape)
print(val_y.shape)
```

```
(1294780, 15)
(323696, 15)
(1294780, 1)
(323696, 1)
```

Sendo que algumas das colunas não são numéricas, vamos considerar como preditores deste modelo apenas as colunas numéricas.

In [134]:

```
clf = DecisionTreeClassifier()
clf.fit(X=train_x[['timestamp', 'C15', 'C16', 'C17', 'C18', 'C19', 'C20', 'C21', 'ar
```

Out[134]:

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=
None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=
None,
            splitter='best')
```

Para efetuarmos as predições do modelo construído, iremos usar os dados de validação para depois podermos obter o 'score' de acerto na classificação das instâncias. Esta função retorna uma array de dados de previsão.

In [135]:

```
preds = clf.predict_proba(val_x[['timestamp', 'C15', 'C16', 'C17', 'C18', 'C19', 'C2
preds = [ x[1] for x in preds ]
```

Para classificar o modelo construído, iremos usar o valor AUC em vez da 'accuracy'. O objetivo da AUC é lidar com situações onde temos distribuições mal balanceadas e queremos um 'score' que tenha em conta o overfit para uma determinada classe. Este é o nosso caso, uma vez que a frequência de casos não fraudulentos é inferior ao número de fraudes.

In [136]:

```
metrics.roc_auc_score(val_y, preds)
```

Out[136]:

```
0.587670802969052
```

Vamos agora obter as predições efetuadas sobre o dataset de teste fornecido no Kaggle:

In [137]:

```
test_preds = clf.predict_proba(test[['timestamp', 'C15', 'C16', 'C17', 'C18', 'C19',
```

A seguir é produzido um dataset que contém apenas as colunas 'id' e 'isfraud'. Este dataset pode ser introduzido na competição Kaggle, de modo a testar a sua accuracy.

In [138]:

```
test_preds = [ x[1] for x in test_preds ]
test_preds = pd.concat([test['id'], pd.Series(test_preds)], axis=1)
test_preds.columns = [['id', 'isfraud']]
test_preds.head()
```

Out[138]:

	id	isfraud
0	32263877	0.000000
1	32263886	0.000000
2	32263890	0.166667
3	32263895	0.166667
4	32263896	0.000000

Este ficheiro pode depois ser guardado no formato .csv para inserir na página da competição.

In [139]:

```
#preds.to_csv("baseline_predictions.csv", index=None)
```

Exploratory Data Analysis (EDA)

Como primeiro passo, é feita uma exploração inicial do dataset. Efetuar uma análise exploratória dos dados é de extrema importância antes da construção de qualquer modelo preditivo. O principal motivo prende-se com a importância de perceber que dados temos, que dados não temos, a sua qualidade, determinar a inexistência de atributos, dados duplicados e outros problemas nos dados que podem levar à construção de modelos incorretos e overfitted.

Conforme sejam encontrados problemas neste dataset, estes serão imediatamente corrigidos.

In [35]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32369524 entries, 0 to 32369523
Data columns (total 16 columns):
id                    int64
timestamp             int64
product_id            object
product_department    object
product_category      object
card_id               object
user_id               object
C15                   int64
C16                   int64
C17                   int64
C18                   int64
C19                   int64
C20                   int64
C21                   int64
amount                float64
isfraud               int64
dtypes: float64(1), int64(10), object(5)
memory usage: 3.9+ GB
```

In [36]:

```
train.head()
```

Out[36]:

	id	timestamp	product_id	product_department	product_category	card_id	user_id	C
0	0	1413851531856	f3845767	1fbe01fe	28905ebd	ecad2386	a99f214a	3
1	1	1413851817483	f3845767	1fbe01fe	28905ebd	ecad2386	a99f214a	3
2	2	1413852597526	f3845767	1fbe01fe	28905ebd	ecad2386	a99f214a	3
3	3	1413851283020	f3845767	1fbe01fe	28905ebd	ecad2386	a99f214a	3
4	4	1413849935779	9166c161	fe8cc448	0569f928	ecad2386	a99f214a	3

Preditores não numéricos

Como é possível observar na pequena amostra do dataset que temos em mãos, alguns dos preditores são compostos por dados alfanuméricos. Com efeito, verificamos que tais dados se encontram no formato hexadecimal. Assim, de modo a poder integrar estas variáveis num modelo preditivo, podemos converter estas variáveis para um formato decimal.

In [37]:

```
train['product_id'] = train['product_id'].apply(int, base=16)
train['product_department'] = train['product_department'].apply(int, base=16)
train['product_category'] = train['product_category'].apply(int, base=16)
train['card_id'] = train['card_id'].apply(int, base=16)
train['user_id'] = train['user_id'].apply(int, base=16)
```

In [38]:

```
train.head()
```

Out[38]:

	id	timestamp	product_id	product_department	product_category	card_id	user_
0	0	1413851531856	4085536615	532546046	680550077	3970769798	28457782
1	1	1413851817483	4085536615	532546046	680550077	3970769798	28457782
2	2	1413852597526	4085536615	532546046	680550077	3970769798	28457782
3	3	1413851283020	4085536615	532546046	680550077	3970769798	28457782
4	4	1413849935779	2439430497	4270638152	90831144	3970769798	28457782

Variável de interesse

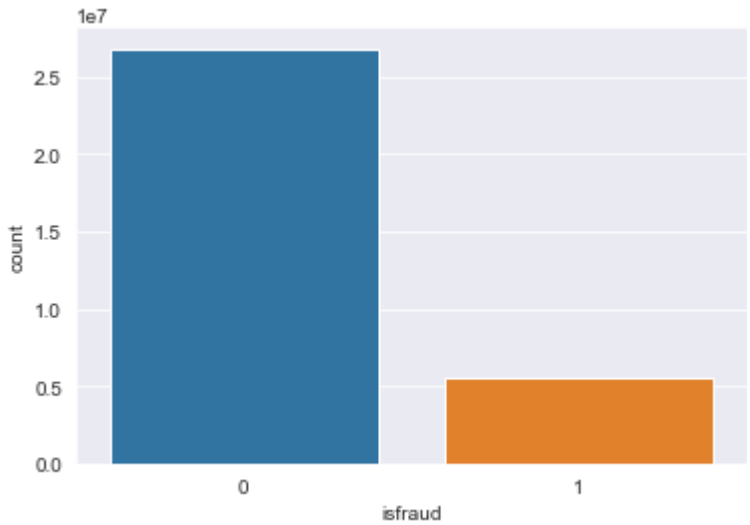
A nossa variável de interesse é designada "isfraud" e diz-nos se a transação foi fraudulenta ou não, sendo portanto, uma variável categórica binária, tratando-se de um problema de classificação.

In [39]:

