In [41]:

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

In [2]:

```
data = pd.read_csv("train.csv")
```

In [3]:

```
data.head(20)
```

Out[3]:

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155
1	1	TRANSFER	181.00	C1305486145	181.00	0.00	C553264065
2	1	CASH_OUT	181.00	C840083671	181.00	0.00	C38997010
3	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703
4	1	PAYMENT	7107.77	C154988899	183195.00	176087.23	M408069119
5	1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M633326333
6	1	PAYMENT	4024.36	C1265012928	2671.00	0.00	M1176932104
7	1	DEBIT	5337.77	C712410124	41720.00	36382.23	C195600860
8	1	DEBIT	9644.94	C1900366749	4465.00	0.00	C997608398
9	1	PAYMENT	3099.97	C249177573	20771.00	17671.03	M2096539129
10	1	PAYMENT	2560.74	C1648232591	5070.00	2509.26	M972865270
11	1	PAYMENT	11633.76	C1716932897	10127.00	0.00	M801569151
12	1	PAYMENT	4098.78	C1026483832	503264.00	499165.22	M1635378213
13	1	CASH_OUT	229133.94	C905080434	15325.00	0.00	C476402209
14	1	PAYMENT	1563.82	C761750706	450.00	0.00	M1731217984
15	1	PAYMENT	1157.86	C1237762639	21156.00	19998.14	M1877062907
16	1	PAYMENT	671.64	C2033524545	15123.00	14451.36	M473053293
17	1	TRANSFER	215310.30	C1670993182	705.00	0.00	C1100439041
18	1	PAYMENT	1373.43	C20804602	13854.00	12480.57	M1344519051
19	1	DEBIT	9302.79	C1566511282	11299.00	1996.21	C1973538135

In [4]:

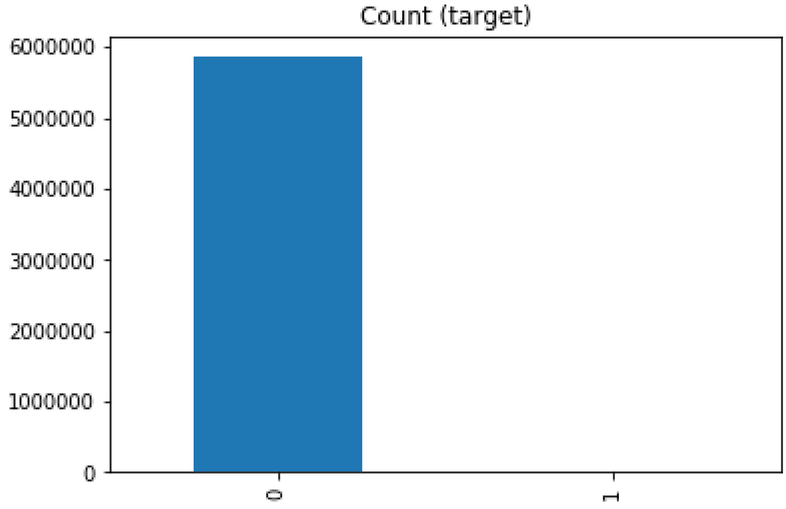
```
data.shape
```

Out[4]:

```
(5862620, 11)
```

In [5]:

```
data.isFraud.value_counts().plot(kind='bar', title='Count (target)');
```



In [6]:

```
seriesObj = data.apply(lambda x: True if x['isFraud'] == 1 else False , axis=1)
numOfRows = len(seriesObj[seriesObj == True].index)

print('isFraud count : ', numOfRows)
```

```
isFraud count :  7513
```

In [7]:

```
FraudPercent = (numOfRows / data.shape[0]) * 100
FraudPercent
```

Out[7]:

```
0.1281508949923413
```

In [8]:

```
#Data is very imbalanced, applying undersampling method

# Class count
count_class_0, count_class_1 = data.isFraud.value_counts()

# Divide by class
df_class_noFraud = data[data['isFraud'] == 0]
df_class_yesFraud = data[data['isFraud'] == 1]
```

