i. Business Understanding

i.i Fraud Company

- The Fraud Company is a company specialized in detecting fraud in financial transactions made through mobile devices. The company has a service called "Fraud" with no guarantee of blocking fraudulent transactions.
- And the business model of the company is of the Service type with the monetization made by the performance of the service provided, that is, the user pays a fixed fee on the success in detecting fraud in the customer's transactions.

i.i.i Expansion Problem

Fraud Company is expanding in Brazil and to acquire customers more quickly, it has adopted a very aggressive strategy. The strategy works as follows:

1. The company will receive 25% of the value of each transaction that is truly detected as fraud.
2. The company will receive 5% of the value of each transaction detected as fraud, but the transaction is truly legitimate.
3. The company will return 100% of the value to the customer, for each transaction detected as legitimate, however the transaction is truly a fraud.

i.ii The Challenge

Deliver Fraud Company a production model in which my access will be done via API, that is, customers will send their transactions via API so that my model classifies them as fraudulent or legitimate.

i.ii.i Business Questions

1. What is the model's Precision and Accuracy?
2. How Reliable is the model in classifying transactions as legitimate or fraudulent?
3. What is the Expected Billing by the Company if we classify 100% of transactions with the model?
4. What is the Loss Expected by the Company in case of model failure?
5. What is the Profit Expected by the Blocker Fraud Company when using the model?

0.0 Imports and Helper Functions

0.1 Imports

In [ ]:
```
import joblib
import warnings
import inflection

import numpy             as np
import pandas            as pd
import seaborn           as sns

import matplotlib.pyplot as plt

from scipy   import stats
from boruta  import BorutaPy
from category_encoders import OneHotEncoder

from IPython.display      import Image
from IPython.core.display import HTML

from xgboost  import XGBClassifier
#from lightgbm import LGBMClassifier

from sklearn.svm          import SVC
from sklearn.dummy        import DummyClassifier
from sklearn.ensemble     import RandomForestClassifier
from sklearn.neighbors    import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression

from sklearn.metrics         import balanced_accuracy_score, precision_score
from sklearn.metrics         import recall_score, f1_score, make_scorer, coh
from sklearn.preprocessing   import MinMaxScaler
from sklearn.model_selection import GridSearchCV, train_test_split, Stratifi
```

0.2 Helper Functions

In [ ]:
```
warnings.filterwarnings('ignore')

seed = 42
np.random.seed(seed)
```

In [ ]:
```
def jupyter_settings():
    %matplotlib inline
    %pylab inline

    sns.set(font_scale=1.6)

    #plt.style.use('seaborn-darkgrid')
    plt.rcParams['figure.figsize'] = [25, 12]
```

```
    plt.rcParams['font.size'] = 16

    display( HTML('<style>.container { width:100% !important; }</style>'))
    pd.options.display.max_columns = None
    pd.options.display.max_rows = None
    pd.set_option('display.expand_frame_repr', False)

jupyter_settings()
```

```
%pylab is deprecated, use %matplotlib inline and import the required librari
es.
Populating the interactive namespace from numpy and matplotlib
```

In [ ]:
```
def ml_scores(model_name, y_true, y_pred):

    accuracy = balanced_accuracy_score(y_true, y_pred)
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    kappa = cohen_kappa_score(y_true, y_pred)

    return pd.DataFrame({'Balanced Accuracy': np.round(accuracy, 3),
                         'Precision': np.round(precision, 3),
                         'Recall': np.round(recall, 3),
                         'F1': np.round(f1, 3),
                         'Kappa': np.round(kappa, 3)},
                        index=[model_name])
```

In [ ]:
```
def calcCramerV(x, y):
    cm = pd.crosstab(x, y).values
    n = cm.sum()
    r, k = cm.shape

    chi2 = stats.chi2_contingency(cm)[0]
    chi2corr = max(0, chi2 - (k-1)*(r-1)/(n-1))

    kcorr = k - (k-1)**2/(n-1)
    rcorr = r - (r-1)**2/(n-1)

    return np.sqrt((chi2corr/n) / (min(kcorr-1, rcorr-1)))
```

In [ ]:
```
def ml_cv_results(model_name, model, x, y, verbose=1):

    '''initial'''
    balanced_accuracies = []
    precisions = []
    recalls = []
    f1s = []
    kappas = []

    mm = MinMaxScaler()

    x_ = x.to_numpy()
    y_ = y.to_numpy()

    count = 0
```

```
    '''cross-validation'''
    skf = StratifiedKFold(n_splits=5, shuffle=True)

    for index_train, index_test in skf.split(x_, y_):
        ## Showing the Fold
        if verbose > 0:
            count += 1
            print('Fold K=%i' % (count))

        ## selecting train and test
        x_train, x_test = x.iloc[index_train], x.iloc[index_test]
        y_train, y_test = y.iloc[index_train], y.iloc[index_test]

        ## applying the scale
        x_train = mm.fit_transform(x_train)
        x_test = mm.transform(x_test)

        ## training the model
        model.fit(x_train, y_train)
        y_pred = model.predict(x_test)

        ## saving the metrics
        balanced_accuracies.append(balanced_accuracy_score(y_test, y_pred))
        precisions.append(precision_score(y_test, y_pred))
        recalls.append(recall_score(y_test, y_pred))
        f1s.append(f1_score(y_test, y_pred))
        kappas.append(cohen_kappa_score(y_test, y_pred))


    '''results'''
    accuracy_mean, accuracy_std = np.round(np.mean(balanced_accuracies), 3),
    precision_mean, precision_std = np.round(np.mean(precisions), 3), np.rou
    recall_mean, recall_std = np.round(np.mean(recalls), 3), np.round(np.std
    f1_mean, f1_std = np.round(np.mean(f1s), 3), np.round(np.std(f1s), 3)
    kappa_mean, kappa_std = np.round(np.mean(kappas), 3), np.round(np.std(ka

    ## saving the results in a dataframe
    return pd.DataFrame({"Balanced Accuracy": "{} +/- {}".format(accuracy_me
                        "Precision": "{} +/- {}".format(precision_mean, prec
                        "Recall": "{} +/- {}".format(recall_mean, recall_std
                        "F1": "{} +/- {}".format(f1_mean, f1_std),
                        "Kappa": "{} +/- {}".format(kappa_mean, kappa_std)},
                       index=[model_name])
```

1.0 Data Description

1.1 Loading Data

In [ ]:
```
df1 = pd.read_csv('/Users/kausik/Desktop/MS Data Science/DSC 680 Applied Dat
```

In [ ]:
```
df1.head()
```

Out[ ]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nar
0	283	CASH_IN	210329.84	C1159819632	3778062.79	3988392.64	C1218
1	132	CASH_OUT	215489.19	C1372369468	21518.00	0.00	C467
2	355	DEBIT	4431.05	C1059822709	20674.00	16242.95	C76
3	135	CASH_OUT	214026.20	C1464960643	46909.73	0.00	C1059
4	381	CASH_OUT	8858.45	C831134427	0.00	0.00	C579

In [ ]:

```
df1.tail()
```

Out[ ]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig
636257	351	CASH_OUT	28761.10	C742050657	0.0	0.00
636258	184	CASH_OUT	167820.71	C561181412	62265.0	0.00
636259	35	PAYMENT	8898.12	C1773417333	30808.0	21909.88
636260	277	CASH_OUT	176147.90	C1423233247	83669.0	0.00
636261	304	CASH_OUT	95142.89	C874575079	0.0	0.00

1.2 Columns

1.2.1 Column Descriptions

step: maps a unit of time in the real world. In this case 1 step is 1 hour of time. Total steps 744 (30 days simulation).

type: CASH-IN, CASH-OUT, DEBIT, PAYMENT and TRANSFER.

amount: amount of the transaction in local currency.

nameOrig: customer who started the transaction

oldbalanceOrg: initial balance before the transaction

newbalanceOrig: new balance after the transaction

nameDest: customer who is the recipient of the transaction

oldbalanceDest: initial balance recipient before the transaction. Note that there is not information for customers that start with M (Merchants).

newbalanceDest: new balance recipient after the transaction. Note that there is not information for customers that start with M (Merchants).

isFraud: This is the transactions made by the fraudulent agents inside the simulation. In this specific dataset the fraudulent behavior of the agents aims to profit by taking control or customers accounts and try to empty the funds by transferring to another account and then cashing out of the system.

isFlaggedFraud: The business model aims to control massive transfers from one account to another and flags illegal attempts. An illegal attempt in this dataset is an attempt to transfer more than 200.000 in a single transaction.

