Fraud Detection (5% isFraud)

April 20, 2020

What is the likelihood of there being financial fraud?

Importing the packages needed for analysis

[1]:
```
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
from sklearn.model_selection import train_test_split, learning_curve
from sklearn.metrics import average_precision_score
import statsmodels.api as sm
```

EDA on dataset

[2]:
```
fraud = pd.read_csv("Fraud_Detection.csv")
```

[3]:
```
fraud.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 11 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            object
 2   amount          float64
 3   nameOrig        object
 4   oldbalanceOrg   float64
 5   newbalanceOrig  float64
 6   nameDest        object
 7   oldbalanceDest  float64
 8   newbalanceDest  float64
 9   isFraud         int64
 10  isFlaggedFraud  int64
dtypes: float64(5), int64(3), object(3)
memory usage: 534.0+ MB
```

[4]:
```
fraud.head()
```

[4]:
```
   step      type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1   PAYMENT   9839.64  C1231006815       170136.0       160296.36
1     1   PAYMENT   1864.28  C1666544295        21249.0        19384.72
2     1  TRANSFER    181.00  C1305486145          181.0            0.00
3     1  CASH_OUT    181.00   C840083671          181.0            0.00
4     1   PAYMENT  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0
```

Changing the type to numerical category for pairplot later on

[5]:
```
fraud['type'] = fraud['type'].astype('category')
```

[6]:
```
fraud['type'].unique()
```

[6]:
```
[PAYMENT, TRANSFER, CASH_OUT, DEBIT, CASH_IN]
Categories (5, object): [PAYMENT, TRANSFER, CASH_OUT, DEBIT, CASH_IN]
```

[7]:
```
fraud['type'] = fraud['type'].map( {'PAYMENT': 1, 'TRANSFER':2, 'CASH_IN':3 ,␣
↪'CASH_OUT': 4, 'DEBIT': 5} ).astype(int)
```

[8]:
```
fraud.head()
```

[8]:
```
   step  type    amount     nameOrig  oldbalanceOrg  newbalanceOrig  \
0     1     1   9839.64  C1231006815       170136.0       160296.36
1     1     1   1864.28  C1666544295        21249.0        19384.72
2     1     2    181.00  C1305486145          181.0            0.00
3     1     4    181.00   C840083671          181.0            0.00
4     1     1  11668.14  C2048537720        41554.0        29885.86

      nameDest  oldbalanceDest  newbalanceDest  isFraud  isFlaggedFraud
0  M1979787155             0.0             0.0        0               0
1  M2044282225             0.0             0.0        0               0
2   C553264065             0.0             0.0        1               0
3    C38997010         21182.0             0.0        1               0
4  M1230701703             0.0             0.0        0               0
```

Deleting the objects because they will not be needed for the future analysis portion

[9]:
```
del fraud['nameOrig']
del fraud['nameDest']
del fraud['isFlaggedFraud']
```

[10]:
```
fraud.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6362620 entries, 0 to 6362619
Data columns (total 8 columns):
 #   Column          Dtype
---  ------          -----
 0   step            int64
 1   type            int64
 2   amount          float64
 3   oldbalanceOrg   float64
 4   newbalanceOrig  float64
 5   oldbalanceDest  float64
 6   newbalanceDest  float64
 7   isFraud         int64
dtypes: float64(5), int64(3)
memory usage: 388.3 MB
```

[11]:
```
fraud.describe().T
```

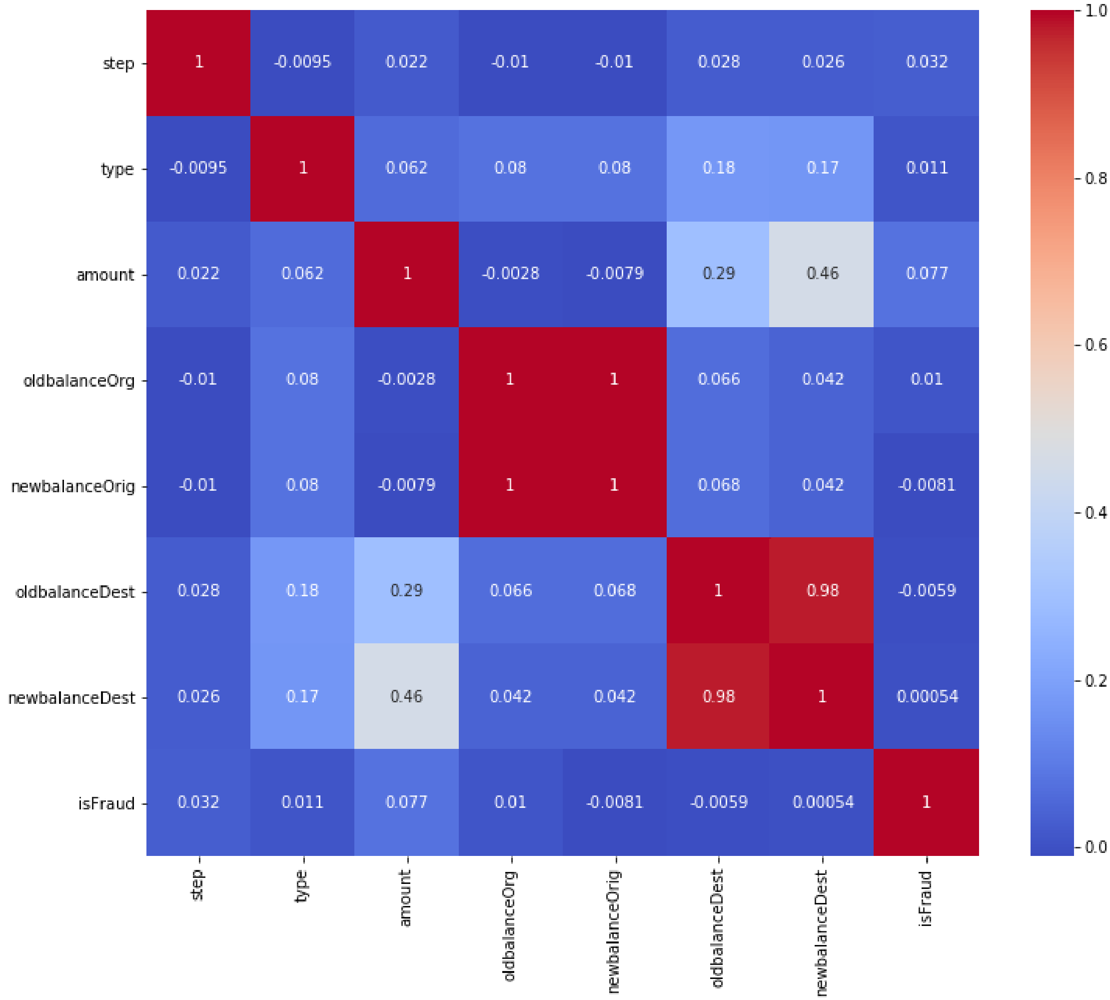
[11]:
```
                    count          mean           std  min       25%  \
step            6362620.0  2.433972e+02  1.423320e+02  1.0    156.00
type            6362620.0  2.604638e+00  1.287533e+00  1.0      1.00
amount          6362620.0  1.798619e+05  6.038582e+05  0.0  13389.57
oldbalanceOrg   6362620.0  8.338831e+05  2.888243e+06  0.0      0.00
newbalanceOrig  6362620.0  8.551137e+05  2.924049e+06  0.0      0.00
oldbalanceDest  6362620.0  1.100702e+06  3.399180e+06  0.0      0.00
newbalanceDest  6362620.0  1.224996e+06  3.674129e+06  0.0      0.00
isFraud         6362620.0  1.290820e-03  3.590480e-02  0.0      0.00

                       50%           75%           max
step               239.000  3.350000e+02  7.430000e+02
type                 3.000  4.000000e+00  5.000000e+00
amount           74871.940  2.087215e+05  9.244552e+07
oldbalanceOrg    14208.000  1.073152e+05  5.958504e+07
newbalanceOrig       0.000  1.442584e+05  4.958504e+07
oldbalanceDest  132705.665  9.430367e+05  3.560159e+08
newbalanceDest  214661.440  1.111909e+06  3.561793e+08
isFraud              0.000  0.000000e+00  1.000000e+00
```