In [9]:

```
count_class_1
```

Out[9]:

```
7513
```

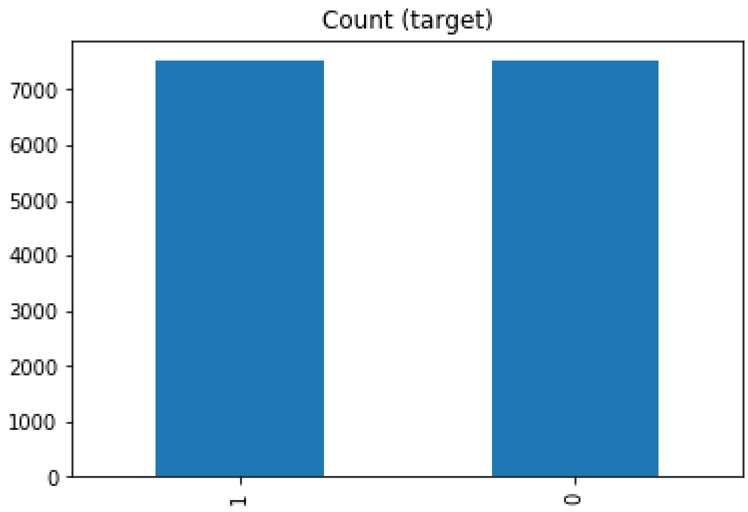
In [10]:

```
#Undersampling the noFraud data set to the same number of yesFraud
df_class_noFraud_under = df_class_noFraud.sample(count_class_1)
df_under = pd.concat([df_class_noFraud_under, df_class_yesFraud], axis=0)

print('Random under-sampling:')
print(df_under.isFraud.value_counts())

df_under.isFraud.value_counts().plot(kind='bar', title='Count (target)');
```

```
Random under-sampling:
1    7513
0    7513
Name: isFraud, dtype: int64
```



In [11]:

```
#One-hot enconding the "type" column

dfDummies = pd.get_dummies(df_under['type'],prefix=['type'])
dfDummies.head(5)
```

Out[11]:

	['type']_CASH_IN	['type']_CASH_OUT	['type']_DEBIT	['type']_PAYMENT	['type']_TRANSF
469627	1	0	0	0	
1891561	1	0	0	0	
2979505	0	0	0	1	
3531630	0	1	0	0	
1067317	1	0	0	0	

In [12]:

```
df_under = pd.concat([df_under, dfDummies], axis=1)
```

In [13]:

```
del df_under['nameDest']
del df_under['nameOrig']
del df_under['type']
```

In [14]:

```
df_under.head(20)
```

Out[14]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isF
469627	20	177402.91	254045.13	431448.04	5048457.83	4871054.93	
1891561	181	339127.75	20435.00	359562.75	86272.86	0.00	
2979505	250	7949.23	60376.05	52426.82	0.00	0.00	
3531630	282	218133.52	30138.00	0.00	1250666.69	1468800.21	
1067317	131	280577.00	6706479.63	6987056.63	344521.69	63944.69	
5574548	493	16449.70	0.00	0.00	0.00	0.00	
12464	7	4225.47	232.00	0.00	0.00	0.00	
5409625	403	3527.69	59116.77	55589.09	0.00	0.00	
5299167	399	18110.14	0.00	0.00	386075.43	404185.57	
5242530	397	14286.80	0.00	0.00	0.00	0.00	
4866133	372	12256.32	90508.00	102764.32	241931.40	229675.08	
2515941	212	369853.23	0.00	0.00	1366201.34	1736054.57	
3449098	279	111909.01	41117.00	153026.01	0.00	0.00	
4150114	325	497.24	297687.11	297189.87	0.00	0.00	
5621605	524	7550.01	0.00	0.00	0.00	0.00	
5858875	714	90768.99	20676.00	111444.99	113212.32	22443.33	
3652002	297	95910.20	85018.00	180928.20	2391656.23	2295746.03	
5719932	587	397067.20	59845.00	456912.20	11323.75	0.00	
4891472	373	278480.33	0.00	0.00	9579852.08	9858332.41	
515220	22	6041.13	50215.00	44173.87	0.00	0.00	

