Data Sampling

In [1]:
```
# Data preprocessing
import pandas as pd
```

read_csv() function allow excel(csv format) to be readable.

In [2]:
```
# read train_set
train = pd.read_csv("D:/Project/fraudTrain.csv")
```

In [3]:
```
print(train.shape)
```

```
(1296675, 23)
```

In [4]:
```
# read test_set
test = pd.read_csv("D:/Project/fraudTest.csv")
```

In [5]:
```
print(test.shape)
```

```
(555719, 23)
```

sample(): It allows a subset of defined numbers of data to be selected randomly.

random_state: It allows the sample data selected to be reproduceable.

In [6]:
```
# Select 10000 random sample from trainset
sample_train = train.sample(10000, random_state =42)
```

reset_index()- reset_index function reset the unordered index number

In [7]:
```
# reset the randomized sample trainset data index to avoid error
sample_train = sample_train.reset_index(drop=True)
```

In [8]:
```
# Select 10000 random sample from testset
sample_test = test.sample(10000, random_state =42)
```

In [9]:
```
# reset the randomized sample testset data index to avoid error
sample_test = sample_test.reset_index(drop=True)
```

In [10]:
```
# print trainset and testset shape(rows and columns)
print(sample_train.shape, sample_test.shape)
```

```
(10000, 23) (10000, 23)
```

In [11]:
```
sample_train['is_fraud'].value_counts(normalize=True)
```

Out[11]:
```
0    0.9941
1    0.0059
Name: is_fraud, dtype: float64
```

In [12]:
```
train['is_fraud'].value_counts(normalize=True).round(4)
```

Out[12]:
```
0    0.9942
1    0.0058
Name: is_fraud, dtype: float64
```

to_csv - to_csv function allow a data to be saved as a csv file.

index = False - It disallow the to_csv function not to create an index as a new column.

In [13]:
```
# save randomized sample of trainset as an excel file(csv format)
sample_train.to_csv('fraud_train_sample.csv',index=False)
```

Modelling

In [14]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import train_test_split

# Plotly
import plotly.express as px
import plotly.graph_objects as go

# algorithm to learn from the input and predict the output
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier

# evaluation metrics 1 for classification(know the accuracy score of the model)
from sklearn.metrics import accuracy_score, roc_auc_score,classification_report

# evaluation metrics 2 for classification(know the variation of the real value to p
from sklearn.metrics import confusion_matrix

# save a trained model to disc
import pickle

#datetime library
import datetime
```

In [15]:
```
data = pd.read_csv("fraud_train_sample.csv")
```

In [16]:
```
data.head()
```

Out[16]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt
0	1045211	2020-03-09 15:09:26	577588686219	fraud_Towne LLC	misc_pos	194.51
1	547406	2019-08-22 15:49:01	30376238035123	fraud_Friesen Ltd	health_fitness	52.32
2	110142	2019-03-04 01:34:16	4658490815480264	fraud_Mohr Inc	shopping_pos	6.53
3	1285953	2020-06-16 20:04:38	3514897282719543	fraud_Gaylord-Powlowski	home	7.33
4	271705	2019-05-14 05:54:48	6011381817520024	fraud_Christiansen, Goyette and Schamberger	gas_transport	64.29

5 rows × 23 columns

In [17]:
```
data.shape
```

Out[17]:
```
(10000, 23)
```

In [18]:
```
data.isna().sum()
```

Out[18]:
```
Unnamed: 0               0
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