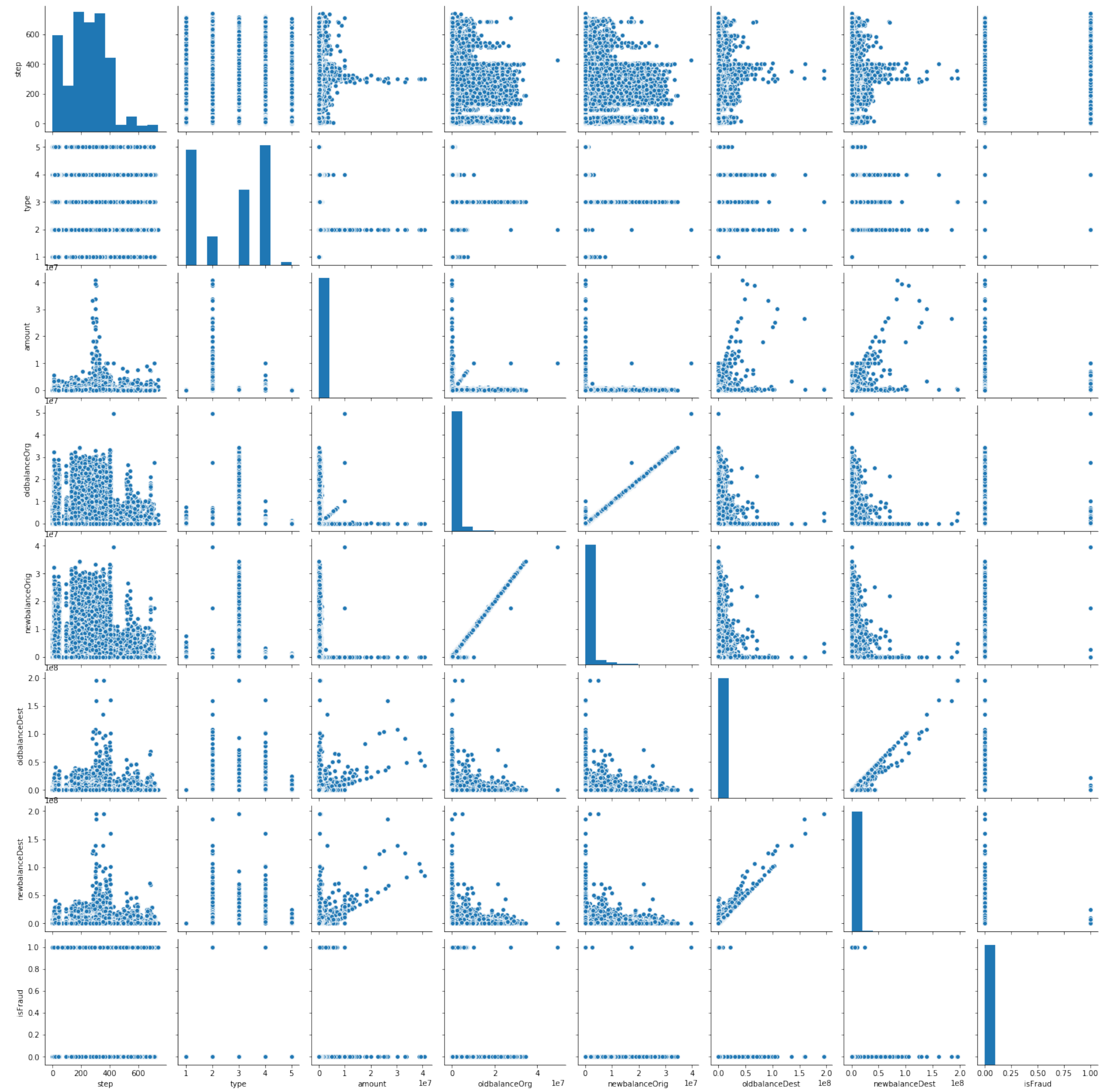
[12]:
```
fraud.isnull().sum() / len(fraud)
```

[12]:
```
step              0.0
type              0.0
amount            0.0
oldbalanceOrg     0.0
newbalanceOrig    0.0
oldbalanceDest    0.0
newbalanceDest    0.0
isFraud           0.0
dtype: float64
```

[13]:
```python
plt.figure(figsize=(12,10))
sns.heatmap(fraud.corr(), cmap='coolwarm',annot=True)
plt.show()
```



[14]:
```python
sns.pairplot(fraud.sample(100000))
plt.show()
```



Looking at the pair plots up above it seems that the items in type that are fraudulent are category 2 and 4 which is Transfers and Cash out

Looking at the number of Fraudulent Transactions

[15]:
```
FraudTransfer = fraud.loc[(fraud.isFraud == 1) & (fraud.type == 2)]
FraudCashout = fraud.loc[(fraud.isFraud == 1) & (fraud.type == 4)]
print(f" The number of Fraud Transfer is {len(FraudTransfer)}, and the number␣
↪of Fraud Cashout is {len(FraudCashout)}.")
```

```
The number of Fraud Transfer is 4097, and the number of Fraud Cashout is 4116.
```

[16]:
```
fraud['isFraud'].value_counts()
```

[16]:
```
0    6354407
1       8213
Name: isFraud, dtype: int64
```

[17]:
```
fraud = fraud.drop(fraud.query('isFraud == 0').sample(frac=.975).index)
```