```
sns.countplot(train['isfraud'])
```

Out[39]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a1e242198>
```



Observando o gráfico de barras acima, verifica-se um evidente desiquilíbrio da variável de interesse. Com efeito, o número de transações não fraudelentas é bastante inferior ao número de fraudes. A percentagem desta distribuição pode ser calculada da seguinte forma:

In [40]:

```
print('Não Fraude corresponde a', round(train['isfraud'].value_counts()[0]/len(trair
print('Fraude corresponde a', round(train['isfraud'].value_counts()[1]/len(train) *
```

```
Não Fraude corresponde a 82.86 % no dataset
Fraude corresponde a 17.14 % no dataset
```

Este desiquilíbrio pode trazer problemas na construção do modelo, uma vez que este pode ser overfitted para casos de não fraude. Desta forma, temos de resolver este desiquilíbrio. Várias técnicas estão ao nosso dispor. Oversampling é uma das técnicas e consiste em criar novas amostras da classe com menor frequência. Undersampling é também uma técnica, mas elimina amostras da classe mais frequente, de modo a igualar o número de amostras de ambas as classes.

Dado o grande tamanho deste dataset, temos a liberdade para poder eliminar amostras do nosso dataset, pelo que iremos recorrer então à técnica de undersampling.

O primeiro passo é baralhar o dataset para o undersampling ser completamente aleatório.

In [41]:

```
train = train.sample(frac=1)
```

De seguida truncamos os casos 'não fraude' ao número de casos de fraude.

In [42]:

```
fraud_train = train.loc[train['isfraud'] == 1]
non_fraud_train = train.loc[train['isfraud'] == 0][:(len(fraud_train))]

print(len(fraud_train))
print(len(non_fraud_train))
```

```
5549697
5549697
```

De seguida concatenamos os dois datasets de volta ao dataset de treino e observámos a nova distribuição.

In [43]:

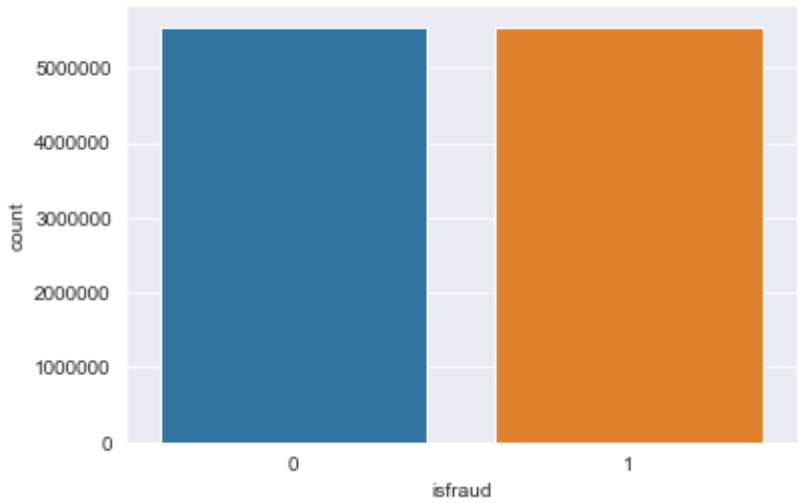
```
train = pd.concat([fraud_train, non_fraud_train])
```

In [44]:

```
sns.countplot(train['isfraud'])
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1b7df3a390>
```



Por fim, voltámos a baralhar o dataset de modo a obtermos os casos de fraude e não fraude misturados.

In [45]:

```
train = train.sample(frac=1, random_state=42)
```

Missing Data

É de extrema importância a verificação da existência de valores em falta ou corruptos no nosso conjunto de dados.

In [46]:

```
train.isnull().values.any()
```

Out[46]:

```
False
```

No entanto, tal como se pode visualizar numa pequena amostra de dados, a coluna C20 contém linhas com o valor -1, o que mostra, quando comparado com as outras linhas, que é um valor em falta.

Vamos verificar:

In [47]:

```
print('O valor "-1" corresponde a', round(((train['C20'] == -1).sum())/len(train) *
print('Os restantes', round(((train['C20'] > 1000).sum())/len(train) * 100,2), '% do
```

```
O valor "-1" corresponde a 48.99 % dos dados da coluna C20.
Os restantes 51.01 % dos dados encontram-se com valores acima de 1000.
```

De facto, acaba por ser uma percentagem significativa de missing data. Para melhorar a compreensão deste dataset, iremos substituir o valor -1 por um atributo NaN.