In [19]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 23 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   Unnamed: 0             10000 non-null  int64  
 1   trans_date_trans_time  10000 non-null  object 
 2   cc_num                 10000 non-null  int64  
 3   merchant               10000 non-null  object 
 4   category               10000 non-null  object 
 5   amt                    10000 non-null  float64
 6   first                  10000 non-null  object 
 7   last                   10000 non-null  object 
 8   gender                 10000 non-null  object 
 9   street                 10000 non-null  object 
 10  city                   10000 non-null  object 
 11  state                  10000 non-null  object 
 12  zip                    10000 non-null  int64  
 13  lat                    10000 non-null  float64
 14  long                   10000 non-null  float64
 15  city_pop               10000 non-null  int64  
 16  job                    10000 non-null  object 
 17  dob                    10000 non-null  object 
 18  trans_num              10000 non-null  object 
 19  unix_time              10000 non-null  int64  
 20  merch_lat              10000 non-null  float64
 21  merch_long             10000 non-null  float64
 22  is_fraud               10000 non-null  int64  
dtypes: float64(5), int64(6), object(12)
memory usage: 1.8+ MB
```

In [20]:

```
data.describe()
```

Out[20]:

	Unnamed: 0	cc_num	amt	zip	lat	long	
count	1.000000e+04	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000	1.000
mean	6.538994e+05	4.138124e+17	69.172016	48876.656100	38.485600	-90.289713	8.830
std	3.762489e+05	1.303913e+18	135.444585	26904.265948	5.054745	13.763297	3.102
min	4.720000e+02	6.041621e+10	1.010000	1257.000000	20.027100	-165.672300	2.300
25%	3.231138e+05	1.800143e+14	9.580000	26237.000000	34.585825	-96.809400	7.430
50%	6.548670e+05	3.517671e+15	47.765000	48174.000000	39.283000	-87.591700	2.401
75%	9.809185e+05	4.629452e+15	83.007500	72047.000000	41.894800	-80.158000	2.047
max	1.296652e+06	4.992346e+18	4430.830000	99783.000000	65.689900	-67.950300	2.906

Exploratory Data Analysis

Univariate Analysis

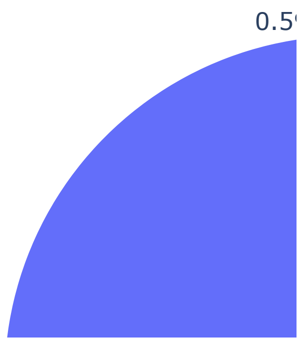
Question1

What is the proportion of the fraud detection?

In [21]:

```
labels = list(data['is_fraud'].value_counts().index)
values = list(data['is_fraud'].value_counts().values)
```

```
fig = go.Figure(data=[go.Pie(labels=labels, values=values)])
fig.show()
```



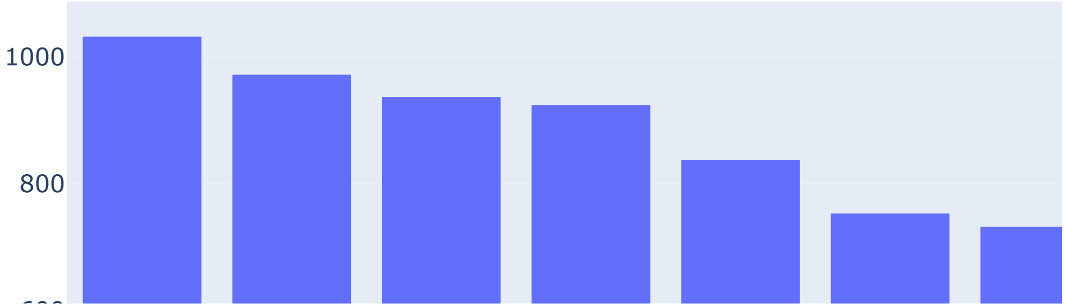
Observation

We have less than 1% of fraudulent cases.

2. Category Frequencies

In [22]:
```
labels = list(data['category'].value_counts().index)
values = list(data['category'].value_counts().values)