[18]:
```
fraud['isFraud'].value_counts()
```

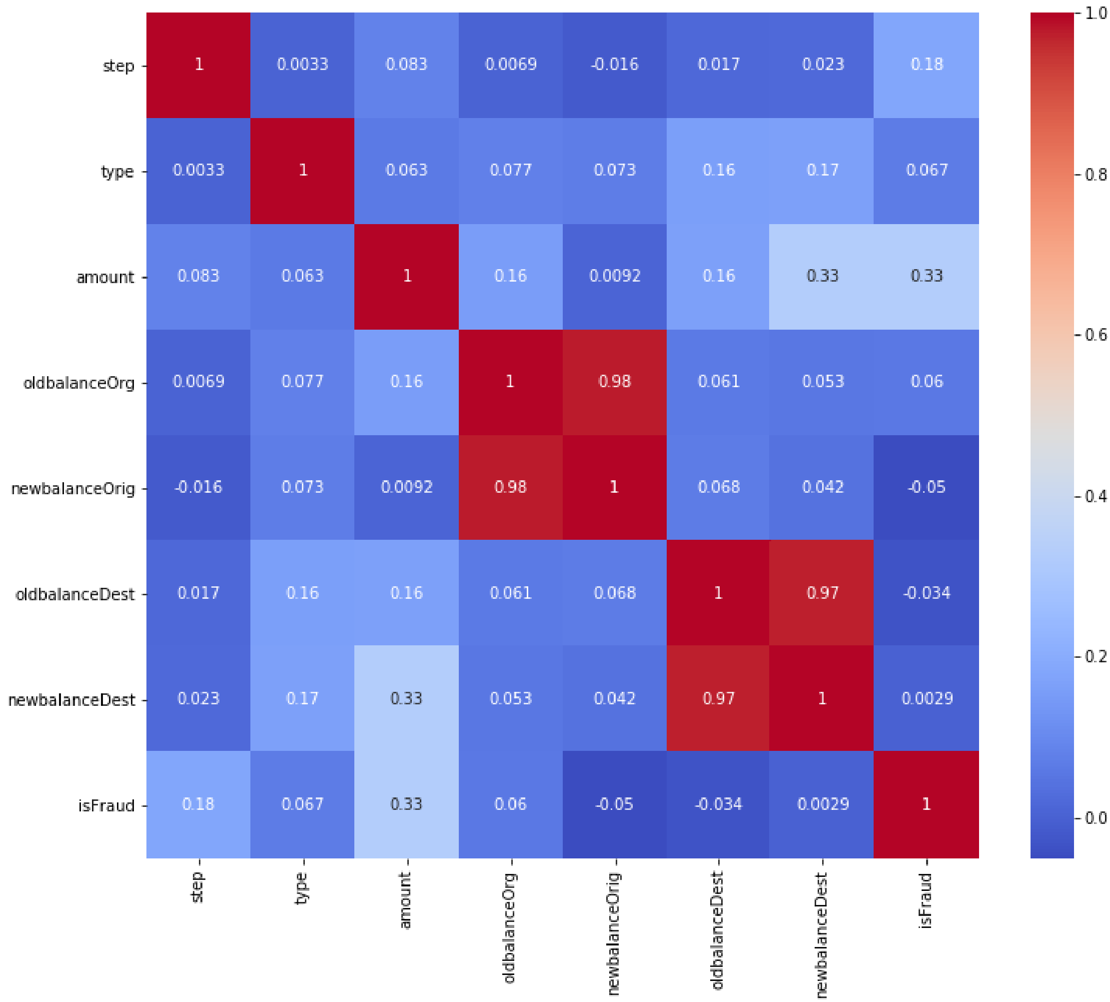
[18]:
```
0    158860
1      8213
Name: isFraud, dtype: int64
```

[19]:
```
sum(fraud.isFraud == 1)/len(fraud) * 100
```

[19]:
```
4.915815242438934
```

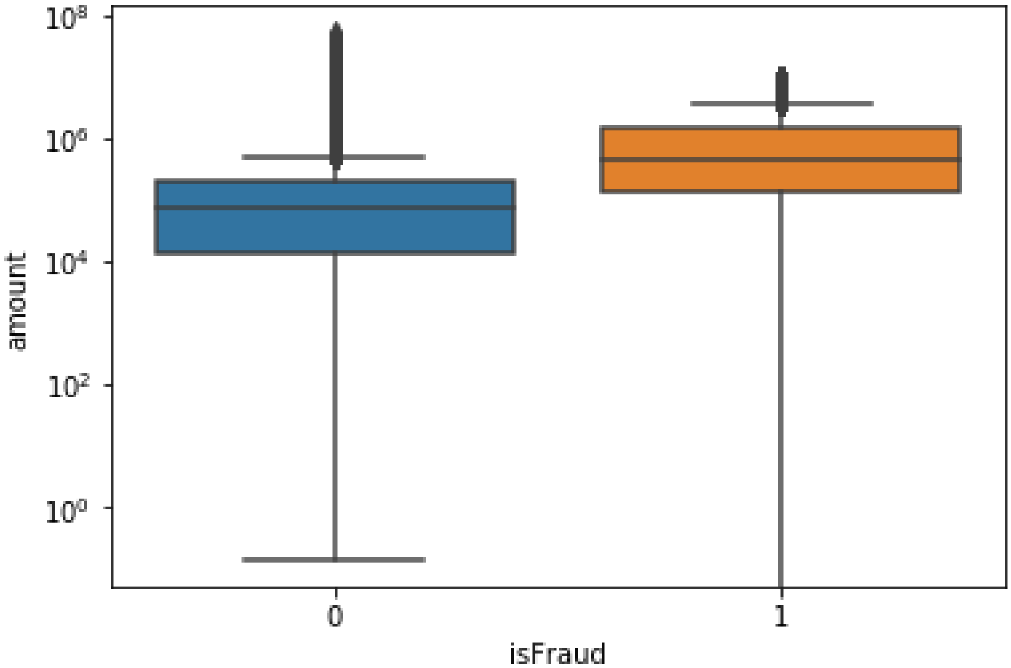
We decided to include ~5% of isFraud into our dataset. If we did not do this we ended up with approximately 99% accuracy for all of our data. This is because we had a 99.9999% zeros and only 0.0001% with ones. We did not want to scale anymore because we would introduce bias into our datset which would also become an issue. Overall this did improve the outcome of our models and gave us more variation in our results.