1.2.2 Column Rename

In [ ]:
```
cols_old = df1.columns.tolist()

snakecase = lambda x: inflection.underscore(x)
cols_new = list(map(snakecase, cols_old))

df1.columns = cols_new
```

In [ ]:
```
df1.columns
```

Out[ ]:
```
Index(['step', 'type', 'amount', 'name_orig', 'oldbalance_org',
       'newbalance_orig', 'name_dest', 'oldbalance_dest', 'newbalance_des
t',
       'is_fraud', 'is_flagged_fraud'],
      dtype='object')
```

1.3 Data Dimension

In [ ]:
```
print('Number of Rows: {}'.format(df1.shape[0]))
print('Number of Cols: {}'.format(df1.shape[1]))
```

```
Number of Rows: 636262
Number of Cols: 11
```

1.4 Data Types and Structure

In [ ]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 636262 entries, 0 to 636261
Data columns (total 11 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   step               636262 non-null  int64  
 1   type               636262 non-null  object 
 2   amount             636262 non-null  float64
 3   name_orig          636262 non-null  object 
 4   oldbalance_org     636262 non-null  float64
 5   newbalance_orig    636262 non-null  float64
 6   name_dest          636262 non-null  object 
 7   oldbalance_dest    636262 non-null  float64
 8   newbalance_dest    636262 non-null  float64
 9   is_fraud           636262 non-null  int64  
 10  is_flagged_fraud   636262 non-null  int64  
dtypes: float64(5), int64(3), object(3)
memory usage: 53.4+ MB
```

1.5 Check NA

In [ ]:
```
df1.isna().mean()
```

Out[ ]:
```
step                0.0
type                0.0
amount              0.0
name_orig           0.0
oldbalance_org      0.0
newbalance_orig     0.0
name_dest           0.0
oldbalance_dest     0.0
newbalance_dest     0.0
is_fraud            0.0
is_flagged_fraud    0.0
dtype: float64
```

1.6 Fill Out NA

There's no NaN values to fill.

1.7 Change Data Type

I will change the values 0 and 1 to 'yes' and 'no'. It'll help on the data description and analysis sections.

In [ ]:
```
df1['is_fraud'] = df1['is_fraud'].map({1: 'yes', 0: 'no'})
df1['is_flagged_fraud'] = df1['is_flagged_fraud'].map({1: 'yes', 0: 'no'})
```

1.8 Description Statistics

In [ ]:
```
num_attributes = df1.select_dtypes(exclude='object')
cat_attributes = df1.select_dtypes(include='object')
```

1.8.1 Numerical Attributes

In [ ]:
```
describe = num_attributes.describe().T

describe['range'] = (num_attributes.max() - num_attributes.min()).tolist()
describe['variation coefficient'] = (num_attributes.std() / num_attributes.m
describe['skew'] = num_attributes.skew().tolist()
describe['kurtosis'] = num_attributes.kurtosis().tolist()

describe
```

Out[ ]:

	count	mean	std	min	25%	50%
step	636262.0	2.429319e+02	1.423309e+02	1.0	155.000	238.000
amount	636262.0	1.800585e+05	6.069714e+05	0.0	13407.425	74815.770
oldbalance_org	636262.0	8.317937e+05	2.885636e+06	0.0	0.000	14239.000
newbalance_orig	636262.0	8.528354e+05	2.921296e+06	0.0	0.000	0.000
oldbalance_dest	636262.0	1.096212e+06	3.375389e+06	0.0	0.000	131539.745
newbalance_dest	636262.0	1.221809e+06	3.656213e+06	0.0	0.000	214712.725

- All the data has a coeficient of variation greater than 25%, therefore they aren't homogeneous.
- The step variable starts from 1 hour to 742 hour (30 days).
- Some variables are higher shap and right skewed.
- 50% of the newbalance_orig is 0. Maybe there are some transfers that don't go to the destination.
- The skew is higher positive, therefore the values may be in less values.

1.8.2 Categorical Attributes

In [ ]:
```
cat_attributes.describe()
```

Out[ ]:

	type	name_orig	name_dest	is_fraud	is_flagged_fraud
count	636262	636262	636262	636262	636262
unique	5	636171	457224	2	2
top	CASH_OUT	C334643493	C2083562754	no	no
freq	224216	2	14	635441	636260

- The majority type is cash_out with 2237500.
- There's a lot of variability in name_orig, so it could be hard to use one hot encoding.
- There's less name_orig than name_dest. There's more users sending than receiving, however use one hot encoding will not help.
- There's more fraud than the flagged fraud, it shows that the current method can't recognize fraud efficiently.

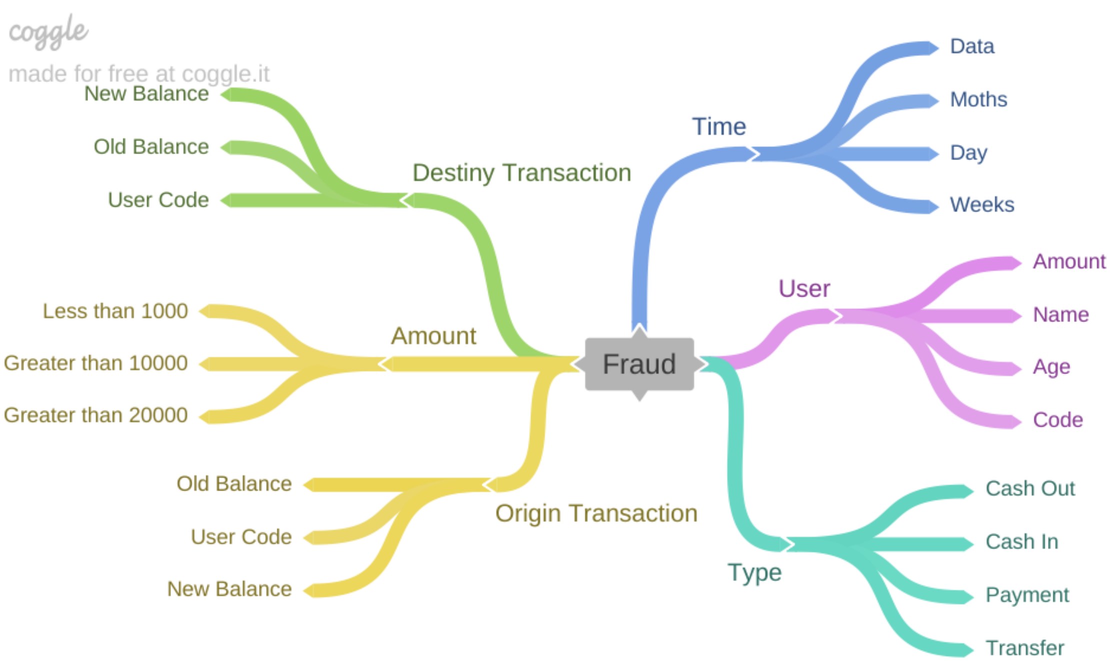
2.0 Feature Engineering

In [ ]:
```
df2 = df1.copy()
```

2.1 Mind Map

In [ ]:
```
Image('/Users/kausik/Desktop/MS Data Science/DSC 680 Applied Data Science/ch
```

Out[ ]:



2.2 Hypothesis Creation

2.2.1 User

- 90% of the twentyone-year-old users did a fraud transiction.
- The majority fraud transiction occours for the same initial letter user.
- The fraud amount is greater than 10.000.
- The 60% of the age is greater than 30 year old.

2.2.2 Type

- 60% of fraud transaction occours using cash-out-type method.
- The majority transfers occours using tranfers-type method.
- Values greater than 100.000 occours using transfers-type method.
- Payment type occurs with values lower than 100.000

2.2.3 Origin and Destiny Transactions

- 60% of the difference between origin destiny transactions is equal 0 for frauds.
- Origin values are greater than destiny values for fraud transaction.

2.2.4 Time

- Fraud transactions occours at least in 3 days.
- 40% of the cash-out transactions occours less than 1 day.
- 60% of the transaction less than 100.000 occours at least 10 days.
- The transactions greater than 10.000 occours at most in 2 weeks.

2.3 Hipothesys List

1. The majority fraud transiction occours for the same initial letter user.

2. All the fraud amount is greater than 10.000.

3. 60% of fraud transaction occours using cash-out-type method.

4. The majority transfers occours using tranfers-type method.

5. Fraud transactions occours at least in 3 days.

2.4 Feature Engineering

In [ ]:
```
# step
df2['step_days'] = df2['step'].apply(lambda i: i/24)
df2['step_weeks'] = df2['step'].apply(lambda i: i/(24*7))

# difference between initial balance before the transaction and new balance
df2['diff_new_old_balance'] = df2['newbalance_orig'] - df2['oldbalance_org']

# difference between initial balance recipient before the transaction and ne
df2['diff_new_old_destiny'] = df2['newbalance_dest'] - df2['oldbalance_dest'