In [48]:

```
train[['C20']] = train[['C20']].replace(-1, np.NaN)
print(train.isnull().sum())
```

```
id                          0
timestamp                   0
product_id                  0
product_department          0
product_category            0
card_id                     0
user_id                     0
C15                         0
C16                         0
C17                         0
C18                         0
C19                         0
C20                   5437219
C21                         0
amount                      0
isfraud                     0
dtype: int64
```

In [49]:

```
train.head()
```

Out[49]:

	id	timestamp	product_id	product_department	product_category	car
18952840	18952840	1414254982885	2114524691	3780239941	4029183787	397076
15124168	15124168	1414153170508	4085536615	532546046	680550077	397076
1729090	1729090	1413880555205	2114524691	3780239941	4029183787	397076
22420515	22420515	1414332300856	1988601966	1527301435	1048658224	397076
27344708	27344708	1414463795086	1359730745	175384852	4029183787	397076

In [51]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 11099394 entries, 18952840 to 25190857
Data columns (total 16 columns):
id                    int64
timestamp             int64
product_id            int64
product_department    int64
product_category      int64
card_id               int64
user_id               int64
C15                   int64
C16                   int64
C17                   int64
C18                   int64
C19                   int64
C20                   float64
C21                   int64
amount                float64
isfraud               int64
dtypes: float64(2), int64(14)
memory usage: 1.4 GB
```

In [52]:

```
train = train[np.isfinite(train['C20'])]
```

In [53]:

```
train.head()
```

Out[53]:

	id	timestamp	product_id	product_department	product_category	car
18952840	18952840	1414254982885	2114524691	3780239941	4029183787	397076
1729090	1729090	1413880555205	2114524691	3780239941	4029183787	397076
22420515	22420515	1414332300856	1988601966	1527301435	1048658224	397076
27344708	27344708	1414463795086	1359730745	175384852	4029183787	397076
29479409	29479409	1414498422798	3303116246	2247578109	1356995040	193150

In [54]:

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5662175 entries, 18952840 to 25190857
Data columns (total 16 columns):
id                    int64
timestamp             int64
product_id            int64
product_department    int64
product_category      int64
card_id               int64
user_id               int64
C15                   int64
C16                   int64
C17                   int64
C18                   int64
C19                   int64
C20                   float64
C21                   int64
amount                float64
isfraud               int64
dtypes: float64(2), int64(14)
memory usage: 734.4 MB
```

In [62]:

```
(train['user_id'] == 2845778250).sum()/len(train)
```

Out[62]:

```
0.8427208625660634
```

In [63]:

```
train_sample = train.sample(3000000, replace=True)
```

In [64]:

```
train_sample.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3000000 entries, 21151950 to 6620369
Data columns (total 16 columns):
id                    int64
timestamp             int64
product_id            int64
product_department    int64
product_category      int64
card_id               int64
user_id               int64
C15                   int64
C16                   int64
C17                   int64
C18                   int64
C19                   int64
C20                   float64
C21                   int64
amount                float64
isfraud               int64
dtypes: float64(2), int64(14)
memory usage: 389.1 MB
```

In [65]:

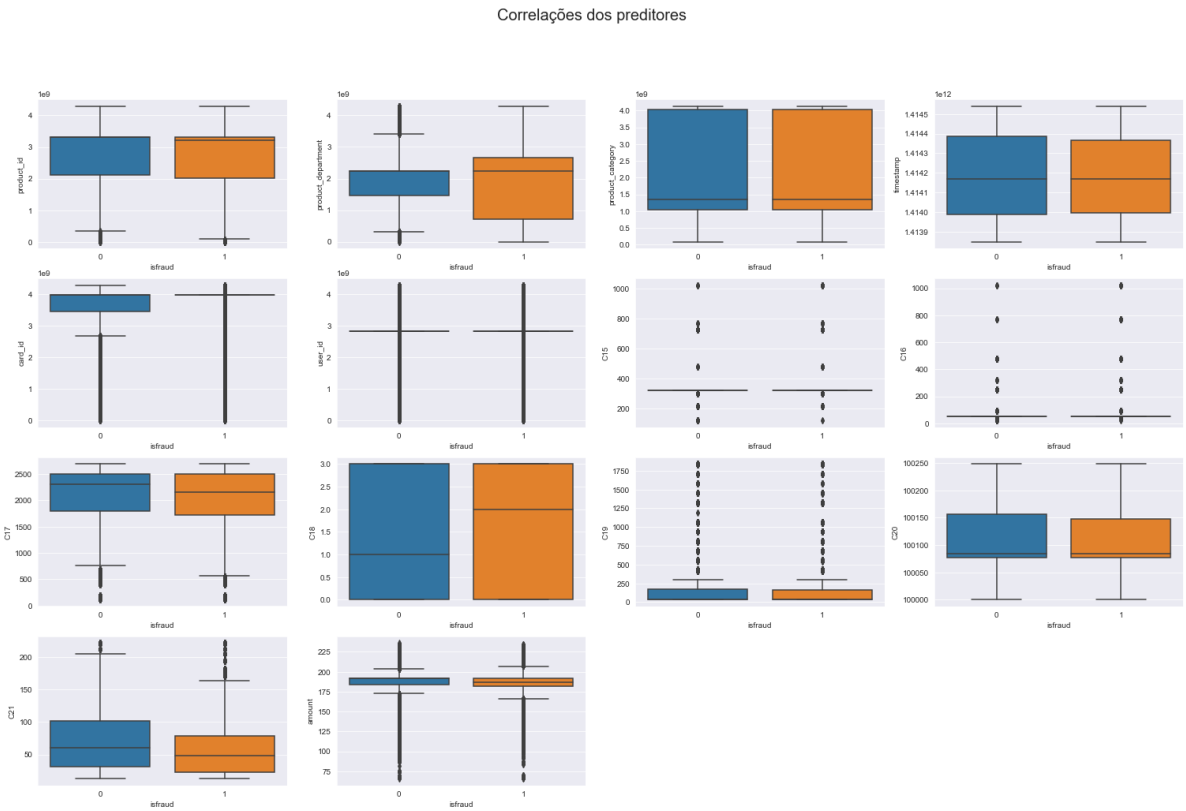
```
train_sample.to_csv("aa2_sample.csv", index=None)
```

Amostra gráfica da distribuição dos valores dos preditores

Através dos gráficos a seguir mostrados podemos observar a distribuição das variáveis do nosso dataset, encontrando padrões que permitam encontrar erros no dataset ou outliers.