[20]:
```
plt.figure(figsize=(12,10))
sns.heatmap(fraud.corr(), cmap='coolwarm',annot=True)
plt.show()
```



[21]:

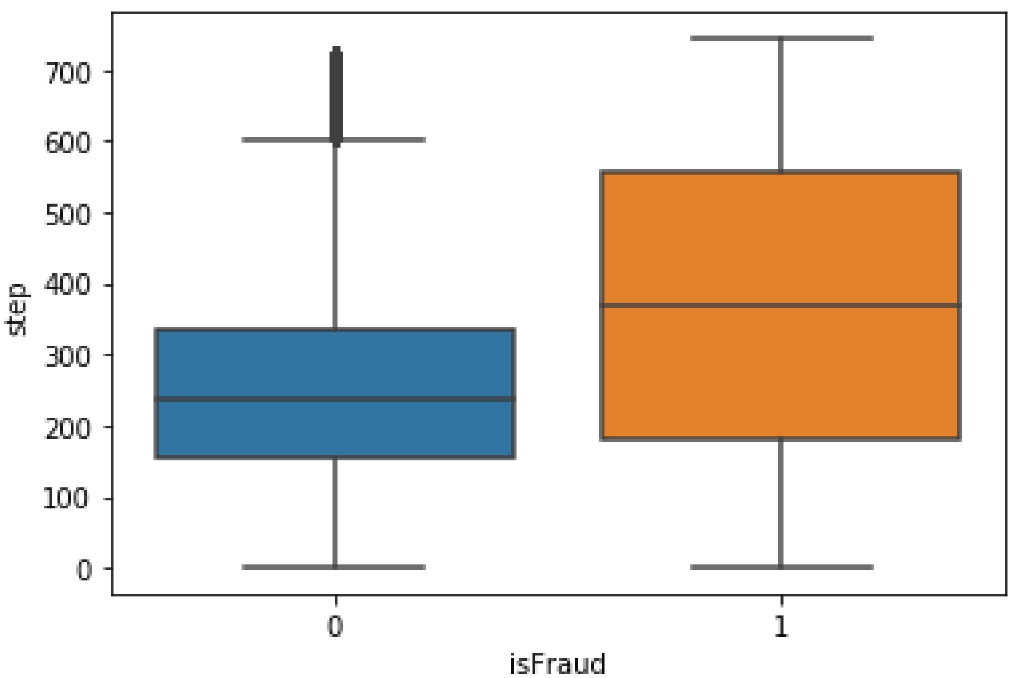
```
sns.boxplot(x = 'isFraud', y = 'amount', data = fraud).set_yscale('log')
```



It seems as though there is more fraud in a higher amount

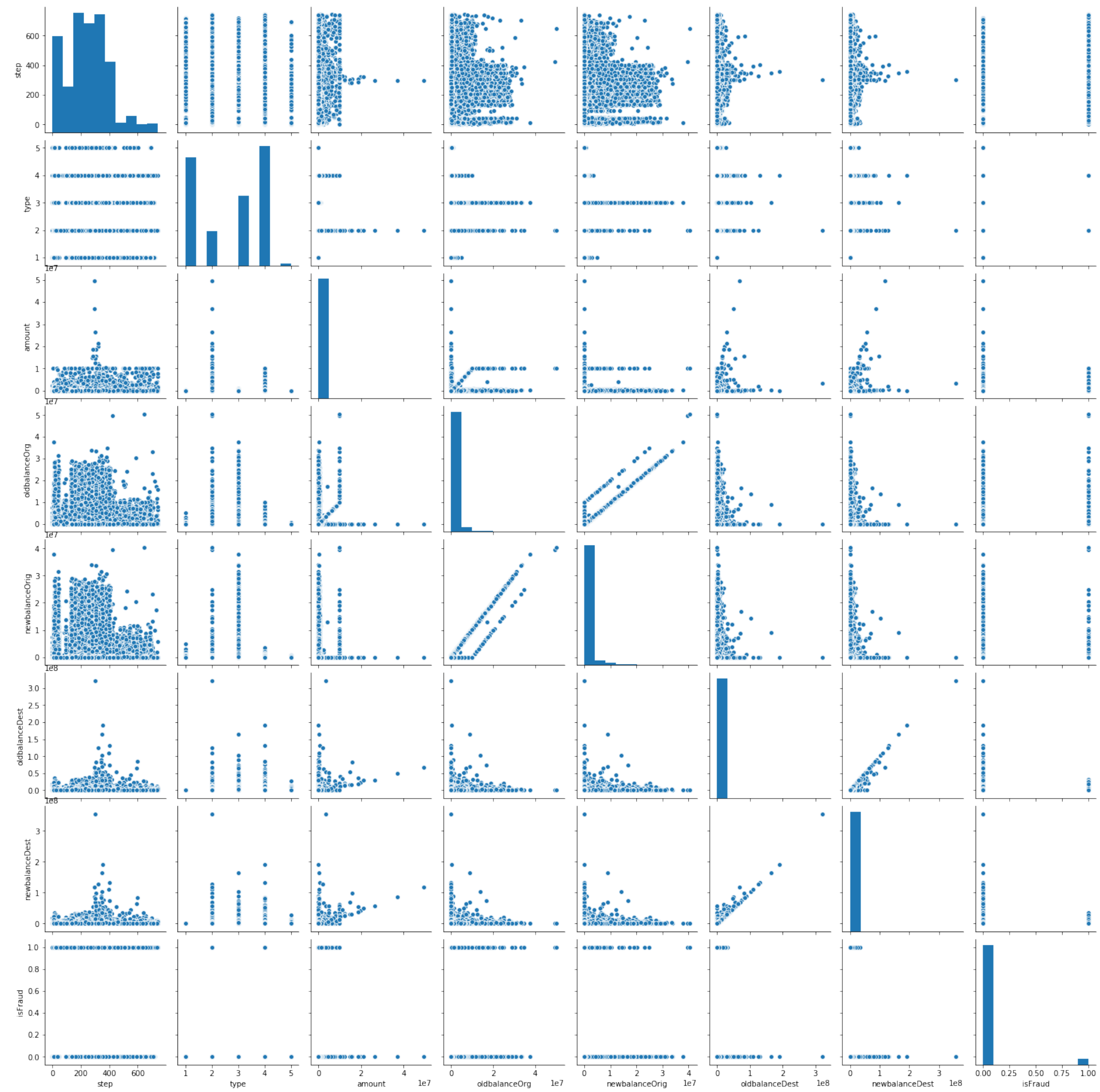
[22]:
```
sns.boxplot(x = 'isFraud', y = 'step',data = fraud)
```

[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1a246fded0>



It seems as though there is fraud with more steps overall

[79]:
```
sns.pairplot(fraud.sample(50000))
plt.show()
```



0.0.1 Logistic Regression

[23]:
```
from sklearn.linear_model import LogisticRegression
fraud = fraud
y = fraud['isFraud']
```

```
X = fraud.drop('isFraud', axis=1)
rand_state = 1000

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=rand_state)

len(X_train)/len(X), X_train.shape, X_test.shape, y_train.shape,y_test.shape
```