# name orig and name dest
df2['name_orig'] = df2['name_orig'].apply(lambda i: i[0])
df2['name_dest'] = df2['name_dest'].apply(lambda i: i[0])
```

3.0 Selecting Columns

In [ ]:
```
df3 = df2.copy()
```

3.1 Selecting Columns

I'll use all the columns for data analysis

3.2 Selecting Lines

I'll use all the lines.

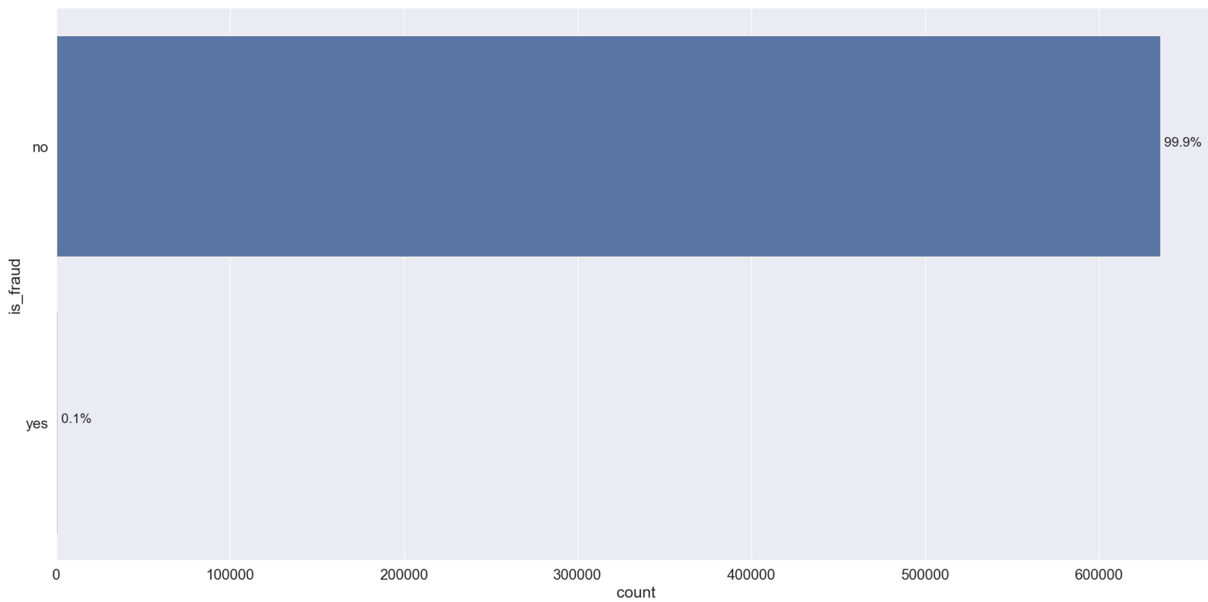
4.0 Exploratory Data Analisys

In [ ]:
```
df4 = df3.copy()
```

4.1 Univariate Analysis

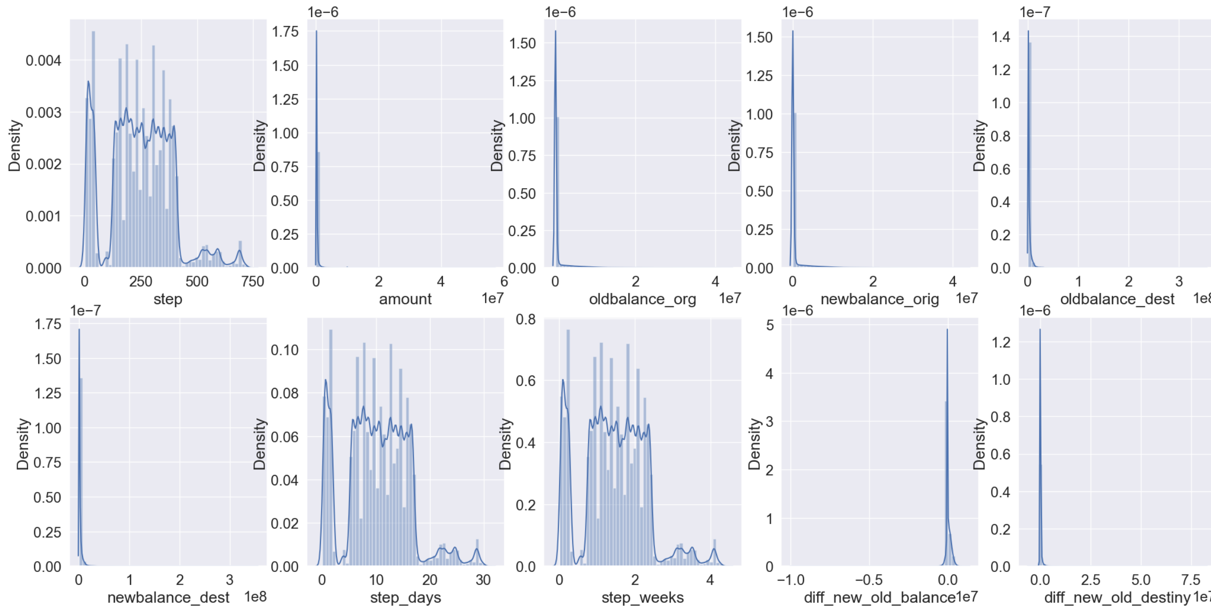
4.1.1 Response Variable

In [ ]:
```
ax = sns.countplot(y='is_fraud', data=df4);

total = df4['is_fraud'].size
for p in ax.patches:
        percentage = ' {:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + p.get_width() + 0.02
        y = p.get_y() + p.get_height()/2
        ax.annotate(percentage, (x, y))
```



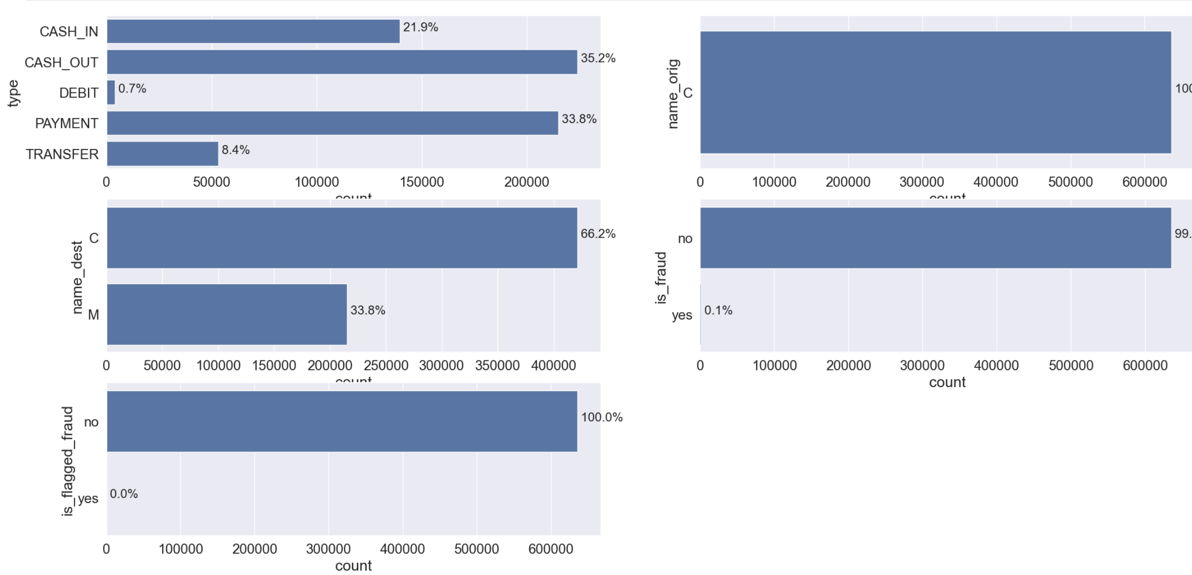
4.1.2 Numerical Variables

In [ ]:
```
num_attributes = df4.select_dtypes(exclude='object')
columns = num_attributes.columns.tolist()
j = 1

for column in columns:
    plt.subplot(2, 5, j)
    sns.distplot(num_attributes[column]);

    j += 1
```



4.1.3 Categorical Variables

In [ ]:
```
cat_attributes = df4.select_dtypes(include='object')
columns = cat_attributes.columns.tolist()
j = 1

for column in columns:
    plt.subplot(3, 2, j)
    ax = sns.countplot(y=column, data=cat_attributes)

    total = cat_attributes[column].size
    for p in ax.patches:
        percentage = ' {:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + p.get_width() + 0.02
        y = p.get_y() + p.get_height()/2
        ax.annotate(percentage, (x, y))