# use textposition = 'auto' for direct text
fig = go.Figure(data=[go.Bar(x=labels, y=values,)])
fig.show()
```



Observation

Most transaction fails into gas_transport and the least is the travel.

3. Amount Distribution.

In [23]:
```
fig = px.box(data, x='amt')
fig.show()
```

In [24]:
```
data[data['is_fraud']==1]['amt'].describe()
```

Out[24]:
```
count      59.000000
mean      459.378644
std       371.555876
min         6.560000
25%       185.545000
50%       313.190000
75%       829.120000
max      1107.130000
Name: amt, dtype: float64
```

Observation

The price is not the only detrimant of the fraud.

4. Gender Distribution.

In [25]:
```
labels = list(data['gender'].value_counts().index)
values = list(data['gender'].value_counts().values)
```

In [26]:
```
fig = go.Figure(data=[go.Pie(labels=labels,values=values)])
fig.show()
```

Observation

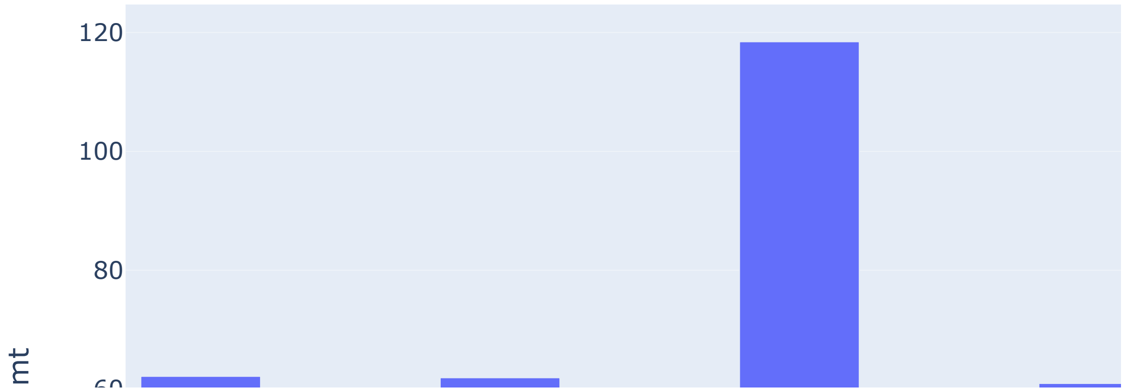
We have 55% of female and 45% of male.

Bi-Variate Analysis

Average Amount per category

In [27]:
```
amt_category = data.groupby('category')['amt'].mean().reset_index()
```

In [28]:
```
fig = px.bar(amt_category, x='category', y='amt')
fig.show()
```



Observation

Average amount of the category grocery_pos is highest and personal_care is lowest.

Feature Engineering

In [29]:
```
data['cc_frequency'] = data['cc_num'].map(data['cc_num'].value_counts())
data['trans_date_trans_time'] = pd.to_datetime(data['trans_date_trans_time'])
data['hour_of_tranx'] = data['trans_date_trans_time'].dt.hour
data['dob'] = pd.to_datetime(data['dob'])
data['age'] = 2022-data['dob'].dt.year
```

In [30]:
```
data.columns
```

Out[30]:
```
Index(['Unnamed: 0', 'trans_date_trans_time', 'cc_num', 'merchant', 'category',
       'amt', 'first', 'last', 'gender', 'street', 'city', 'state', 'zip',
       'lat', 'long', 'city_pop', 'job', 'dob', 'trans_num', 'unix_time',
       'merch_lat', 'merch_long', 'is_fraud', 'cc_frequency', 'hour_of_tranx',
       'age'],
      dtype='object')
```

In [31]:
```
to_drop = ['Unnamed: 0','first','last','gender','trans_date_trans_time','merchant',
           'zip','city_pop','job','trans_num','unix_time','dob']
data = data.drop(to_drop,axis=1)
```

In [32]:
```
data = pd.get_dummies(data=data, columns=['category'],drop_first=True)
```

In [33]:
```
### Splitting the data
```

In [34]:
```
# seperate the input from the output and store as x
x = data.drop(['is_fraud'], axis = 1)