[23]:
```
(0.7999976058369694, (133658, 7), (33415, 7), (133658,), (33415,))
```

[24]:
```
logistic = LogisticRegression(solver='lbfgs', max_iter =1000)
logistic.fit(X_train, y_train)
```

[24]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[25]:
```
y_pred_test_logistic = logistic.predict(X_test)
```

[26]:
```
from sklearn.metrics import confusion_matrix
def my_confusion_matrix(y, y_hat):
    cm = confusion_matrix(y, y_hat)
    TN, FP, FN, TP = cm[0,0], cm[0,1], cm[1,0], cm[1,1]
    accuracy = round((TP+TN) / (TP+ FP+ FN+ TN) ,5)
    precision = round( TP / (TP+FP),5)
    recall = round( TP / (TP+FN),5)
    cm_labled = pd.DataFrame(cm, index=['Actual : 0 ','Actual : 1'],␣
↪columns=['Predict : 0','Predict :1 '])
    print('\n')
    print('Accuracy = {}'.format(accuracy))
    print('Precision = {}'.format(precision))
    print('Recall = {}'.format(recall))
    print("-----------------------------------------")
    return cm_labled
```

[27]:
```
my_confusion_matrix(y_test,y_pred_test_logistic)
```

```
Accuracy = 0.97268
Precision = 0.72746
Recall = 0.67154
-----------------------------------------
```

[27]:
```
               Predict : 0  Predict :1
Actual : 0         31445          396
Actual : 1           517         1057
```

0.0.2 Cross Validation for Logistic Regression

[28]:
```
from sklearn.model_selection import cross_val_score
```

[29]:
```
accuracy5_logistic = cross_val_score(estimator = logistic, X = X_train, y =␣
 ↪y_train, cv = 5 , scoring="accuracy" )
accuracy5_logistic
```

[29]:
```
array([0.97272931, 0.97097112, 0.97145743, 0.97452396, 0.971606  ])
```

[30]:
```
accuracy10_logistic = cross_val_score(estimator = logistic, X = X_train, y =␣
 ↪y_train, cv = 10 , scoring="accuracy")
accuracy10_logistic
```

[30]:
```
array([0.9726919 , 0.9726919 , 0.97112075, 0.97112075, 0.97194374,
       0.97082149, 0.97366452, 0.97426306, 0.97194164, 0.97126824])
```

[31]:
```
round(accuracy5_logistic.mean(),5) , round(accuracy10_logistic.mean(),5)
```

[31]:
```
(0.97226, 0.97215)
```

0.0.3 Scaled Data for Logistic Regression Visual ans remainder of ML data

[32]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_sc = sc.fit_transform(X_train)
X_test_sc = sc.transform(X_test)
```

[80]:
```
logistic.fit(X_train_sc, y_train)
```

[80]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=1000,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[81]:
```
y_pred_test_logistic = logistic.predict(X_test_sc)
```

[82]:
```
my_confusion_matrix(y_test,y_pred_test_logistic)
```

```
Accuracy = 0.97905
```

```
Precision = 0.97091
Recall = 0.57243
----------------------------------------
```

[82]:

```
                 Predict : 0  Predict :1
 Actual : 0          31814           27
 Actual : 1            673          901
```

[88]:

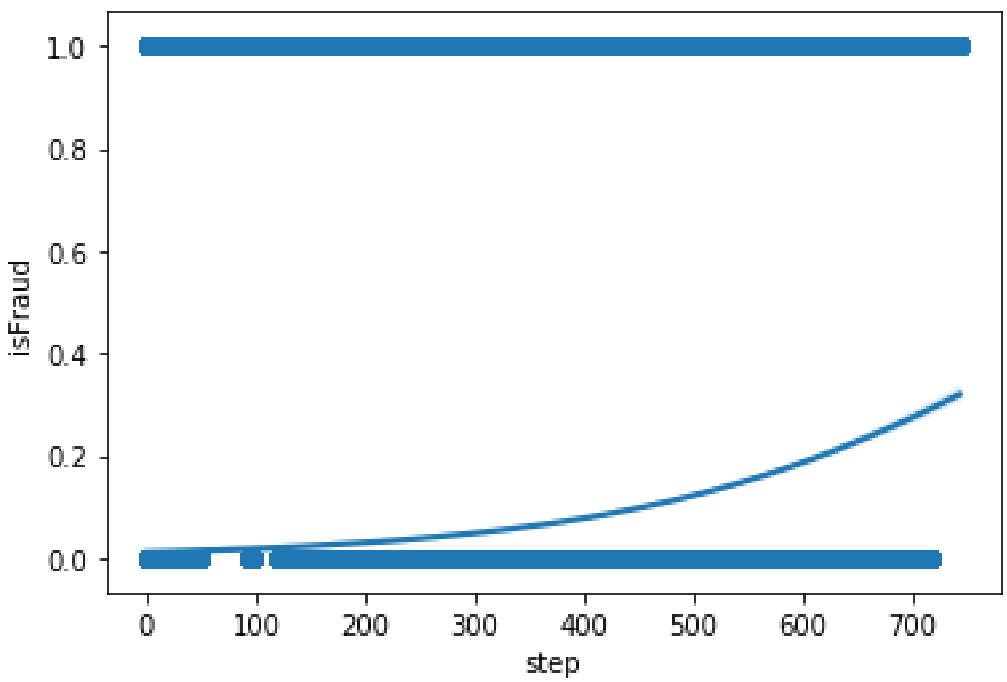
```
acc10_log_sc = cross_val_score(estimator = logistic, X = X_train_sc, y =␣
 ↪y_train, cv = 10 , scoring="accuracy")
round(acc10_log_sc.mean(),5)
```

[88]: 0.97812

[33]:

```
sns.regplot(x='step', y='isFraud', data=fraud, logistic=True)
```

[33]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2aea6650>



0.0.4 KNN Classification

[34]:

```
from sklearn.neighbors import KNeighborsClassifier
```

[35]:

```
KNN_classifier = KNeighborsClassifier(n_neighbors=5)
KNN_classifier.fit(X_train_sc, y_train)
```

[35]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

[36]:
```python
y_pred_test_knn = KNN_classifier.predict(X_test_sc)
```

Confusion Matrix

[37]:
```python
my_confusion_matrix(y_test,y_pred_test_knn)
```