    j += 1
```

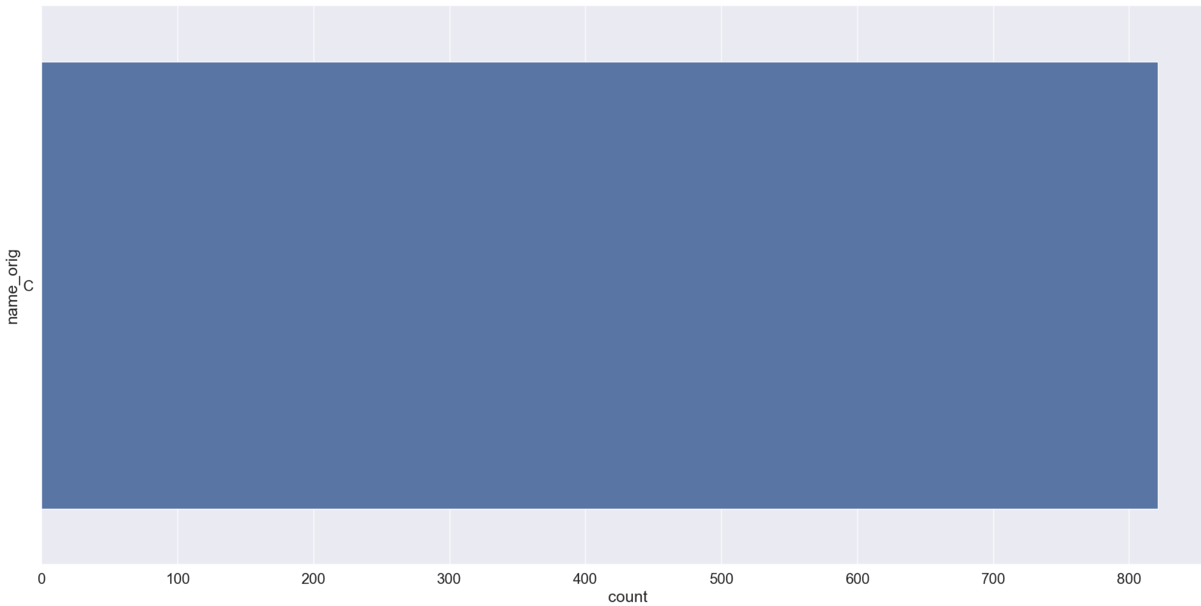


4.2 Bivariate Analysis

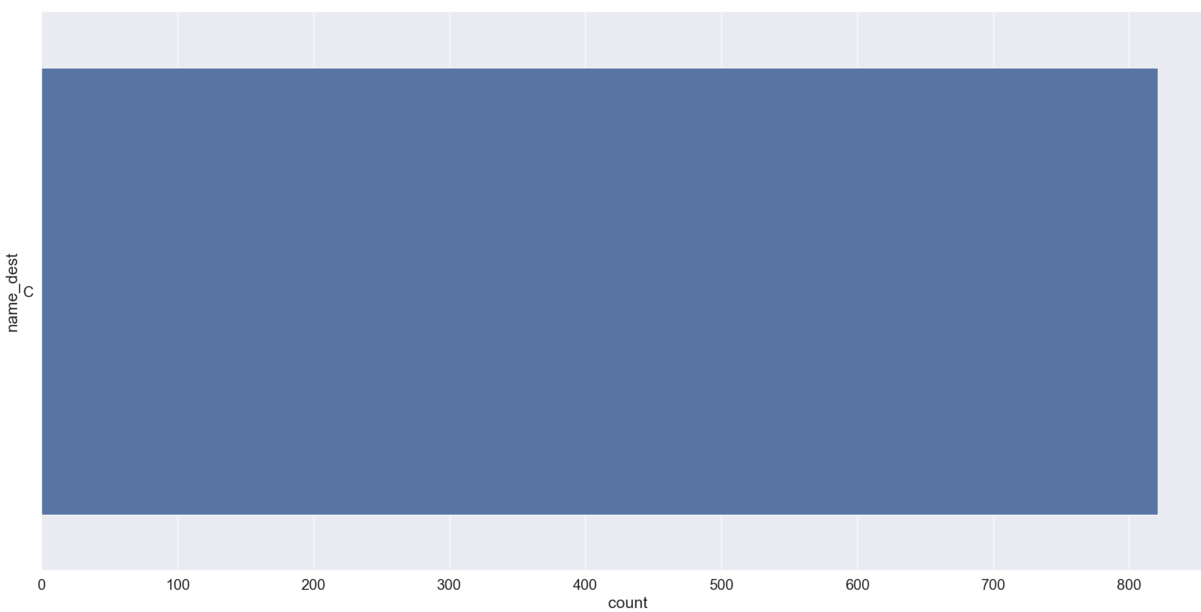
H1 The majority fraud transiction occours for the same user.

TRUE: The same user origem and destiny has got the same inital letter.

In [ ]:
```
aux1 = df4[df4['is_fraud'] == 'yes']
sns.countplot(y='name_orig', data=aux1);
```



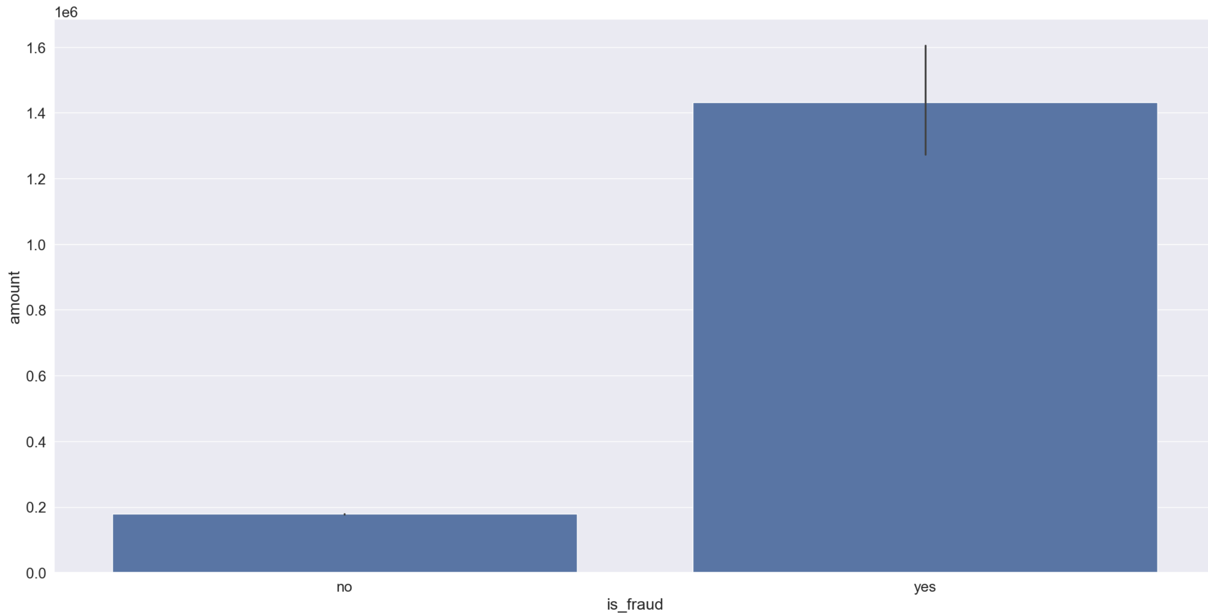
In [ ]:
```
sns.countplot(y='name_dest', data=aux1);
```



H2 All the fraud amount is greater than 10.000.

TRUE: The values are greater than 10.000. But it's important to note that the no-fraud values is greater than 100.000 also.

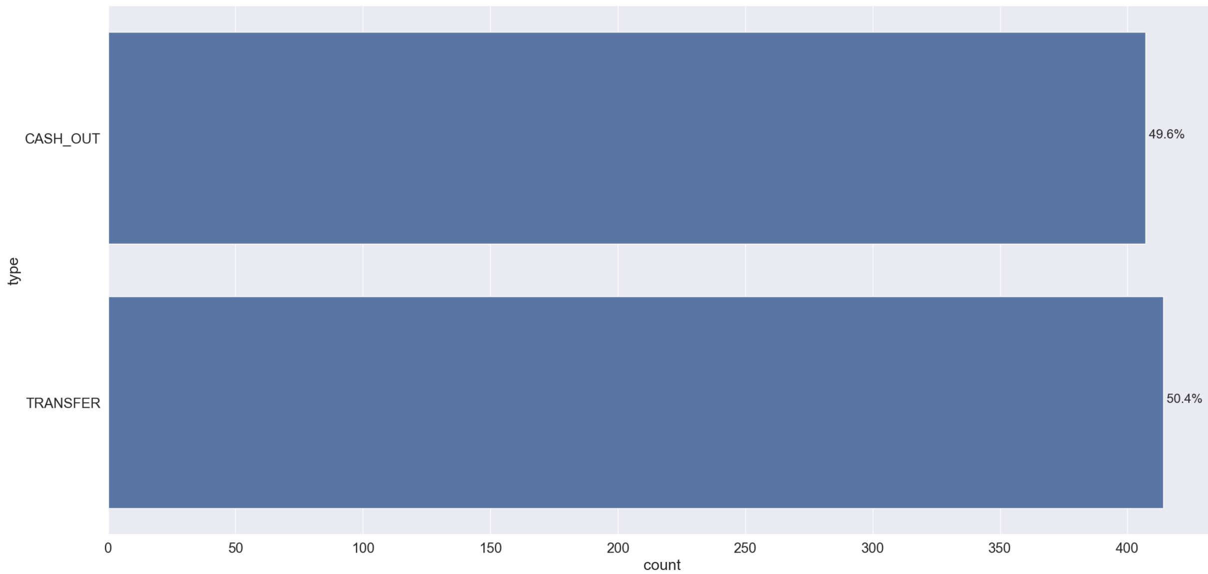
In [ ]:
```
sns.barplot(y='amount', x='is_fraud', data=df4);
```



H3 60% of fraud transaction occours using cash-out-type method.

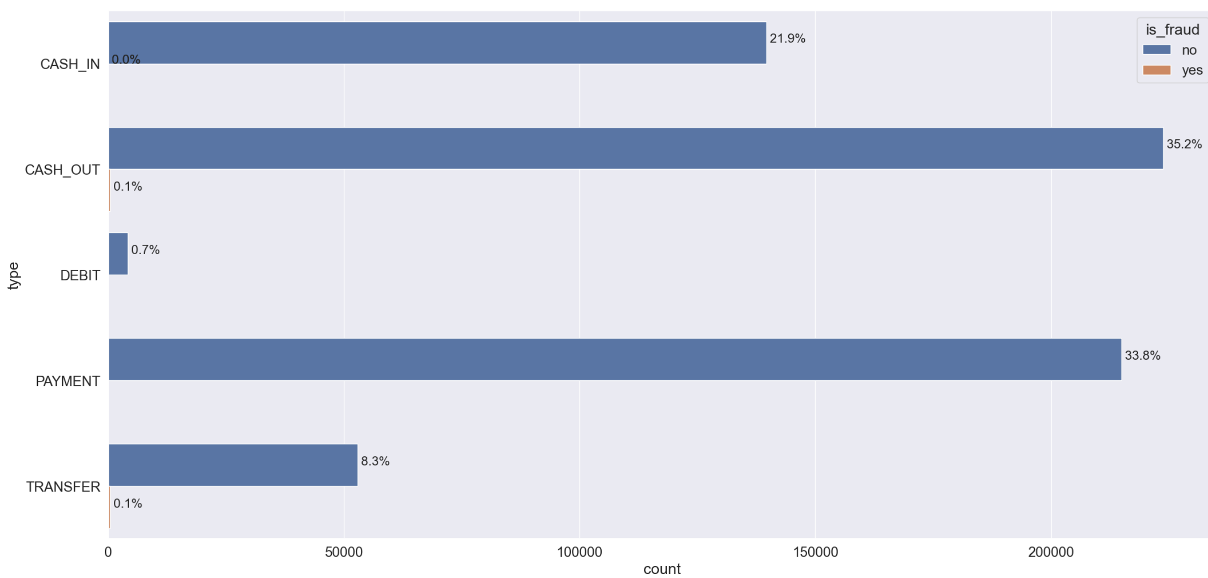
FALSE: The fraud transaction occours in transfer and cash-out type. However they're almost the same value.

In [ ]:
```
aux1 = df4[df4['is_fraud'] == 'yes']
ax = sns.countplot(y='type', data=aux1)