In [15]:

```
#Now the dataset is ready to apply the random forest algorithm

# Use numpy to convert to arrays
target = np.array(df_under['isFraud'])
# Remove the target from the features
# axis 1 refers to the columns
features = df_under.drop('isFraud', axis = 1)
# Saving feature names for later use
feature_list = list(features.columns)
# Convert to numpy array
features = np.array(features)
```

In [16]:

```
# Using Skicit-Learn to split data into training and testing sets
# Split the data into training and testing sets
train_features, test_features, train_labels, test_labels = train_test_split(features, targe
```

In [17]:

```
print('Training Features Shape:', train_features.shape)
print('Training Labels Shape:', train_labels.shape)
print('Testing Features Shape:', test_features.shape)
print('Testing Labels Shape:', test_labels.shape)
```

```
Training Features Shape: (11269, 12)
Training Labels Shape: (11269,)
Testing Features Shape: (3757, 12)
Testing Labels Shape: (3757,)
```

In [18]:

```
# Instantiate model with 1000 decision trees
rf = RandomForestRegressor(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(train_features, train_labels);
```

In [42]:

```
# Use the forest's predict method on the test data
predictions = rf.predict(test_features)
```

In [56]:

```
print('Expected performance')
print('Mean Absolute Error:', metrics.mean_absolute_error(test_labels, predictions))
print('Mean Squared Error:', metrics.mean_squared_error(test_labels, predictions))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(test_labels, predictic
```

```
Expected performance
Mean Absolute Error: 0.012365717327655045
Mean Squared Error: 0.006284042587170615
Root Mean Squared Error: 0.07927195334524446
```

In [44]:

```
teste = pd.read_csv("test.csv")
ids = teste['transaction_id']
teste.head(5)
```

Out[44]:

	transaction_id	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	
0	0	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M
1	1	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	I
2	2	1	PAYMENT	5086.48	C598357562	0.00	0.00	M
3	3	1	CASH_OUT	56953.90	C1570470538	1942.02	0.00	'
4	4	1	DEBIT	4874.49	C811207775	153.00	0.00	C

In [35]:

```
#One-hot enconding the "type" column

dfDummies = pd.get_dummies(teste['type'],prefix=['type'])
dfDummies.head(5)
```

Out[35]:

	['type']_CASH_IN	['type']_CASH_OUT	['type']_DEBIT	['type']_PAYMENT	['type']_TRANSFER
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	1	0
3	0	1	0	0	0
4	0	0	1	0	0

In [36]:

```
teste = pd.concat([teste, dfDummies], axis=1)
del teste['nameDest']
del teste['nameOrig']
del teste['type']
del teste['transaction_id']
teste.head(5)
```

Out[36]:

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFlaggedF
0	1	1864.28	21249.00	19384.72	0.0	0.00	
1	1	7817.71	53860.00	46042.29	0.0	0.00	
2	1	5086.48	0.00	0.00	0.0	0.00	
3	1	56953.90	1942.02	0.00	70253.0	64106.18	
4	1	4874.49	153.00	0.00	253104.0	0.00	

In [37]:

```
# Saving feature names for later use
feature_list_test = list(teste)
# Convert to numpy array
features_test = np.array(teste)
```

In [38]:

```
predictions_test = rf.predict(features_test)
```

In [39]:

```
predictions_test
```

Out[39]:

```
array([0.007, 0.004, 0.017, ..., 1.   , 1.   , 0.912])
```

In [46]:

```
len(predictions_test)
```

Out[46]:

```
500000
```

In [47]:

```
data_final = {'transaction_id':ids, 'isFraudprob':predictions_test}
```

In [49]:

```
df_final = pd.DataFrame(data_final)
```

In [51]:

```
df_final.head()
```

Out[51]:

	transaction_id	isFraudprob
0	0	0.007
1	1	0.004
2	2	0.017
3	3	0.788
4	4	0.044

In [53]:

```
export_csv = df_final.to_csv (r'probabilities.csv', index = None, header=True)
```