```
Accuracy = 0.98821
Precision = 0.9351
Recall = 0.80559
-----------------------------------------
```

[37]:

	Predict : 0	Predict :1
Actual : 0	31753	88
Actual : 1	306	1268

[38]:
```python
accuracy_knn = cross_val_score(estimator = KNN_classifier, X = X_train_sc, y =
 ↪y_train, cv = 10 , scoring="accuracy" )
accuracy_knn
```

[38]:
```
array([0.98750561, 0.98668263, 0.98765524, 0.98645818, 0.98840341,
       0.98705671, 0.98900195, 0.98698189, 0.98683128, 0.98653199])
```

[39]:
```python
round(accuracy_knn.mean(),3)
```

[39]: 0.987

0.0.5 Choosing K

[40]:
```python
my_confusion_matrix(y=y_test, y_hat=KNeighborsClassifier(n_neighbors=5).
 ↪fit(X_train_sc,y_train).predict(X_test_sc))
```

```
Accuracy = 0.98821
Precision = 0.9351
Recall = 0.80559
-----------------------------------------
```

[40]:

	Predict : 0	Predict :1
Actual : 0	31753	88
Actual : 1	306	1268

[41]:
```
test_error_rate = []
CV_error_rate=[]
k=50

for i in range(1,k):
    KNN_i = KNeighborsClassifier(n_neighbors=i)
    KNN_i.fit(X_train_sc, y_train)
    MAE_i =  -1*cross_val_score(estimator = KNN_i, X = X_train_sc, y = y_train,␣
 ↪cv = 5 , scoring="neg_mean_absolute_error" )
    CV_error_rate.append(np.mean(MAE_i))
    test_error_rate.append(np.mean(y_test != KNN_i.predict(X_test_sc)) )

optimal_k = pd.DataFrame({'CV_error_rates': CV_error_rate, 'test_error_rates':
 ↪test_error_rate}, index=range(1,k))
```

[42]:
```
optimal_k.head(10)
```

[42]:

	CV_error_rates	test_error_rates
1	0.014126	0.013467
2	0.012914	0.011761
3	0.012682	0.011582
4	0.012936	0.011612
5	0.013078	0.011791
6	0.013377	0.011911
7	0.013647	0.012150
8	0.013894	0.012210
9	0.013991	0.012449
10	0.014223	0.012360

[43]:
```
KNN_opt = print(optimal_k[optimal_k.test_error_rates == optimal_k.
 ↪test_error_rates.min()])
KNN_opt
```

```
   CV_error_rates  test_error_rates
3        0.012682          0.011582
```

[44]:
```
my_confusion_matrix(y=y_test, y_hat=KNeighborsClassifier(n_neighbors=5).
 ↪fit(X_train_sc,y_train).predict(X_test_sc))
```

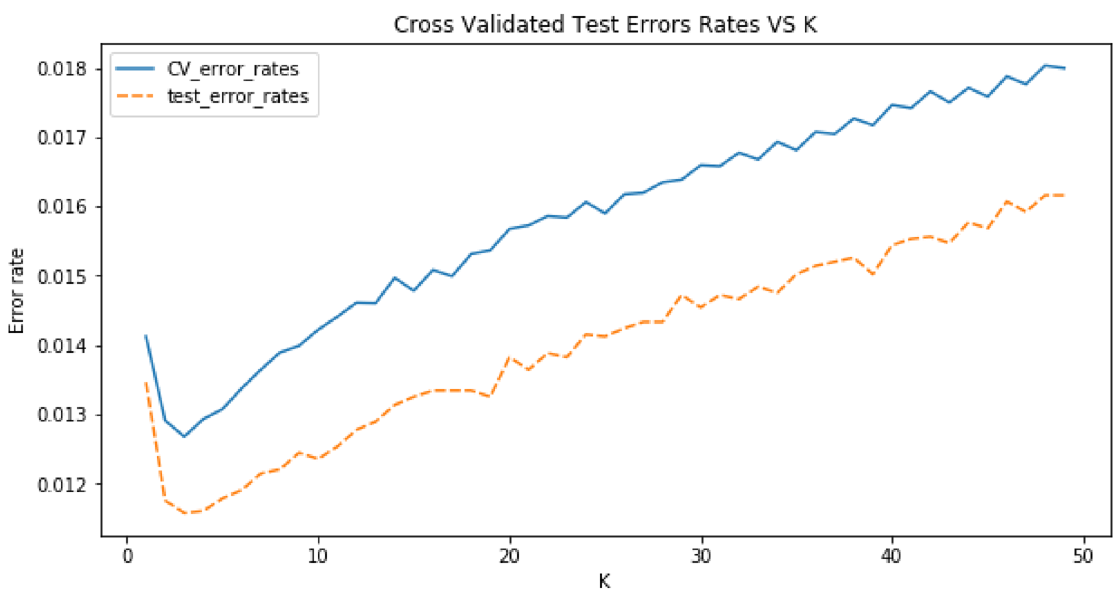
```
Accuracy = 0.98821
Precision = 0.9351
Recall = 0.80559
-----------------------------------------
```

[44]:
```
               Predict : 0  Predict :1
Actual : 0           31753           88
Actual : 1             306         1268
```

[84]:
```
KNN_classifier_optim = KNeighborsClassifier(n_neighbors=3)
accuracy_knn_opt = cross_val_score(estimator = KNN_classifier_optim, X =␣
 ↪X_train_sc, y = y_train, cv = 10 , scoring="accuracy" )
accuracy_knn_opt
```

[84]:
```
array([0.98847823, 0.98720634, 0.98750561, 0.98638336, 0.98780488,
       0.98720634, 0.98825378, 0.98720634, 0.98660681, 0.98772914])
```

[46]:
```
plt.figure(figsize=(10,5))
sns.lineplot(data=optimal_k)
plt.title('Cross Validated Test Errors Rates VS K')
plt.xlabel('K')
plt.ylabel('Error rate')
plt.show()
```



0.0.6 Support Vector Classification

[47]:
```
from sklearn.svm import SVC
SVM_classification = SVC(random_state = rand_state)
SVM_classification.fit(X_train_sc, y_train)
```

[47]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
```

```
    max_iter=-1, probability=False, random_state=1000, shrinking=True,
    tol=0.001, verbose=False)
```

[48]:
```python
y_pred_test = SVM_classification.predict(X_test_sc)
```

[49]:
```python
my_confusion_matrix(y_test,y_pred_test)
```

```
Accuracy = 0.98261
Precision = 0.98345
Recall = 0.64168
-----------------------------------------
```

[49]:

	Predict : 0	Predict :1
Actual : 0	31824	17
Actual : 1	564	1010

0.0.7 Cross Validation for SVC

[50]:
```python
SVM_classification = SVC(random_state = rand_state, gamma='scale')
accuracy_SVC = cross_val_score(estimator = SVM_classification, X = X_train_sc,␣
↪y = y_train, cv = 10 , scoring="accuracy" )
accuracy_SVC
```

[50]:
```
array([0.98152028, 0.98174473, 0.98181954, 0.98077211, 0.98234326,
       0.98159509, 0.98166991, 0.97964986, 0.98136925, 0.98077067])
```