In [258]:

```
f, axes = plt.subplots(nrows=4, ncols=4, figsize=(26,16))
f.suptitle('Correlações dos preditores', size=20)
sns.boxplot(x="isfraud", y="product_id", data=train, ax=axes[0,0])
sns.boxplot(x="isfraud", y="product_department", data=train, ax=axes[0,1])
sns.boxplot(x="isfraud", y="product_category", data=train, ax=axes[0,2])
sns.boxplot(x="isfraud", y="timestamp", data=train, ax=axes[0,3])
sns.boxplot(x="isfraud", y="card_id", data=train, ax=axes[1,0])
sns.boxplot(x="isfraud", y="user_id", data=train, ax=axes[1,1])
sns.boxplot(x="isfraud", y="C15", data=train, ax=axes[1,2])
sns.boxplot(x="isfraud", y="C16", data=train, ax=axes[1,3])
sns.boxplot(x="isfraud", y="C17", data=train, ax=axes[2,0])
sns.boxplot(x="isfraud", y="C18", data=train, ax=axes[2,1])
sns.boxplot(x="isfraud", y="C19", data=train, ax=axes[2,2])
sns.boxplot(x="isfraud", y="C20", data=train, ax=axes[2,3])
sns.boxplot(x="isfraud", y="C21", data=train, ax=axes[3,0])
sns.boxplot(x="isfraud", y="amount", data=train, ax=axes[3,1])

f.delaxes(axes[3,2])
f.delaxes(axes[3,3])
```



Abaixo temos a mesma distribuição, mas agora de acordo com a variável resultado 'isfraud'. Estes gráficos ajudam a encontrar a existência de maiores frequências de um determinado valor para casos fraudulentos, podendo ter efeito na correlação entre esse preditor e a variável resultado.

In [259]:

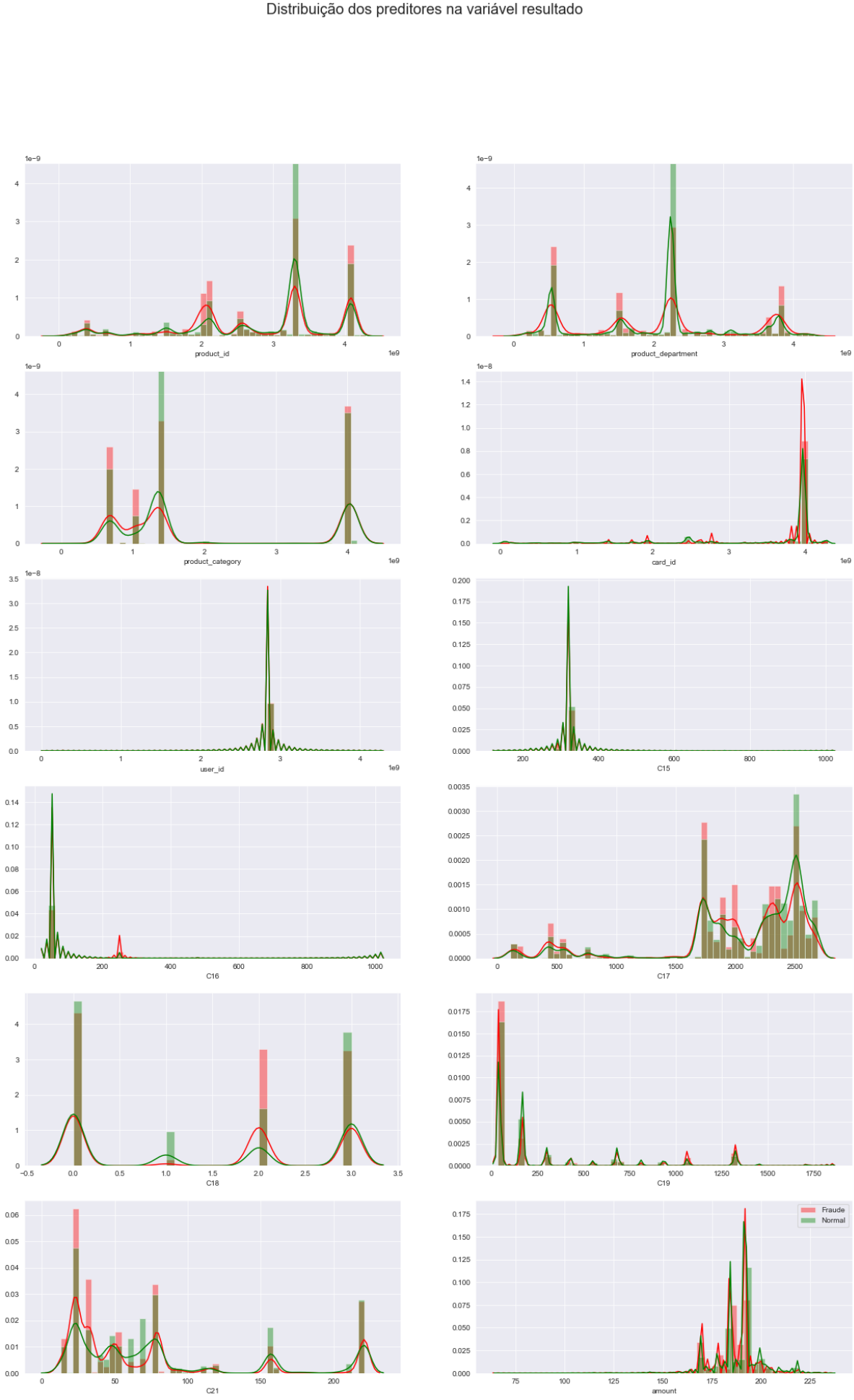
```
fraud_data = train[train["isfraud"]==1]
normal_data = train[train["isfraud"]==0]
```