total = aux1['type'].size
for p in ax.patches:
        percentage = ' {:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + p.get_width() + 0.02
        y = p.get_y() + p.get_height()/2
        ax.annotate(percentage, (x, y))
```



To see the complete transiction-type and I'll plot them here.

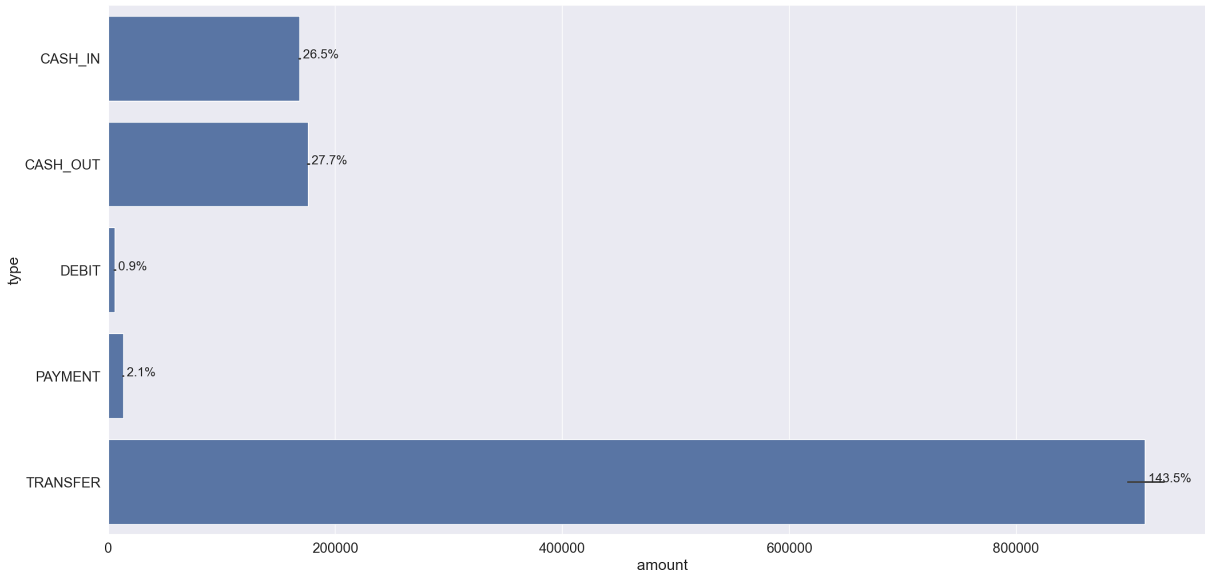
In [ ]:
```
ax = sns.countplot(y='type', hue='is_fraud', data=df4)

total = df4['type'].size
for p in ax.patches:
        percentage = ' {:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + p.get_width() + 0.02
        y = p.get_y() + p.get_height()/2
        ax.annotate(percentage, (x, y))
```



H4 Values greater than 100.000 occours using transfers-type method.

FALSE: The majority transactions occours in trasnfer-type, however transactions greater than 100.000 occour in cash-out and cash-in too.

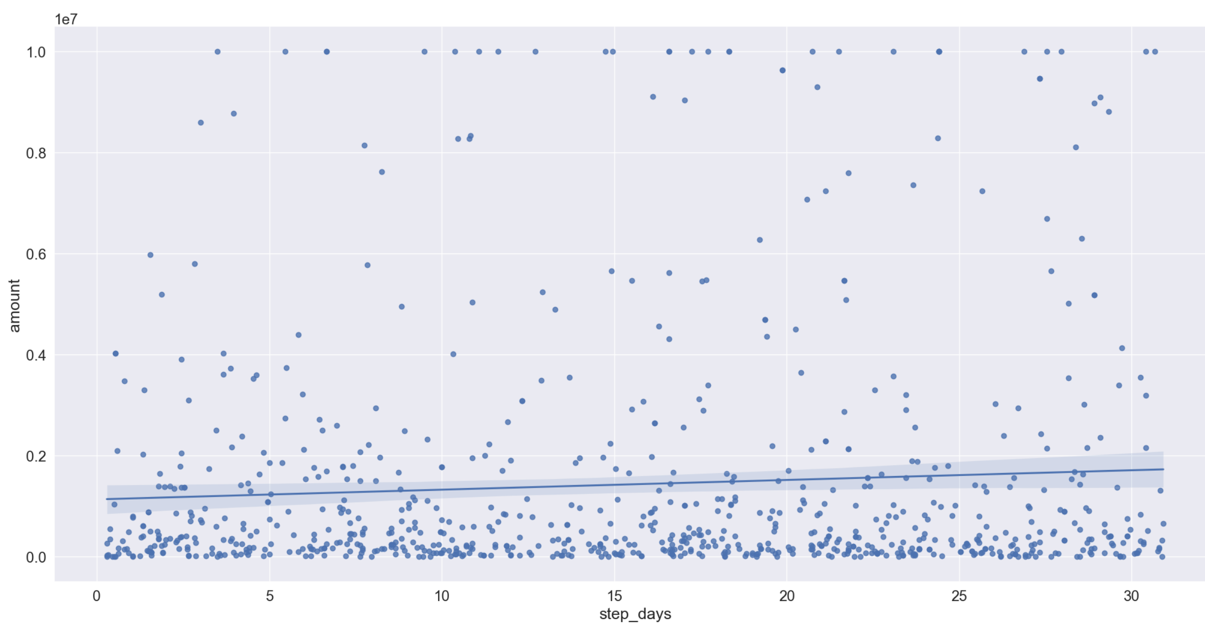
In [ ]:
```
ax = sns.barplot(y='type', x='amount', data=df4);

total = df4['type'].size
for p in ax.patches:
        percentage = ' {:.1f}%'.format(100 * p.get_width()/total)
        x = p.get_x() + p.get_width() + 0.02
        y = p.get_y() + p.get_height()/2
        ax.annotate(percentage, (x, y))
```



H5 Fraud transactions occours at least in 3 days.

TRUE: The values for transactions and days in fraud aren't similar.

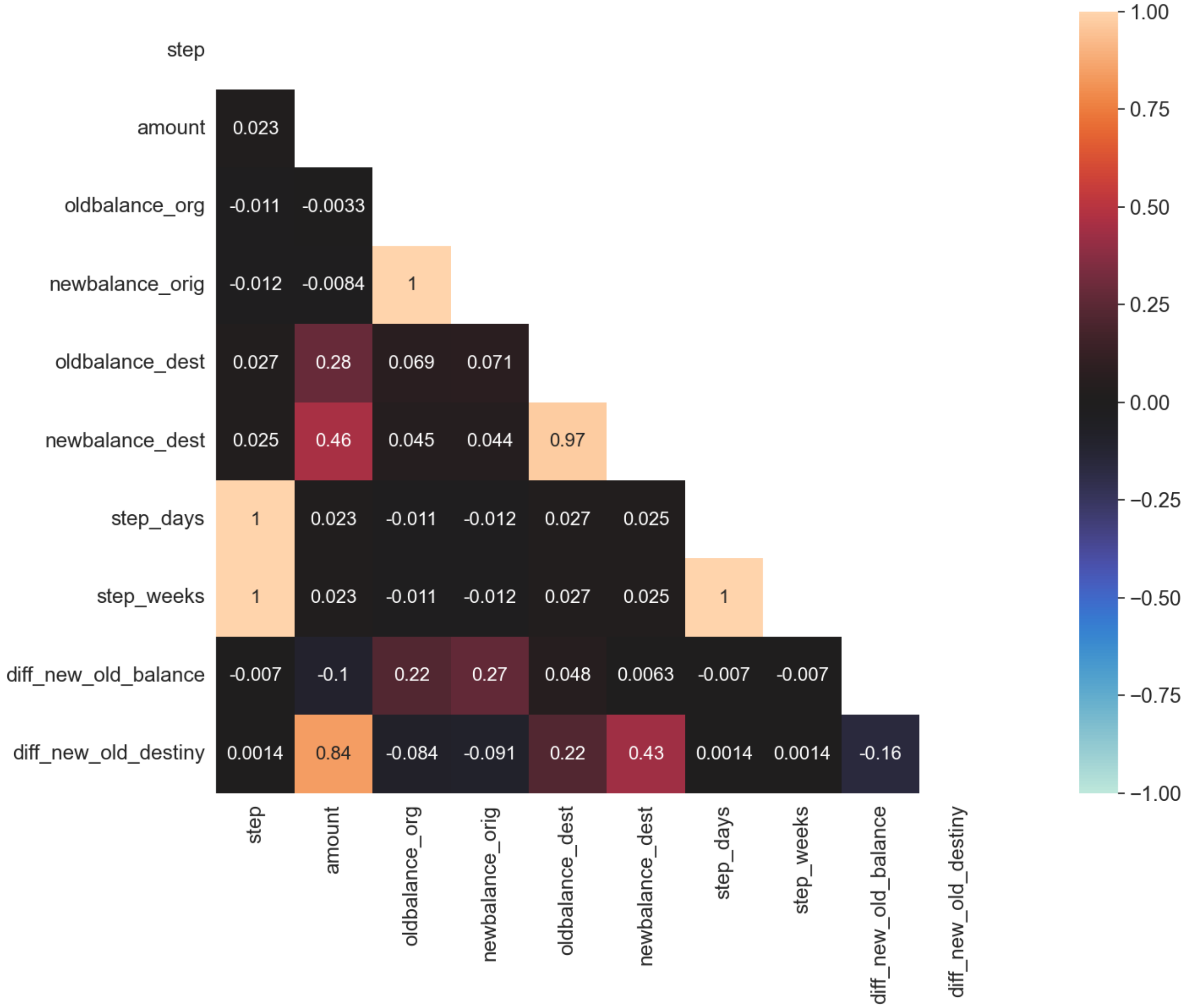
In [ ]:
```
aux1 = df4[df4['is_fraud'] == 'yes']
sns.regplot(x='step_days', y='amount', data=aux1);
```



4.3 Multivariaty Analysis

4.3.1 Numerical Analysis

In [ ]:
```
corr = num_attributes.corr()

mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True

with sns.axes_style("white"):
    ax = sns.heatmap(corr, annot=True, mask=mask, vmin=-1, center=0, vmax=1,
```

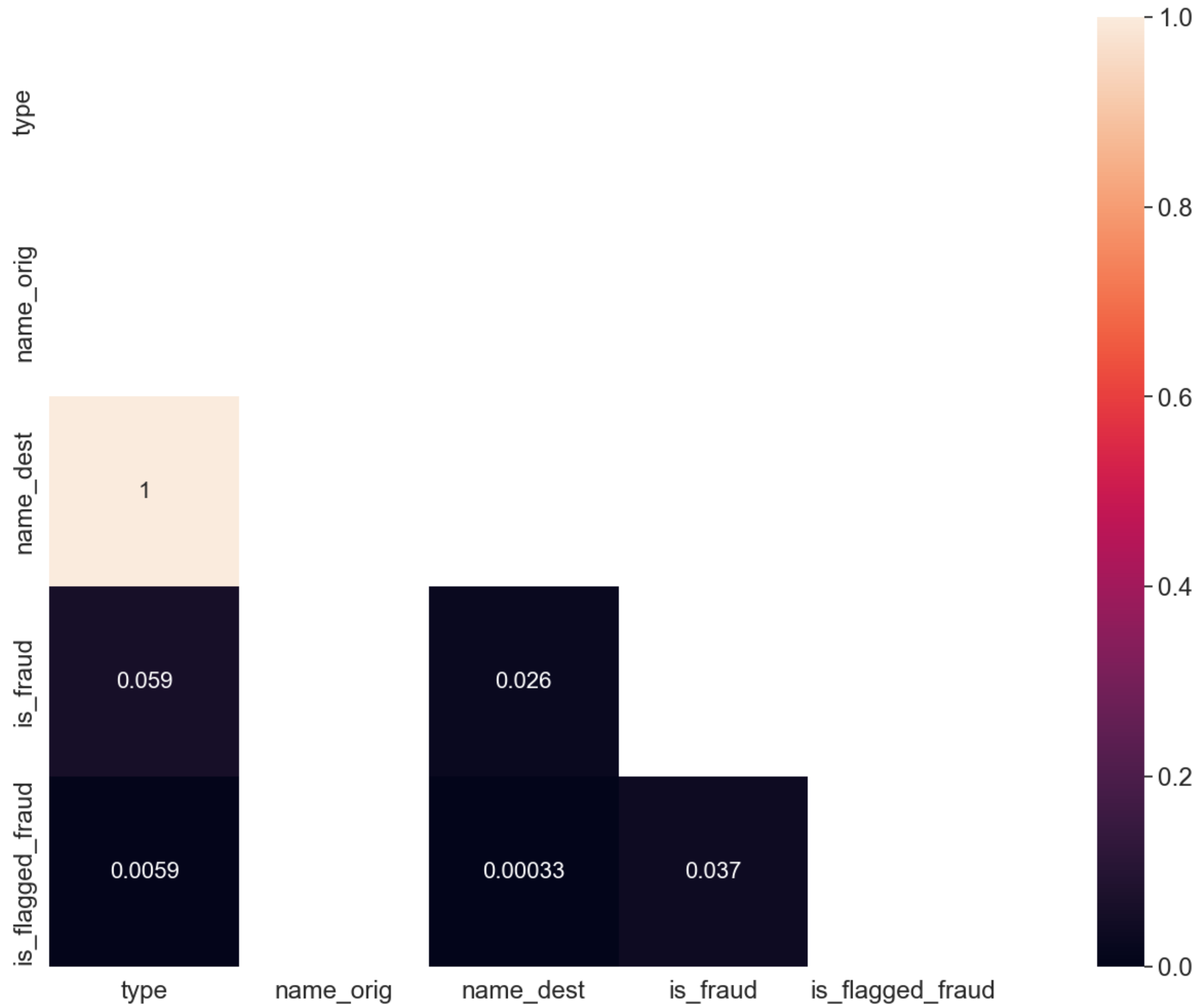


4.3.2 Categorical Variables

In [ ]:
```
dict_corr = {}
columns = cat_attributes.columns.tolist()

for column in columns:
    dict_corr[column] = {}

    for column2 in columns:
        dict_corr[column][column2] = calcCramerV(cat_attributes[column], cat
```

```
corr = pd.DataFrame(dict_corr)
```

In [ ]:
```
mask = np.zeros_like(corr)
mask[np.triu_indices_from(mask)] = True

with sns.axes_style("white"):
    ax = sns.heatmap(corr, annot=True, mask=mask, vmin=0, vmax=1, square=Tru
```



5.0 Data Preparation

In [ ]:
```
df5 = df4.copy()
```

5.1 Spliting into Train, Valid and Test

In [ ]:
```
X = df5.drop(columns=['is_fraud', 'is_flagged_fraud', 'name_orig', 'name_des
                      'step_weeks', 'step_days'], axis=1)
y = df5['is_fraud'].map({'yes': 1, 'no': 0})
```

In [ ]:
```
# spliting into temp and test
```

```
X_temp, X_test, y_temp, y_test = train_test_split(X, y, test_size=.2, strati
```

In [ ]:
```
# spliting into train and valid
X_train, X_valid, y_train, y_valid = train_test_split(X_temp, y_temp, test_s
```

5.2 One Hot Encoder

In [ ]:
```
ohe = OneHotEncoder(cols=['type'], use_cat_names=True)

X_train = ohe.fit_transform(X_train)
X_valid = ohe.transform(X_valid)

X_temp = ohe.fit_transform(X_temp)
X_test = ohe.transform(X_test)
```

5.3 Rescaling

In [ ]:
```
num_columns = ['amount', 'oldbalance_org', 'newbalance_orig', 'oldbalance_de
               'diff_new_old_balance', 'diff_new_old_destiny']
mm = MinMaxScaler()
X_params = X_temp.copy()

X_train[num_columns] = mm.fit_transform(X_train[num_columns])
X_valid[num_columns] = mm.transform(X_valid[num_columns])

X_params[num_columns] = mm.fit_transform(X_temp[num_columns])
X_test[num_columns] = mm.transform(X_test[num_columns])
```

6.0 Feature Selection

6.1 Boruta

In [ ]:
```
# X_boruta = X_params.values
# y_boruta = y_temp.values.ravel()
```

In [ ]:
```
# boruta = BorutaPy(RandomForestClassifier(), n_estimators='auto')
# boruta.fit(X_boruta, y_boruta)
```

6.1.1 Best Features

In [ ]:
```
# cols_selected_boruta = boruta.support_.tolist()
```

In [ ]:
```
# columns_selected = X_params.loc[:, cols_selected_boruta].columns.tolist()
```

In [ ]:
```
# columns_selected
```

In [ ]:
```
# ['step',
#  'amount',
#  'oldbalance_org',
#  'newbalance_orig',
#  'oldbalance_dest',
#  'newbalance_dest',
#  'diff_new_old_balance',
#  'diff_new_old_destiny',
#  'type_TRANSFER']
```

In [ ]:
```
final_columns_selected = ['step', 'oldbalance_org',
                          'newbalance_orig', 'newbalance_dest',
                          'diff_new_old_balance', 'diff_new_old_destiny',
                          'type_TRANSFER']
```

7.0 Machine Learning Modeling

In [ ]:
```
X_train_cs = X_train[final_columns_selected]
X_valid_cs = X_valid[final_columns_selected]

X_temp_cs = X_temp[final_columns_selected]
X_test_cs = X_test[final_columns_selected]

X_params_cs = X_params[final_columns_selected]
```

7.1 Baseline

In [ ]:
```
dummy = DummyClassifier()
dummy.fit(X_train_cs, y_train)

y_pred = dummy.predict(X_valid_cs)
```

In [ ]:
```
dummy_results = ml_scores('dummy', y_valid, y_pred)
dummy_results
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
dummy	0.5	0.0	0.0	0.0	0.0

7.1.1 Classification Report

In [ ]:
```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    101671
           1       0.00      0.00      0.00       131

    accuracy                           1.00    101802
   macro avg       0.50      0.50      0.50    101802
weighted avg       1.00      1.00      1.00    101802
```