# seperate the output from the input and store as y
y = data.is_fraud
```

In [35]:
```
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,random_state=
```

Train Algorithm

1. Logistic Regression
2. Random Forest
3. Xgboost

Logistic Regression

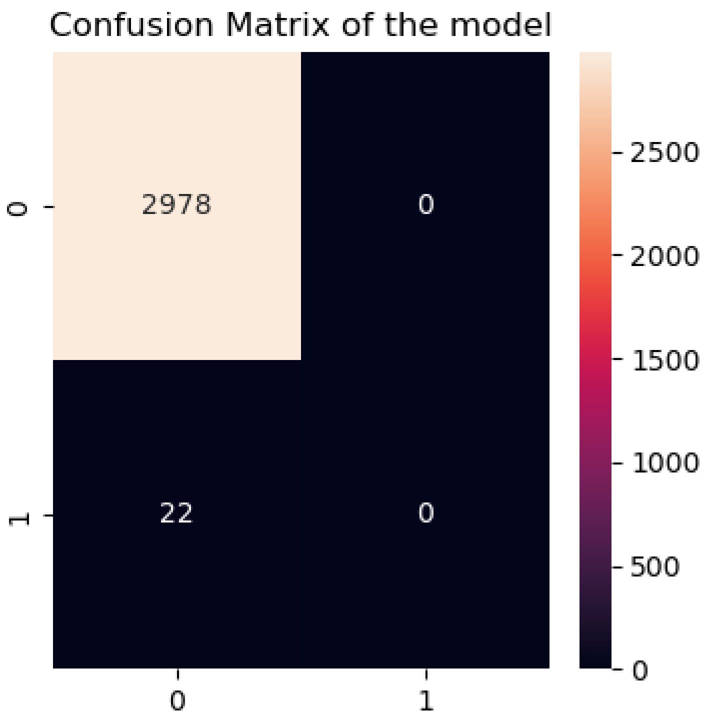
In [36]:
```
def evaluate_model():
    prediction = Logreg.predict(x_test)
    acc_score = accuracy_score(y_test,prediction)
    print("we have {}% accuracy: ".format(acc_score*100))
    # plot predicted vs actual
    plt.figure(figsize=(4,4))
    print(classification_report(y_test,prediction))
    plt.title("Confusion Matrix of the model")
    sns.heatmap(confusion_matrix(y_test,prediction),annot=True, fmt='.5g')
    plt.show()
```

In [37]:
```
Logreg = LogisticRegression()
Logreg.fit(x_train,y_train)
evaluate_model()
```

```
we have 99.26666666666667% accuracy: 
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2978
           1       0.00      0.00      0.00        22

    accuracy                           0.99      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       0.99      0.99      0.99      3000
```



Observation

The model is unable to detect any fraudulent activities, which means the model fails and the business must not deploy.

Random Forest Model

In [38]:
```
rf = RandomForestClassifier(class_weight='balanced')
rf.fit(x_train,y_train)
```

Out[38]:
```
RandomForestClassifier(class_weight='balanced')
```

In [39]:
```
y_pred_rf = rf.predict(x_test)
```

In [40]:
```
accuracy = accuracy_score(y_test,y_pred_rf)
print(accuracy)
```

```
0.9923333333333333
```

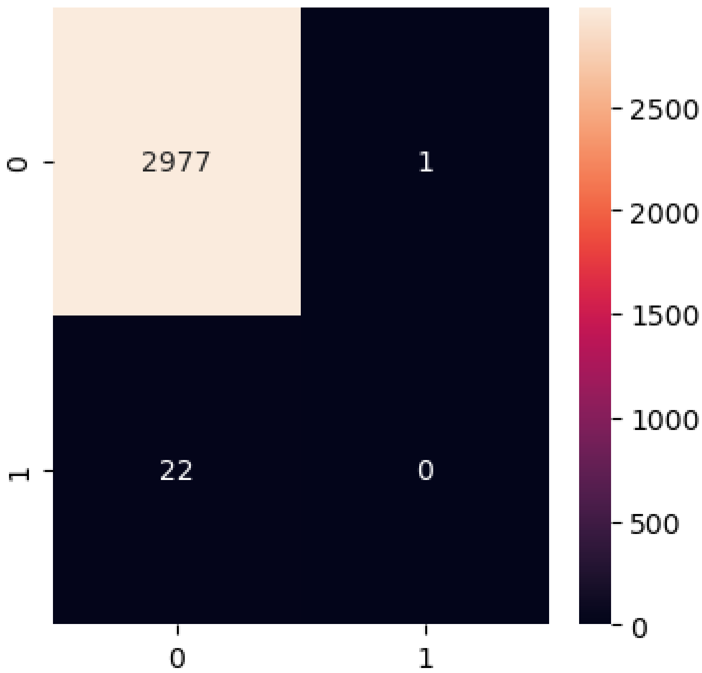
In [41]:
```
report = classification_report(y_test, y_pred_rf)
print('Classification Report:')
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      1.00      2978
           1       0.00      0.00      0.00        22