In [264]:

```
f, axes = plt.subplots(nrows=6, ncols=2, figsize=(20,30))
f.suptitle('Distribuição dos preditores na variável resultado', size=20)

sns.distplot( fraud_data["product_id"] , color="red", label="Fraude", ax=axes[0,0])
sns.distplot( normal_data["product_id"] , color="green", label="Normal", ax=axes[0,0

sns.distplot( fraud_data["product_department"] , color="red", label="Fraude", ax=axe
sns.distplot( normal_data["product_department"] , color="green", label="Normal", ax=

sns.distplot( fraud_data["product_category"] , color="red", label="Fraude", ax=axes[
sns.distplot( normal_data["product_category"] , color="green", label="Normal", ax=ax

sns.distplot( fraud_data["card_id"] , color="red", label="Fraude", ax=axes[1,1])
sns.distplot( normal_data["card_id"] , color="green", label="Normal", ax=axes[1,1])

sns.distplot( fraud_data["user_id"] , color="red", label="Fraude", ax=axes[2,0])
sns.distplot( normal_data["user_id"] , color="green", label="Normal", ax=axes[2,0])

sns.distplot( fraud_data["C15"] , color="red", label="Fraude", ax=axes[2,1])
sns.distplot( normal_data["C15"] , color="green", label="Normal", ax=axes[2,1])

sns.distplot( fraud_data["C16"] , color="red", label="Fraude", ax=axes[3,0])
sns.distplot( normal_data["C16"] , color="green", label="Normal", ax=axes[3,0])

sns.distplot( fraud_data["C17"] , color="red", label="Fraude", ax=axes[3,1])
sns.distplot( normal_data["C17"] , color="green", label="Normal", ax=axes[3,1])

sns.distplot( fraud_data["C18"] , color="red", label="Fraude", ax=axes[4,0])
sns.distplot( normal_data["C18"] , color="green", label="Normal", ax=axes[4,0])

sns.distplot( fraud_data["C19"] , color="red", label="Fraude", ax=axes[4,1])
sns.distplot( normal_data["C19"] , color="green", label="Normal", ax=axes[4,1])

sns.distplot( fraud_data["C21"] , color="red", label="Fraude", ax=axes[5,0])
sns.distplot( normal_data["C21"] , color="green", label="Normal", ax=axes[5,0])

sns.distplot( fraud_data["amount"] , color="red", label="Fraude", ax=axes[5,1])
sns.distplot( normal_data["amount"] , color="green", label="Normal", ax=axes[5,1])

plt.legend()
```

Out[264]:

```
<matplotlib.legend.Legend at 0x1c04914278>
```



Correlações

As correlações entre os preditores e a variável resultado é uma medida de grande utilidade. Estes valores permitem visualizar o relacionamento entre um preditor e o outcome, neste caso. Se o módulo da correlação entre um preditor e o resultado for elevado, então o preditor pode ser considerado de forte, exercendo grande influência no resultado. Assim torna-se uma variável de extrema importância na construção de um modelo preditivo.

Vamos analisar as correlações existentes entre os nossos preditores e a variável resultado:

In [272]:

```
corr = train.corr()
corr[['isfraud']]
```

Out[272]:

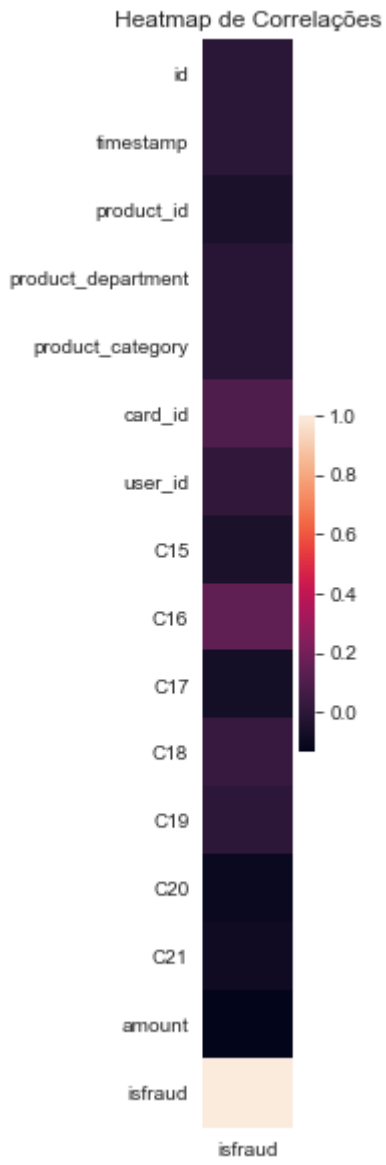
	isfraud
id	-0.006985
timestamp	-0.007003
product_id	-0.050979
product_department	-0.012610
product_category	-0.009745
card_id	0.097791
user_id	0.013748
C15	-0.049246
C16	0.147543
C17	-0.074333
C18	0.032712
C19	0.004333
C20	-0.101557
C21	-0.087337
amount	-0.132399
isfraud	1.000000

In [273]:

```
plt.figure(figsize=(1,10))
heat = sns.heatmap(data=corr[['isfraud']])
plt.title('Heatmap de Correlações')
```

Out[273]:

```
Text(0.5, 1.0, 'Heatmap de Correlações')
```



Como podemos observar, encontrámos duas variáveis com correlação positiva significativa relativamente à variável resultado, sendo elas C16 (0.1475) e card_id (0.0978). Temos ainda C20 (-0.1016) e amount (-0.1324) com correlação negativa relativamente à variável resultado. Estes 4 preditores apresentam correlações com maior magnitude relativamente às restantes, pelo que podemos considerar que terão mais efeito na previsão da classe do outcome.

Este foi o nosso primeiro modelo que agora vamos tentar melhorar utilizando tecnicas avançadas de Machine Learning

NOTA: Daqui para cima está tudo organizado!!!