7.1.2 Cross Validation

In [ ]:
```
dummy_cv = ml_cv_results('Dummy', DummyClassifier(), X_temp, y_temp)
dummy_cv
```

```
Fold K=1
Fold K=2
Fold K=3
Fold K=4
Fold K=5
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
Dummy	0.5 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0

7.2 Logistic Regression

In [ ]:
```
lg = LogisticRegression()
lg.fit(X_train_cs, y_train)

y_pred = lg.predict(X_valid_cs)
```

In [ ]:
```
lg_results = ml_scores('Logistic Regression', y_valid, y_pred)
lg_results
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
Logistic Regression	0.5	0.0	0.0	0.0	0.0

7.2.1 Classification Report

In [ ]:
```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    101671
           1       0.00      0.00      0.00       131

    accuracy                           1.00    101802
   macro avg       0.50      0.50      0.50    101802
weighted avg       1.00      1.00      1.00    101802
```

7.2.2 Cross Validation

In [ ]:
```
lg_cv = ml_cv_results('Logistic Regression',
                      LogisticRegression(),
                      X_temp_cs, y_temp)
lg_cv
```

```
Fold K=1
Fold K=2
Fold K=3
Fold K=4
Fold K=5
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
Logistic Regression	0.542 +/- 0.024	1.0 +/- 0.0	0.084 +/- 0.049	0.151 +/- 0.079	0.151 +/- 0.079

7.3 K Nearest Neighbors

In [ ]:
```
knn = KNeighborsClassifier()
knn.fit(X_train_cs, y_train)

y_pred = knn.predict(X_valid_cs)
```

In [ ]:
```
knn_results = ml_scores('K Nearest Neighbors', y_valid, y_pred)
knn_results
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
K Nearest Neighbors	0.565	1.0	0.13	0.23	0.23

7.3.1 Classification Report

In [ ]:
```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    101671
           1       1.00      0.13      0.23       131

    accuracy                           1.00    101802
   macro avg       1.00      0.56      0.61    101802
weighted avg       1.00      1.00      1.00    101802
```

7.3.2 Cross Validation

In [ ]:
```
knn_cv = ml_cv_results('K Nearest Neighbors', KNeighborsClassifier(),
                       X_temp_cs, y_temp)
```

```
knn_cv
```

```
Fold K=1
Fold K=2
Fold K=3
Fold K=4
Fold K=5
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
K Nearest Neighbors	0.705 +/- 0.017	0.943 +/- 0.033	0.411 +/- 0.034	0.572 +/- 0.038	0.572 +/- 0.038

7.4 Support Vector Machine

In [ ]:
```
svm = SVC()
svm.fit(X_train_cs, y_train)

y_pred = svm.predict(X_valid_cs)
```

In [ ]:
```
svm_results = ml_scores('SVM', y_valid, y_pred)
svm_results
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
SVM	0.5	0.0	0.0	0.0	0.0

7.4.1 Classification Report

In [ ]:
```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    101671
           1       0.00      0.00      0.00       131

    accuracy                           1.00    101802
   macro avg       0.50      0.50      0.50    101802
weighted avg       1.00      1.00      1.00    101802
```

7.4.2 Cross Validation

In [ ]:
```
svm_cv = ml_cv_results('SVM', SVC(), X_temp_cs, y_temp)
svm_cv
```

```
Fold K=1
Fold K=2
Fold K=3
Fold K=4
Fold K=5
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
SVM	0.596 +/- 0.017	1.0 +/- 0.0	0.192 +/- 0.034	0.32 +/- 0.047	0.32 +/- 0.047

7.5 Random Forest

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

y_pred = rf.predict(X_valid_cs)
```

In [ ]:
```
rf_results = ml_scores('Random Forest', y_valid, y_pred)
rf_results
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
Random Forest	0.844	0.978	0.687	0.807	0.807

7.5.1 Classification Report

In [ ]:
```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    101671
           1       0.98      0.69      0.81       131

    accuracy                           1.00    101802
   macro avg       0.99      0.84      0.90    101802
weighted avg       1.00      1.00      1.00    101802
```

7.5.2 Cross Validation

In [ ]:
```
rf_cv = ml_cv_results('Random Forest',
                      RandomForestClassifier(),
                      X_temp_cs, y_temp)
rf_cv
```

```
Fold K=1
Fold K=2
Fold K=3
Fold K=4
Fold K=5
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
Random Forest	0.861 +/- 0.019	0.969 +/- 0.018	0.721 +/- 0.038	0.827 +/- 0.029	0.827 +/- 0.029

7.6 XGBoost

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

y_pred = xgb.predict(X_valid_cs)
```

In [ ]:
```
xgb_results = ml_scores('XGBoost', y_valid, y_pred)
xgb_results
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
XGBoost	0.874	0.942	0.748	0.834	0.834

7.6.1 Classification Report

In [ ]:
```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    101671
           1       0.94      0.75      0.83       131

    accuracy                           1.00    101802
   macro avg       0.97      0.87      0.92    101802
weighted avg       1.00      1.00      1.00    101802
```

7.6.2 Cross Validation

In [ ]:
```
xgb_cv = ml_cv_results('XGBoost', XGBClassifier(),
                       X_temp_cs, y_temp)
xgb_cv
```

```
Fold K=1
Fold K=2
Fold K=3
Fold K=4
Fold K=5
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
XGBoost	0.887 +/- 0.021	0.938 +/- 0.017	0.775 +/- 0.042	0.848 +/- 0.023	0.848 +/- 0.023

7.7 LightGBM

In [ ]:
```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in /Library/Frameworks/Python.framew
ork/Versions/3.12/lib/python3.12/site-packages (4.3.0)
Requirement already satisfied: numpy in /Library/Frameworks/Python.framewor
k/Versions/3.12/lib/python3.12/site-packages (from lightgbm) (1.26.3)
Requirement already satisfied: scipy in /Library/Frameworks/Python.framewor
k/Versions/3.12/lib/python3.12/site-packages (from lightgbm) (1.12.0)
```

In [ ]:
```
from lightgbm import LGBMClassifier
lightgbm = LGBMClassifier()
lightgbm.fit(X_train_cs, y_train)

y_pred = lightgbm.predict(X_valid_cs)
```

```
[LightGBM] [Info] Number of positive: 526, number of negative: 406681
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.004844 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1532
[LightGBM] [Info] Number of data points in the train set: 407207, number of
used features: 7
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001292 -> initscore=-6.650
483
[LightGBM] [Info] Start training from score -6.650483
```

In [ ]:
```
lightgbm_results = ml_scores('LightGBM', y_valid, y_pred)
lightgbm_results
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
LightGBM	0.683	0.063	0.374	0.107	0.105

7.7.1 Classification Report

In [ ]:
```
print(classification_report(y_valid, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      1.00    101671
           1       0.06      0.37      0.11       131

    accuracy                           0.99    101802
   macro avg       0.53      0.68      0.55    101802
weighted avg       1.00      0.99      0.99    101802
```

7.7.2 Cross Validation

In [ ]:
```
lightgbm_cv = ml_cv_results('LightGDM', LGBMClassifier(),
                            X_temp_cs, y_temp)
lightgbm_cv
```

```
Fold K=1
[LightGBM] [Info] Number of positive: 526, number of negative: 406681
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.004108 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1532
[LightGBM] [Info] Number of data points in the train set: 407207, number of
used features: 7
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001292 -> initscore=-6.650
483
[LightGBM] [Info] Start training from score -6.650483
Fold K=2
[LightGBM] [Info] Number of positive: 526, number of negative: 406681
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of te
sting was 0.034742 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1532
[LightGBM] [Info] Number of data points in the train set: 407207, number of
used features: 7
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001292 -> initscore=-6.650
483
[LightGBM] [Info] Start training from score -6.650483
Fold K=3
[LightGBM] [Info] Number of positive: 525, number of negative: 406682
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.004007 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1532
[LightGBM] [Info] Number of data points in the train set: 407207, number of
used features: 7
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001289 -> initscore=-6.652
389
[LightGBM] [Info] Start training from score -6.652389
Fold K=4
[LightGBM] [Info] Number of positive: 525, number of negative: 406682
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.004014 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1532
[LightGBM] [Info] Number of data points in the train set: 407207, number of
used features: 7
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001289 -> initscore=-6.652
389
[LightGBM] [Info] Start training from score -6.652389
Fold K=5
[LightGBM] [Info] Number of positive: 526, number of negative: 406682
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of te
sting was 0.004159 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1532
[LightGBM] [Info] Number of data points in the train set: 407208, number of
used features: 7
```

```
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.001292 -> initscore=-6.650
486
[LightGBM] [Info] Start training from score -6.650486
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
LightGDM	0.665 +/- 0.09	0.13 +/- 0.158	0.343 +/- 0.174	0.169 +/- 0.177	0.167 +/- 0.178

7.8 Comparing Model's Performance

7.8.1 Single Performance

In [ ]:
```
modeling_performance = pd.concat([dummy_results, lg_results, knn_results,
                                  rf_results, xgb_results, lightgbm_results,
                                 svm_results])
modeling_performance.sort_values(by="F1", ascending=True)
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
dummy	0.500	0.000	0.000	0.000	0.000
Logistic Regression	0.500	0.000	0.000	0.000	0.000
SVM	0.500	0.000	0.000	0.000	0.000
LightGBM	0.683	0.063	0.374	0.107	0.105
K Nearest Neighbors	0.565	1.000	0.130	0.230	0.230
Random Forest	0.844	0.978	0.687	0.807	0.807
XGBoost	0.874	0.942	0.748	0.834	0.834