    accuracy                           0.99      3000
   macro avg       0.50      0.50      0.50      3000
weighted avg       0.99      0.99      0.99      3000
```

In [42]:
```
plt.figure(figsize=(4,4))
sns.heatmap(confusion_matrix(y_test,y_pred_rf),annot=True, fmt='.5g')
plt.show()
```



Observation

The model is unable to detect any fraudulent activities, which means the model fails and the business must not deploy.

Therefore the model fails again.

Xgboost Model

In [43]:
```
xgb = XGBClassifier()
xgb.fit(x_train,y_train)
```

Out[43]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

In [44]:
```
y_pred_xgb = xgb.predict(x_test)
```

In [45]:
```
accuracy = accuracy_score(y_test,y_pred_xgb)
print(accuracy)
```

```
0.9946666666666667
```

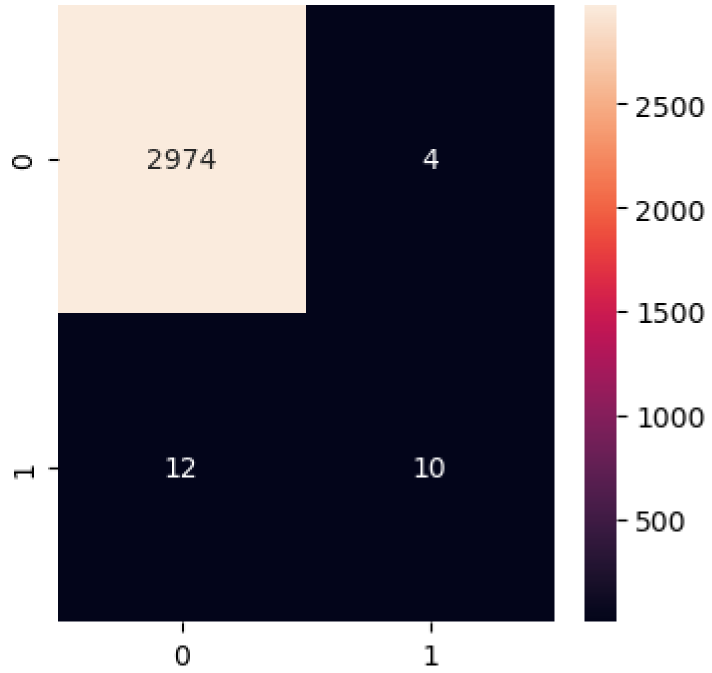
In [46]:
```
report = classification_report(y_test, y_pred_xgb)
print('Classification Report:')
print(report)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2978
           1       0.71      0.45      0.56        22

