SMARTBRIDGE
Let's Bridge the Gap
a Veranda Enterprise

Smart Internz

Data Collection and Preprocessing Phase

Date	30 April 2024
Team ID	738286
Project Title	Online Payments Fraud Detection Using Machine Learning
Maximum Marks	6 Marks

Data Collection:

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So this section allows you to download the required dataset.

Data Set :

In this project we have used PS_20174392719_1491204439457_logs.csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download dataset

Visualizing And Analyzing Data:

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Importing The Libraries:

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

Importing Libraries¶

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.svm import SVC
import xgboost as xgb
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
```

Read The Dataset:

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

```
# Reading the csv data
df = pd.read_csv(r'C:\Users\user\Desktop\PS_20174392719_1491204439457_logs.csv')
```

```
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	0	0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0	0
2	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0	0
3	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	M573487274	0.00	0.00	0	0
4	1	PAYMENT	7107.77	C154988899	183195.00	176087.23	M408069119	0.00	0.00	0	0
...	...	...	...	...	...	...	...	...	...	...	...
2425	95	CASH_OUT	56745.14	C526144262	56745.14	0.00	C79051264	51433.88	108179.02	1	0
2426	95	TRANSFER	33676.59	C732111322	33676.59	0.00	C1140210295	0.00	0.00	1	0
2427	95	CASH_OUT	33676.59	C1000086512	33676.59	0.00	C1759363094	0.00	33676.59	1	0
2428	95	TRANSFER	87999.25	C927181710	87999.25	0.00	C757947873	0.00	0.00	1	0
2429	95	CASH_OUT	87999.25	C409531429	87999.25	0.00	C1827219533	0.00	87999.25	1	0

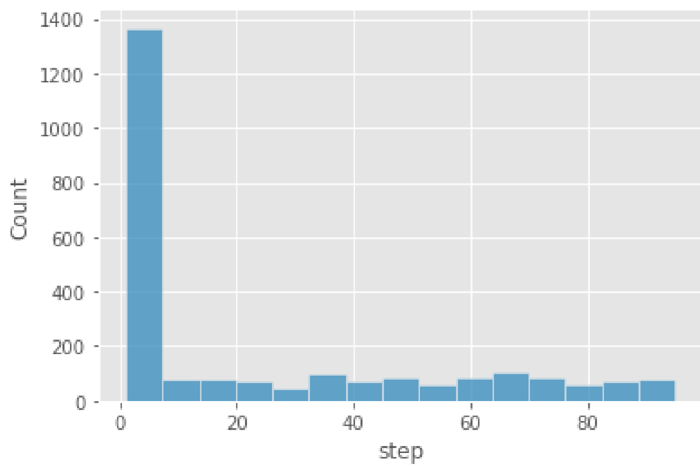
2430 rows × 11 columns

Univariate Analysis :

In simple words, univariate analysis is understanding the data with a single feature. Here I have displayed the graph such as histplot .

```
#step
sns.histplot(data=df,x='step')
```

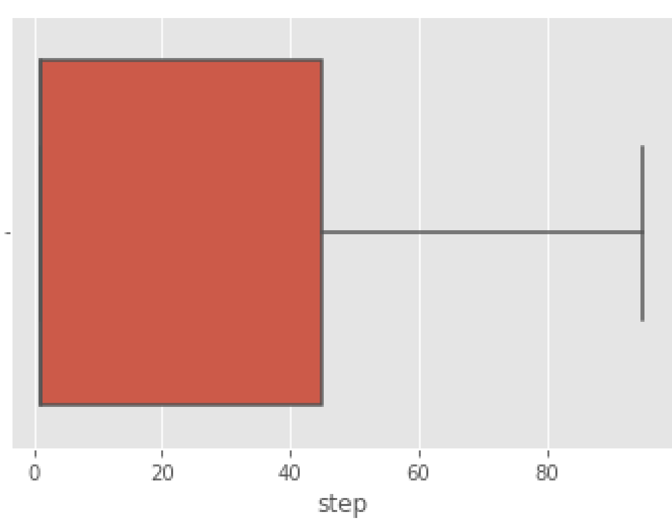
```
<AxesSubplot:xlabel='step', ylabel='Count'>
```