Em seguida iremos realizar os seguintes passos:

```
- Tratamento do Imbalanced Data
- EDA - Exploração dos dados
- Seleção de atributos
    - Remover atributos altamente correlacionados
- Divisão da amostra para aprendizagem máquina
- Testar dados com feature selection
- Otimização de hiperparâmetros
- Feature engineering -
- Por fim produzir um modelo final com todos os dados

Agregação dos dados ???
Redução de    dimensionalidade ???
Feature Scaling
#Se não fizermos, teremos de saber explicar porque não foi realizado
```

Modelos que iremos utilizar para melhorar as nossas previsões:

```
- SVM           - Support Vector Machines
- Random Forest - or random decision forests are an ensemble learning method
for classification, regression and other tasks that operates by constructing
a multitude of decision trees at training time and outputting the class that
is the mode of the classes (classification) or mean prediction (regression)
 of the individual trees.
- XGBoost       - XGBoost, short for “Extreme Gradient Boosting”, was introd
uced by Chen in 2014. Since its introduction, XGBoost has become one of the
 most popular machine learning algorithm.
```

Feature Selection

```
Feature Engineering é o processo de seleção e transformação de dados para melhorar
a precisão dos
algoritmos de Machine Learning.

Este processo requer profunda familiaridade com os dados e é cosensualmente
considerado
um dos processos mais importantes para a obtençao de resltados de qualidade.

Dentro desta temática iremos realizar a seguinte análise
 - remoção das colunas que card_id, user_id e product_id
 - Realizar o one-hot encodin na variáveis categóricas
 - transformar o timestamp em colunas com (Ano, mês, dia, hora, minuto e segundo
```

In [46]:

```
train = train[['timestamp','product_id','product_department','product_category','ca
train.head()
```

Out[46]:

	timestamp	product_id	product_department	product_category	card_id	us
1474846	1413855371103	4085536615	532546046	680550077	3970769798	28457
1074157	1414298239714	1988601966	1817921580	1048658224	3970769798	28457
839433	1413886770474	3303116246	2247578109	1356995040	1059744955	28457
930101	1413872679436	2555849849	3648327399	4029183787	3970769798	28457
235928	1413973625470	3303116246	2247578109	1356995040	3915871216	28457

In [47]:

```
print(train.shape)
```

```
(554970, 14)
```

Testes estatísticos univariados

In [48]:

```
x = train.iloc[:,0:13]  #independent columns
y = train.iloc[:,-1]    #target column i.e price range
```

In [49]:

```
x.head()
```

Out[49]:

	timestamp	product_id	product_department	product_category	card_id	us
1474846	1413855371103	4085536615	532546046	680550077	3970769798	28457
1074157	1414298239714	1988601966	1817921580	1048658224	3970769798	28457
839433	1413886770474	3303116246	2247578109	1356995040	1059744955	28457
930101	1413872679436	2555849849	3648327399	4029183787	3970769798	28457
235928	1413973625470	3303116246	2247578109	1356995040	3915871216	28457

In [50]:

```
y.head()
```

Out[50]:

```
1474846    0
1074157    1
839433     0
930101     1
235928     0
Name: isfraud, dtype: int64
```

In [51]:

```
#apply SelectKBest class to extract top 8 best features
bestfeatures = SelectKBest(score_func=chi2, k=13)
fit = bestfeatures.fit(x,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(x.columns)
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns,dfscores],axis=1)
featureScores.columns = ['Specs','Score']  #naming the dataframe columns
print(featureScores.nlargest(13,'Score'))  #print 8 best features
```

```
                 Specs         Score
4              card_id  1.375943e+12
1           product_id  5.504553e+11
2   product_department  5.639717e+10
3     product_category  3.728974e+10
5              user_id  1.155424e+10
0            timestamp  1.123486e+06
7                  C16  6.007076e+05
8                  C17  5.615803e+05
11                 C21  2.353640e+05
12              amount  7.680067e+03
10                 C19  5.500543e+03
6                  C15  1.680880e+03
9                  C18  6.618732e+02
```

Extra Tree Classifier

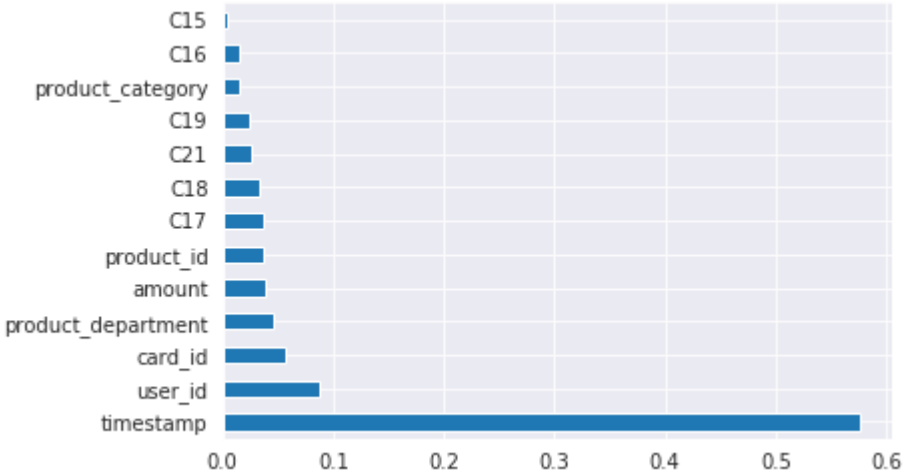
In [52]:

```
model = ExtraTreesClassifier()
model.fit(x,y)
print(model.feature_importances_) #use inbuilt class feature_importances of tree bas
```

```
[0.5769315  0.03692125 0.04537633 0.01570893 0.05670623 0.08779631
 0.00519852 0.01523285 0.03672507 0.03332343 0.02382608 0.02677848
 0.03947502]
```

In [53]:

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=x.columns)
feat_importances.nlargest(13).plot(kind='barh')
plt.show()
```



Vamos fazer agora a divisão estratificada da Amostra

```
Amostragem Estratificada: As amostras são desenhadas dentro de categorias pré-
especificadas (ou seja, estratos).
O erro de amostragem é geralmente menor na amostragem estratificada do que na
amostragem aleatória.