0.0.8 Grid Search

[51]:
```python
param_grid = {'C': [0.1,1, 10], 'gamma': [1,0.1,0.01], 'kernel':␣
↪['rbf','linear']}
```

[52]:
```python
from sklearn.model_selection import GridSearchCV
```

[53]:
```python
grid_SVC = GridSearchCV(SVC(),param_grid,refit=True,verbose=0, cv=5)
```

[54]:
```python
grid_SVC.fit(X_train_sc,y_train)
```

[54]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                           class_weight=None, coef0=0.0,
                           decision_function_shape='ovr', degree=3,
                           gamma='scale', kernel='rbf', max_iter=-1,
                           probability=False, random_state=None, shrinking=True,
                           tol=0.001, verbose=False),
             iid='deprecated', n_jobs=None,
```

```
                 param_grid={'C': [0.1, 1, 10], 'gamma': [1, 0.1, 0.01],
                             'kernel': ['rbf', 'linear']},
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                 scoring=None, verbose=0)
```

[55]:
```
grid_SVC.best_params_
```

[55]:
```
{'C': 10, 'gamma': 1, 'kernel': 'rbf'}
```

[85]:
```
SVM_final = SVC(C=10, kernel='rbf', gamma=1, random_state=rand_state)
y_pred_test_optimized = SVM_final.fit(X_train_sc,y_train).predict(X_test_sc)
y_pred_test_optimized
```

[85]:
```
array([0, 0, 0, …, 0, 0, 0])
```

[86]:
```
my_confusion_matrix(y_test, y_pred_test_optimized)
```

```
Accuracy = 0.9895
Precision = 0.96361
Recall = 0.8075
-----------------------------------------
```

[86]:

	Predict : 0	Predict :1
Actual : 0	31793	48
Actual : 1	303	1271

[87]:
```
SVM_classification = SVC(random_state = rand_state, gamma=1,C =10 ,kernel␣
↪='rbf')
accuracy_SVC_opt = cross_val_score(estimator = SVM_classification, X =␣
↪X_train_sc, y = y_train, cv = 10 , scoring="accuracy" )
accuracy_SVC_opt
```

[87]:
```
array([0.98952566, 0.98802933, 0.98825378, 0.98817896, 0.98907676,
       0.98817896, 0.98885231, 0.98780488, 0.98660681, 0.98862701])
```

0.0.9 Random Forest

[59]:
```
from sklearn.ensemble import RandomForestClassifier

RF_classifier = RandomForestClassifier(n_estimators = 1000, criterion='gini',␣
↪max_features='sqrt', random_state=1000)
RF_classifier.fit(X_train, y_train)
```

[59]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='sqrt',
```

```
                            max_leaf_nodes=None, max_samples=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=1000,
                            n_jobs=None, oob_score=False, random_state=1000,
                            verbose=0, warm_start=False)
```

[60]:
```
y_pred_test_RF = RF_classifier.predict(X_test)
```

[61]:
```
my_confusion_matrix(y_test, y_pred_test_RF)
```

```
Accuracy = 0.99629
Precision = 0.97635
Recall = 0.94409
-----------------------------------------
```

[61]:

	Predict : 0	Predict :1
Actual : 0	31805	36
Actual : 1	88	1486

[62]:
```
accuracy_RF = cross_val_score(estimator = RF_classifier, X = X_train, y =␣
 ↪y_train, cv = 10 , scoring="accuracy" )
accuracy_RF
```

[62]:
```
array([0.99760587, 0.99543618, 0.99663325, 0.99648362, 0.99603471,
       0.99648362, 0.9959599 , 0.99566063, 0.99506173, 0.99678264])
```

[63]:
```
round(accuracy_RF.mean(),5)
```

[63]:
```
0.99621
```

0.0.10 Grid Search for RF

[64]:
```
param_grid_RF = {'max_depth': [5,10,20], 'criterion': ['entropy','gini'],␣
 ↪'max_features':['log2','sqrt']}
```

[65]:
```
grid_RF = GridSearchCV(RandomForestClassifier(n_estimators=100,␣
 ↪random_state=100),param_grid_RF,refit=True,verbose=0, cv=5)
```

[66]:
```
grid_RF.fit(X_train,y_train)
```

[66]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
```

```
                                                max_features='auto',
                                                max_leaf_nodes=None,
                                                max_samples=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=100, n_jobs=None,
                                                oob_score=False, random_state=100,
                                                verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['entropy', 'gini'],
                         'max_depth': [5, 10, 20],
                         'max_features': ['log2', 'sqrt']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

[67]:
```
grid_RF.best_params_
```

[67]:
```
{'criterion': 'gini', 'max_depth': 20, 'max_features': 'log2'}
```

[68]:
```
grid_predictions_RF = grid_RF.predict(X_test)
```

[69]:
```
my_confusion_matrix(y_test,grid_predictions_RF)
```

```
Accuracy = 0.99623
Precision = 0.97694
Recall = 0.94219
-----------------------------------------
```

[69]:

	Predict : 0	Predict :1
Actual : 0	31806	35
Actual : 1	91	1483

[70]:
```
RF_classifier = RandomForestClassifier(n_estimators = 1000, max_depth = 20,␣
↪criterion='gini', max_features='log2', random_state=1000)
accuracy_RF_opt = cross_val_score(estimator = RF_classifier, X = X_train, y =␣
↪y_train, cv = 10 , scoring="accuracy" )
accuracy_RF_opt
```

[70]:
```
array([0.9973066 , 0.99528655, 0.99648362, 0.99625917, 0.99581026,
       0.9964088 , 0.99603471, 0.995511  , 0.99483726, 0.99670782])
```

0.0.11 Feature Space Importance

[71]:
```
features= list(X_train.columns)
features
```

[71]:
```
['step',
 'type',
 'amount',
 'oldbalanceOrg',
 'newbalanceOrig',
 'oldbalanceDest',
 'newbalanceDest']
```

[72]:
```
RF_classifier = RandomForestClassifier(n_estimators = 100, max_features='log2',␣
↪max_depth=20, criterion='entropy')
RF_classifier.fit(X_train, y_train)
```

[72]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=20, max_features='log2',
                       max_leaf_nodes=None, max_samples=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=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[73]:
```
importances= RF_classifier.feature_importances_
importances
```

[73]:
```
array([0.10664627, 0.12775313, 0.18895597, 0.28874261, 0.13652339,
       0.0581806 , 0.09319804])
```