    accuracy                           0.99      3000
   macro avg       0.86      0.73      0.78      3000
weighted avg       0.99      0.99      0.99      3000
```

In [47]:
```
plt.figure(figsize=(4,4))
sns.heatmap(confusion_matrix(y_test,y_pred_xgb),annot=True, fmt='.5g')
plt.show()
```



Observation

The model is unable to detect any fraudulent activities, which means the model fails and the business must not deploy.

Therefore the model fails again.

Why the model fails?

We should take note that data is heavily imbalanced against the rare cases(Fraudulant activities). The Threshold using predict is <=0.5 == 0 and > 0.5 == 1

Note: Accuracy is not always the right metrics specially when data is imbalanced.

What Next?

Solve the problems by introducing a new metrics and predict probabilities.

Let us tackle the problem by predicting probabilities and using roc_auc curve to get something right for the business.

In [48]:
```
predict_proba = xgb.predict_proba(x_test)[:,1]
```

In [49]:
```
roc_auc_score(y_test,predict_proba)
```

Out[49]:
```
0.965397765431345
```

In [50]:
```
pd.DataFrame(predict_proba).describe()
```

Out[50]:

	0
count	3.000000e+03
mean	5.177819e-03
std	5.848252e-02
min	9.924288e-07
25%	1.145367e-05
50%	3.014189e-05
75%	9.882714e-05
max	9.928535e-01

statistics don't lie, we have that most of test set prediction falls below 0.000015, but I am sure we get it right here.

In [51]:
```
# let us set the threshold with normal threshold
pred_round = np.where(predict_proba <= 0.5,0,1)
```

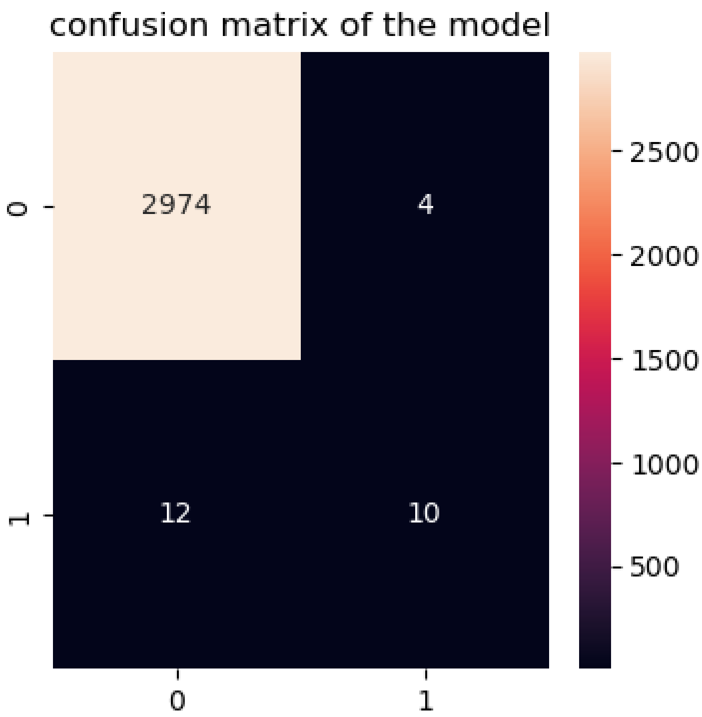
In [52]:
```
plt.figure(figsize=(4,4))
print(classification_report(y_test,pred_round))
plt.title("confusion matrix of the model")
sns.heatmap(confusion_matrix(y_test,pred_round),annot=True, fmt='.5g')
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2978
           1       0.71      0.45      0.56        22

    accuracy                           0.99      3000
   macro avg       0.86      0.73      0.78      3000
weighted avg       0.99      0.99      0.99      3000
```

Out[52]:
```
<AxesSubplot:title={'center':'confusion matrix of the model'}>
```



observation

We have our model can now capture 45% of the misclassified fraudulant cases.

Another Problem??

First talk to your Stakeholders.

Yes we observe we have

1. 12, 10 False Negative(means someone fraud but predicted as non-fraud) == Type 2 Error
2. 4 False Positive (means someone did not fraud but predicted as fraud) == Type 1 Error

Since type 2 Error is greater than type 1 it is the best to save the business by reducing the Type 2 because it is more dangerous.

Storing the model

In [53]:
```
import joblib
joblib.dump(xgb, 'model.joblib')
```

Out[53]:
```
['model.joblib']
```