Estamos a fazer uma divisão de 80% para dados de treino e 20% para dados de teste.
```

In [54]:

```
# Nas linhas abaixo implementamos a tecnica

split = StratifiedShuffleSplit(n_splits=1, test_size=0.8, random_state=42)

for test_index, train_index in split.split(train, train["isfraud"]):
    test = train.iloc[test_index]
    train = train.iloc[train_index]
```

In [55]:

```
# tamanho do dataset de treino
train.shape
```

Out[55]:

```
(443976, 14)
```

In [56]:

```
# tamanho do dataset de validação
test.shape
```

Out[56]:

```
(110994, 14)
```

In [57]:

```
train.head()
```

Out[57]:

	timestamp	product_id	product_department	product_category	card_id	us
276424	1413850495172	4085536615	532546046	680550077	3970769798	28457
430341	1414134912085	1988601966	1817921580	1048658224	3970769798	28457
1201356	1414237556871	3303116246	2247578109	1356995040	2465562635	28457
580275	1414328614652	2114524691	3780239941	4029183787	3970769798	28457
519473	1414003534328	3303116246	2247578109	1356995040	2480375494	28457

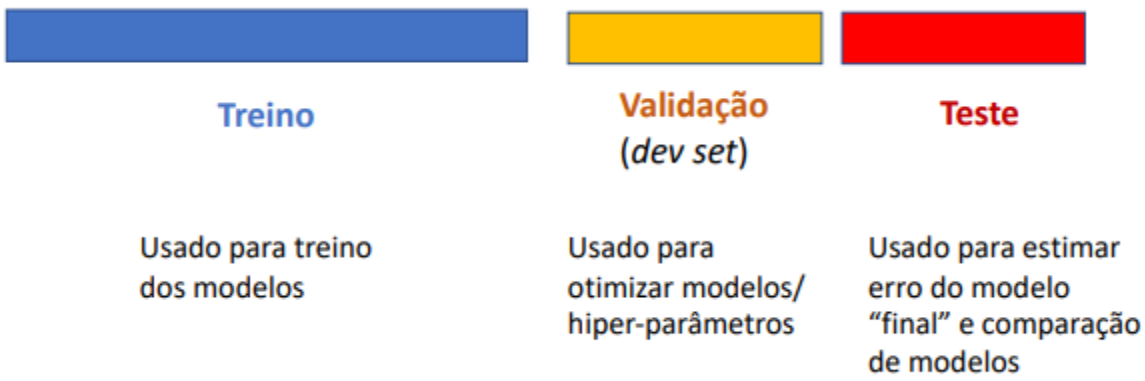
In [58]:

```
#Neste momento estamos com a distribuição apresentada na figura abaixo.
    # Iremos treinar o nosso modelo nos dados de Treino
    # Faremos as melhorias analisando as previsões sobre os dados de validação (dev
    # Por fim aplicaremos o nosso modelo , somente uma vez, nos dados de teste.
```

In [59]:

```
Image("img/avaliacao-do-modelo.png")
```

Out[59]:



In [60]:

```
train["isfraud"].value_counts() / len(train)
```

Out[60]:

```
1    0.5
0    0.5
Name: isfraud, dtype: float64
```

In [61]:

```
# Quantidade de registos na nossa amostra de treino
train.shape
```

Out[61]:

```
(443976, 14)
```

In [62]:

```
# Quantidade de registos na nossa amostra de treino
test.shape
```

Out[62]:

```
(110994, 14)
```

Feature Engineering

```
Feature Engineering é o processo de seleção e transformação de dados para melhorar
a precisão dos
algoritmos de Machine Learning.

Este processo requer profunda familiaridade com os dados e é cosensualmente
considerado
um dos processos mais importantes para a obtençao de resltados de qualidade.

Dentro desta temática iremos realizar a seguinte análise
 - remoção das colunas que card_id, user_id e product_id
 - Realizar o one-hot encodin na variáveis categóricas
 - transformar o timestamp em colunas com (Ano, mês, dia, hora, minuto e segundo
```

In [63]:

```
# Define a function splitDate
def addDates(dateValue):
    epoch_time = float (str(dateValue)[0: 10])
    date=datetime.utcfromtimestamp(epoch_time)
    return date.year,date.month,date.day,date.hour,date.minute,date.second
```

In [64]:

```
train['year'], train['month'], train['day'], train['hour'], train['minute'], train[
```

In [65]:

```
# Remove the timestamp column
train.drop(columns=['timestamp'], axis=1, inplace=True)
```

In [66]:

```
train.head()
```

Out[66]:

	product_id	product_department	product_category	card_id	user_id	C15	C16
276424	4085536615	532546046	680550077	3970769798	2845778250	320	50
430341	1988601966	1817921580	1048658224	3970769798	2845778250	300	250
1201356	3303116246	2247578109	1356995040	2465562635	2845778250	320	50
580275	2114524691	3780239941	4029183787	3970769798	2845778250	320	50
519473	3303116246	2247578109	1356995040	2480375494	2845778250	320	480

Modelação - Random Forest Classifier

In [67]:

```
# Manuel

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
```

In [68]:

```
# Choose the model
random_forest = RandomForestClassifier()
```

In [69]:

```
# Create a DataFrame with the features (X) and labels (y)
x = train.drop(["isfraud","user_id","hour","product_id","product_category","card_id"
y = train["isfraud"]
```

In [70]:

```
x.head()
```

Out[70]:

	product_department	C17	C21
276424	532546046	1722	79
430341	1817921580	2162	33
1201356	2247578109	2424	71
580275	3780239941	2340	159
519473	2247578109	2502	221

In [71]:

```
y.head()
```

Out[71]:

```
276424     0
430341     1
1201356    0
580275     0
519473     0
Name: isfraud, dtype: int64
```

Cross Validation for random forest

In [72]:

```
from sklearn.model_selection import cross_val_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedKFold
```

In [73]:

```
# Temporariamente retiro o C20
# train.drop(columns=['C20'], axis=1, inplace=True) JA FOI REMOVIDO NO FEATURE SELEC
```

In [74]:

```
# Fit the model
# Make the predictions
# Return and Array of scores of the estimator for each run of the cross validation.


#scores = cross_val_score(random_forest, train, y, scoring='roc_auc', cv=10)
####### OU #######
scores = cross_val_score(random_forest, x, y, scoring='roc_auc', cv=10)


#scoring='roc_auc'
```

In [75]:

```
print("Mean of scores: {:.3f}".format(scores.mean()))
print("Mean of scores: {:.3f}".format(scores.var()))
```

```
Mean of scores: 0.716
Mean of scores: 0.000
```

In [76]:

```
#cross_val_score trains models on inputs with true values, performs predictions, the
#predictions to the true values—the scoring step. That's why you pass in y: it's the
#the "ground truth".