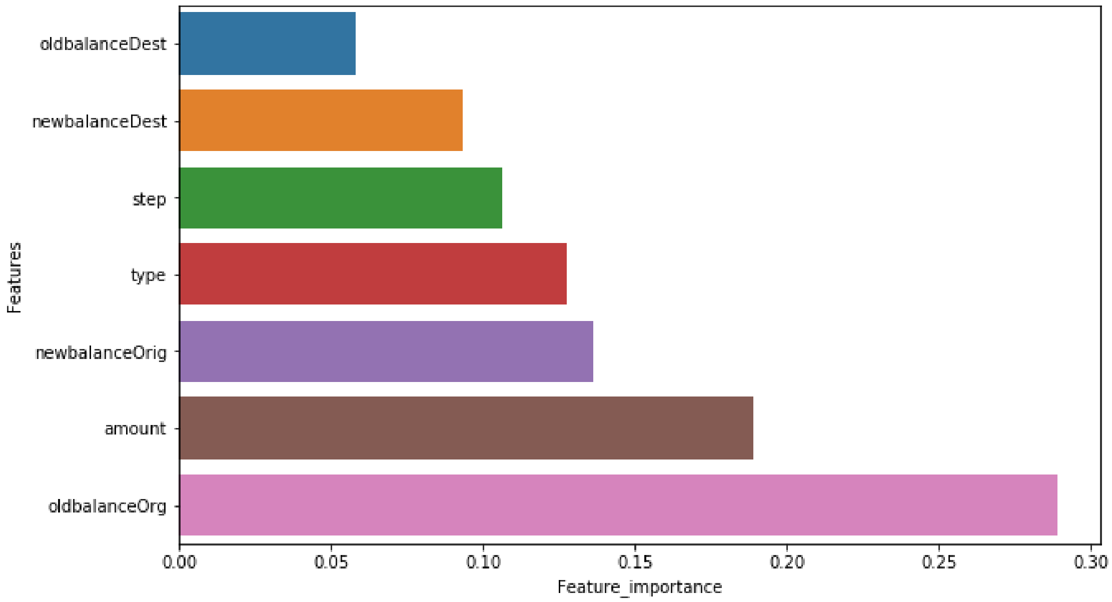
[74]:
```
FIM = pd.DataFrame({'Features': features , 'Feature_importance':importances})
FIM=FIM.sort_values(by=['Feature_importance'])
FIM
```

[74]:

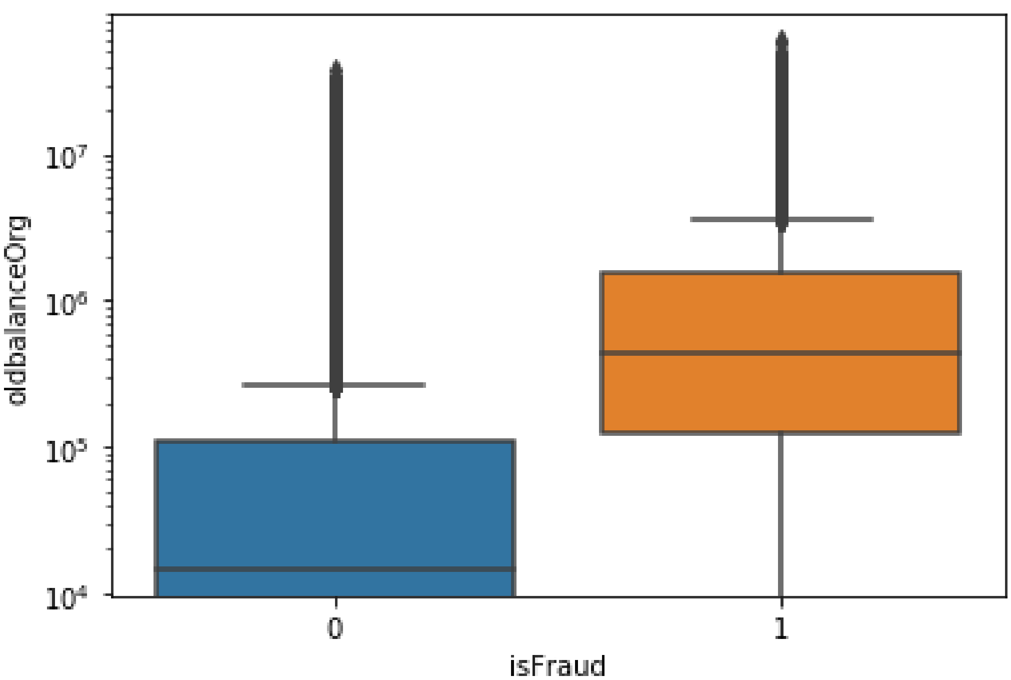
	Features	Feature_importance
5	oldbalanceDest	0.058181
6	newbalanceDest	0.093198
0	step	0.106646
1	type	0.127753
4	newbalanceOrig	0.136523
2	amount	0.188956
3	oldbalanceOrg	0.288743

[75]:
```
plt.figure(figsize=(10,6))
sns.barplot(y='Features', x='Feature_importance', data=FIM)
```

[75]: <matplotlib.axes._subplots.AxesSubplot at 0x1a23785710>



[76]:
```
sns.boxplot(x = 'isFraud', y = 'oldbalanceOrg',data = fraud).set_yscale('log')
```



[89]:
```
models = pd.DataFrame({
    'Model': ['Logistic Regression', 'KNearestNeighbors', 'Support Vector␣
↪Machine',
              'Random Forest'],
    'Score': [round(accuracy10_logistic.mean(),5), round(accuracy_knn_opt.
↪mean(),5), round(accuracy_SVC_opt.mean(),5),
              round(accuracy_RF_opt.mean(),5)]})
models = models.sort_values(by='Score', ascending=False)
models
```

[89]:

	Model	Score
3	Random Forest	0.99606
2	Support Vector Machine	0.98831
1	KNearestNeighbors	0.98744
0	Logistic Regression	0.97215

[78]:
```
print(optimal_k[optimal_k.test_error_rates == optimal_k.test_error_rates.
↪min()]),grid_SVC.best_params_, grid_RF.best_params_
```

```
   CV_error_rates  test_error_rates
3        0.012682          0.011582
```

[78]:
```
(None,
 {'C': 10, 'gamma': 1, 'kernel': 'rbf'},
 {'criterion': 'gini', 'max_depth': 20, 'max_features': 'log2'})
```

1 Conclusion

From the results above, the best model to run in this scenario is Random Forest When we ran the dataset without having 5% of the dataset being ‘isFraud’ = 1, Logistic Regression did well. However, when we changed to the 5%, Logistic Regression was the worst of the machine learning methods. This is a very interesting observation and one that would be intersting to look more in to. The way that I like to think about Random Forest is the follow (This is as explained by Adele Cutler): Suppose you have 100 students in a class that need to submit a single 100-question test together. However, each student only knows 50 of the questions correctly. Random Forest groups the students together in various ways to take the test and find the groups with the most correct answers. Random Forest is also a good method because it decorrelates trees and reduces the variance.