7.8.2 Cross Validation Performance

In [ ]:
```
modeling_performance_cv = pd.concat([dummy_cv, lg_cv, knn_cv, rf_cv,
                                     xgb_cv, lightgbm_cv, svm_cv])

modeling_performance_cv.sort_values(by="F1", ascending=True)
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
Dummy	0.5 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0	0.0 +/- 0.0
Logistic Regression	0.542 +/- 0.024	1.0 +/- 0.0	0.084 +/- 0.049	0.151 +/- 0.079	0.151 +/- 0.079
LightGDM	0.665 +/- 0.09	0.13 +/- 0.158	0.343 +/- 0.174	0.169 +/- 0.177	0.167 +/- 0.178
SVM	0.596 +/- 0.017	1.0 +/- 0.0	0.192 +/- 0.034	0.32 +/- 0.047	0.32 +/- 0.047
K Nearest Neighbors	0.705 +/- 0.017	0.943 +/- 0.033	0.411 +/- 0.034	0.572 +/- 0.038	0.572 +/- 0.038
Random Forest	0.861 +/- 0.019	0.969 +/- 0.018	0.721 +/- 0.038	0.827 +/- 0.029	0.827 +/- 0.029
XGBoost	0.887 +/- 0.021	0.938 +/- 0.017	0.775 +/- 0.042	0.848 +/- 0.023	0.848 +/- 0.023

8.0 Hyperparameter Fine Tuning

In [ ]:
```
f1 = make_scorer(f1_score)
```

In [ ]:
```
params = {
    'booster': ['gbtree', 'gblinear', 'dart'],
    'eta': [0.3, 0.1, 0.01],
    'scale_pos_weight': [1, 774, 508, 99]
}
```

In [ ]:
```
gs = GridSearchCV(XGBClassifier(),
                  param_grid=params,
                  scoring=f1,
                  cv=StratifiedKFold(n_splits=5))

gs.fit(X_params_cs, y_temp)
```

Out[ ]:
```
▸ GridSearchCV
▸ estimator: XGBClassifier
    ▸ XGBClassifier
```

In [ ]:
```
best_params = gs.best_params_
best_params
```

Out[ ]: `{'booster': 'gbtree', 'eta': 0.3, 'scale_pos_weight': 1}`

In [ ]:
```
best_params = {'booster': 'gbtree', 'eta': 0.3, 'scale_pos_weight': 1}
```

In [ ]:
```
gs.best_score_
```

Out[ ]:
```
0.8475894936731084
```

8.1 Results

In [ ]:
```
xgb_gs = XGBClassifier(
    booster=best_params['booster'],
    eta=best_params['eta'],
    scale_pos_weight=best_params['scale_pos_weight']
)
```

In [ ]:
```
xgb_gs.fit(X_train_cs, y_train)
```

Out[ ]:

XGBClassifier

```
XGBClassifier(base_score=None, booster='gbtree', callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_ro
unds=None,
              enable_categorical=False, eta=0.3, 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=N
one,
```

In [ ]:
```
y_pred = xgb_gs.predict(X_valid_cs)
```

8.1.2 Single Results

In [ ]:
```
xgb_gs_results = ml_scores('XGBoost GS', y_valid, y_pred)
xgb_gs_results
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
XGBoost GS	0.874	0.942	0.748	0.834	0.834

8.1.3 Cross Validation

In [ ]:
```
xgb_gs_cv = ml_cv_results('XGBoost GS', xgb_gs, X_temp_cs, y_temp)
xgb_gs_cv
```

```
Fold K=1
Fold K=2
Fold K=3
Fold K=4
Fold K=5
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
XGBoost GS	0.877 +/- 0.01	0.943 +/- 0.018	0.753 +/- 0.02	0.837 +/- 0.018	0.837 +/- 0.018

9.0 Conclusions

9.1 Final Model

In [ ]:

```
final_model = XGBClassifier(
    booster=best_params['booster'],
    eta=best_params['eta'],
    scale_pos_weight=best_params['scale_pos_weight']
)

final_model.fit(X_params_cs, y_temp)
```

Out[ ]:

```
▾ XGBClassifier
XGBClassifier(base_score=None, booster='gbtree', callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_ro
unds=None,
              enable_categorical=False, eta=0.3, 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=N
one,
```

9.1.1 Unseen Data Score

In [ ]:

```
y_pred = final_model.predict(X_test_cs)
```

In [ ]:

```
unseen_scores = ml_scores('unseen', y_test, y_pred)
unseen_scores
```

Out[ ]:

	Balanced Accuracy	Precision	Recall	F1	Kappa
unseen	0.912	0.957	0.823	0.885	0.885

9.2 Fraud Company Expasion

9.2.1 The company receives 25% of each transaction value truly detected as fraud.

In [ ]:
```
df_test = df5.loc[X_test.index, :]
df_test['predictions'] = y_pred
```

In [ ]:
```
aux1 = df_test[(df_test['is_fraud'] == 'yes') & (df_test['predictions'] == 1
receives = aux1['amount'].sum() * 0.25
```

In [ ]:
```
print('The company can receive %.2f detecting fraud transactions.' % (receiv
```

```
The company can receive 60638881.09 detecting fraud transactions.
```

9.2.2 The company receives 5% of each transaction value detected as fraud, however the transaction is legitimate.

In [ ]:
```
aux1 = df_test[(df_test['is_fraud'] == 'no') & (df_test['predictions'] == 1)
receives = aux1['amount'].sum() * 0.05

print('For wrong decisions, the company can receive %.2f.' % (receives))
```

```
For wrong decisions, the company can receive 108137.29.
```

9.2.3 The company gives back 100% of the value for the customer in each transaction detected as legitimate, however the transaction is actually a fraud.

In [ ]:
```
aux1 = df_test[(df_test['is_fraud'] == 'yes') & (df_test['predictions'] == 0
receives = aux1['amount'].sum()

print('However, the company must return the amount of %.2f.' % (receives))
```

```
However, the company must return the amount of 3445682.59.
```

9.3 Model's Performance

9.3.1 What is the model's Precision and Accuracy?

In [ ]:
```
print('For unseen data, the values of balanced accuracy is equal %.2f and pr
```

```
For unseen data, the values of balanced accuracy is equal 0.91 and precision
is equal 0.96.
```

9.3.2 How reliable is the model in classifying transactions as legitimate or fraudulent?

In [ ]:
```
print('The model can detect 0.851 +/- 0.023 of the fraud. However it detecte
```

```
The model can detect 0.851 +/- 0.023 of the fraud. However it detected 0.84
of the frauds from a unseen data.
```

9.3.3 What is the revenue expected by the company classify 100% of transactions with the model?

In [ ]:
```
aux1 = df_test[(df_test['is_fraud'] == 'yes') & (df_test['predictions'] == 1
receives = aux1['amount'].sum() * 0.25

aux2 = df_test[(df_test['is_fraud'] == 'no') & (df_test['predictions'] == 1)
receives2 = aux2['amount'].sum() * 0.05

print('Using the model the company can revenue %.2f.' % (receives + receives
```

```
Using the model the company can revenue 60747018.38.
```

In [ ]:
```
aux3 = df_test[(df_test['is_fraud'] == 'yes') & (df_test['is_flagged_fraud']
curr_receives = aux3['amount'].sum() * 0.25

aux4 = df_test[(df_test['is_fraud'] == 'no') & (df_test['is_flagged_fraud']
curr_receives2 = aux4['amount'].sum() * 0.05

print('However the currently method the revenue is %.2f.' % (curr_receives +
```

```
However the currently method the revenue is 0.00.
```

9.3.4 What is the loss expected by the Company if it classifies 100% of the transactions with the model?

In [ ]:
```
aux1 = df_test[(df_test['is_fraud'] == 'yes') & (df_test['predictions'] == 0
loss = aux1['amount'].sum()

print('For wrong classifications the company must return the amount of %.2f.
```

```
For wrong classifications the company must return the amount of 3445682.59.
```

In [ ]:
```
aux1 = df_test[(df_test['is_fraud'] == 'yes') & (df_test['is_flagged_fraud']
curr_loss = aux1['amount'].sum()

print('For wrong classifications using the currently method, the company mus
```

```
For wrong classifications using the currently method, the company must retur
n the amount of 246001206.94.
```

9.3.5 What is the profit expected by the blocker fraud company when using the model?

In [ ]:
```
print('The company can expect the profit of %.2f.' % (receives + receives2 -
```

```
The company can expect the profit of 57301335.79.
```

In [ ]:
```
print('Using the currently method, the profit is %.2f.' % (curr_receives + c
```

```
Using the currently method, the profit is -246001206.94.
```

10.0 Model Deploy

10.1 Saving

In [ ]:
```python
final_model = XGBClassifier(
    booster=best_params['booster'],
    eta=best_params['eta'],
    scale_pos_weight=best_params['scale_pos_weight']
)

final_model.fit(X_params_cs, y_temp)

joblib.dump(final_model, '../models/model_cycle1.joblib')
```

Out[ ]:
```
['../models/model_cycle1.joblib']
```

In [ ]:
```python
mm = MinMaxScaler()
mm.fit(X_params_cs, y_temp)

joblib.dump(mm, '../functions/minmaxscaler_cycle1.joblib')
```

Out[ ]:
```
['../functions/minmaxscaler_cycle1.joblib']
```

In [ ]:
```python
joblib.dump(ohe, '../functions/onehotencoder_cycle1.joblib')
```

Out[ ]:
```
['../functions/onehotencoder_cycle1.joblib']
```

In [ ]:
```python

```