#The roc_auc_score function that is called by specifying scoring='roc_auc' relies on
#the ground truth and the predicted values based on X for your model.
```

Modelação - Support Vector Machine

In [77]:

```
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score

# Choose the model
support_vector_machines = SVC(kernel='linear')
```

In [78]:

```
# Fit the model
# Make the predictions
# Return and Array of scores of the estimator for each run of the cross validation.

#scores = cross_val_score(support_vector_machines, train, y, scoring='roc_auc', cv=
####### OU #######
#scores = cross_val_score(support_vector_machines, x, y, scoring='roc_auc', cv=10)
```

In [79]:

```
#print("Mean of scores: {:.3f}".format(scores.mean()))
#print("Mean of scores: {:.3f}".format(scores.var()))
```

Modelação - XGBoost: Extreme Gradient Boosting

In [80]:

```
# XGBoost, short for “Extreme Gradient Boosting”, was introduced by Chen in 2014.
# Since its introduction, XGBoost has become one of the most popular machine learni
```

In [81]:

```
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier
```

In [82]:

```
model_xgboost = XGBClassifier()
```

In [83]:

```
# Fit the model
# Make the predictions
# Return and Array of scores of the estimator for each run of the cross validation.
scores = cross_val_score(model_xgboost, x, y,scoring='roc_auc', cv=10)
```

In [84]:

```
print("Mean of scores: {:.3f}".format(scores.mean()))
print("Mean of scores: {:.3f}".format(scores.var()))
```

```
Mean of scores: 0.693
Mean of scores: 0.000
```

Hyper Parameter Tuning - Random Forest

In [85]:

```
# Vamos fazer uma pesquisa dos hyperparameters de uma forma automática
# Os métodos que conhecemos para realizar esta tarefa são:
        #Grid Search
        #Randomized search
```

In [86]:

```
# função que permite fazer um  to report best scores
def report(results, n_top=3):
    for i in range(1, n_top + 1):
        candidates = np.flatnonzero(results['rank_test_score'] == i)
        for candidate in candidates:
            print("Model with rank: {0}".format(i))
            print("Mean validation score: {0:.3f} (std: {1:.3f})".format(
                  results['mean_test_score'][candidate],
                  results['std_test_score'][candidate]))
            print("Parameters: {0}".format(results['params'][candidate]))
            print("")
```

In [87]:

```
# Em seguida imprimimos os hyperparameters usados por defeito pelo modelo random fo

from pprint import pprint
print('Parameters currently in use:\n')
pprint(random_forest.get_params())
```

```
Parameters currently in use:

{'bootstrap': True,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'auto',
 'max_leaf_nodes': None,
 'min_impurity_decrease': 0.0,
 'min_impurity_split': None,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 'warn',
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

Explicação dos hyperparameters que vamos modificar

```
    n_estimators = número de árvores no foreset
    max_features = número máximo de recursos considerados para dividir um nó
    max_depth = número máximo de níveis em cada árvore de decisão
    min_samples_split = min número de pontos de dados colocados em um nó antes q
    ue o nó seja dividido
    min_samples_leaf = min número de pontos de dados permitidos em um nó de folh
    a
    bootstrap = método para amostragem de pontos de dados (com ou sem substituiç
    ão)
```

In [89]:

```
# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 200, stop = 2000, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(10, 110, num = 11)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
pprint(random_grid)
```

```
{'bootstrap': [True, False],
 'max_depth': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, None],
 'max_features': ['auto', 'sqrt'],
 'min_samples_leaf': [1, 2, 4],
 'min_samples_split': [2, 5, 10],
 'n_estimators': [200, 400, 600, 800, 1000, 1200, 1400, 1600, 1800, 20
00]}
```

In [ ]:

```
# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations, and use all available cores
# Mudar para 100 que é o aconselhavel
rf_random = RandomizedSearchCV(estimator = random_forest,
                               param_distributions = random_grid, n_iter = 10, cv =
                               verbose=2, scoring='roc_auc', random_state=42, n_jobs
# Fit the random search model
rf_random.fit(x, y)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

[Parallel(n_jobs=6)]: Using backend LokyBackend with 6 concurrent work
ers.
```

In [ ]:

```
#We can view the best parameters from fitting the random search:

rf_random.best_params_
```

Avaliar Random Search

In [7]:

```
# Para determinar se a pesquisa aleatória gerou um modelo melhor, comparamos o model
```

In [ ]:

```
def evaluate(model, test_features, test_labels):
    predictions = model.predict(test_features)
    errors = abs(predictions - test_labels)
    mape = 100 * np.mean(errors / test_labels)
    accuracy = 100 - mape
    print('Model Performance')
    print('Average Error: {:0.4f} degrees.'.format(np.mean(errors)))
    print('Accuracy = {:0.2f}%.'.format(accuracy))

    return accuracy
```

In [ ]:

```
base_model = RandomForestRegressor(n_estimators = 10, random_state = 42)
base_model.fit(train_features, train_labels)
base_accuracy = evaluate(base_model, test_features, test_labels)

best_random = rf_random.best_estimator_
random_accuracy = evaluate(best_random, test_features, test_labels)

print('Improvement of {:0.2f}%.'.format( 100 * (random_accuracy - base_accuracy) / b
```

Hyper Parameter Tuning - Support Vector Machines

In [ ]:

```
from sklearn.model_selection import GridSearchCV
from sklearn import svm

parameters = {'C':[1, 10, 100], 'gamma':[0.01, 0.001]}
svm_model_d = svm.SVC()
opt_model_d = GridSearchCV(svm_model_d, parameters)
opt_model_d.fit(train_x, train_y)
print (opt_model_d.best_estimator_)
opt_model_d.score(val_x, val_y)
```

In [ ]:

```
# Hyper Parameter Tuning -  XGBoost
```