The distribution of one or more variables is represented by a histogram, a traditional visualisation tool, by counting the number of observations that fall within.

```
sns.boxplot(data=df,x='step')
```

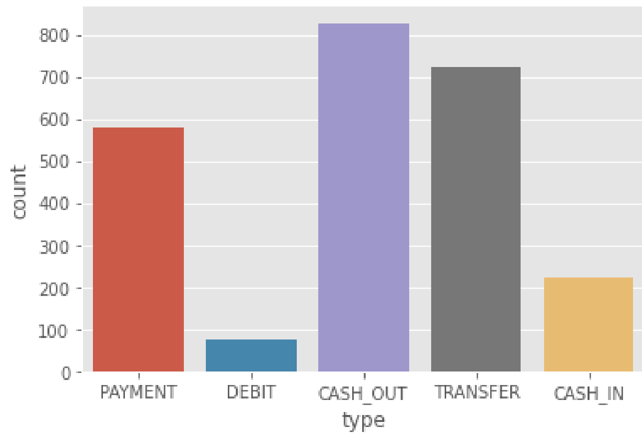
```
<AxesSubplot:xlabel='step'>
```



Here, the relationship between the step attribute and the boxplot is visualised.

```
#type
sns.countplot(data=df,x='type')
```

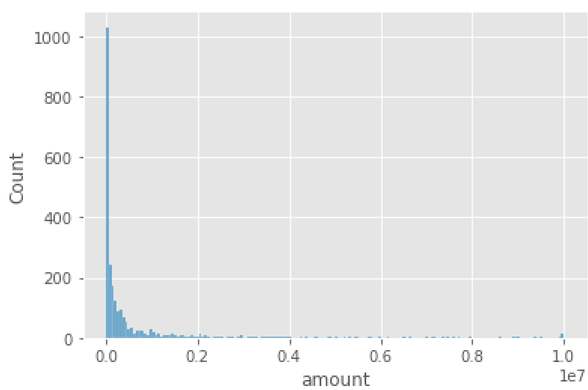
```
<AxesSubplot:xlabel='type', ylabel='count'>
```



Here, the counts of observations in the type attribute of the dataset will be displayed using a countplot.

```
#amount
sns.histplot(data=df,x='amount')
```

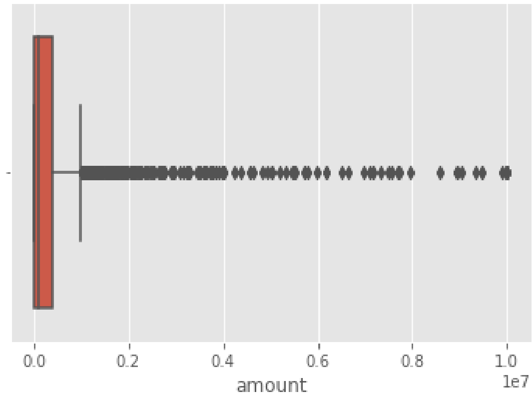
```
<AxesSubplot:xlabel='amount', ylabel='Count'>
```



By creating bins along the data's range and then drawing bars to reflect the number of observations that fall within the amount attribute in the dataset.

```
#amount
sns.boxplot(data=df,x='amount')
```

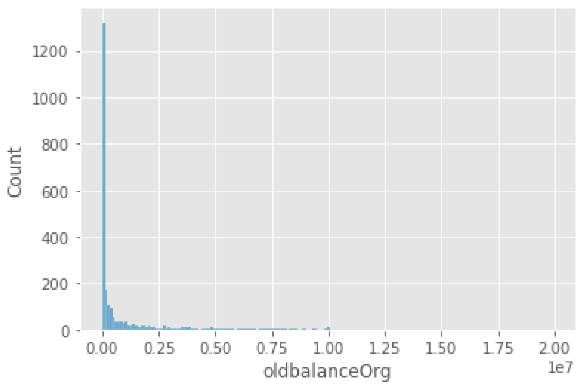
```
<AxesSubplot:xlabel='amount'>
```



Here, the relationship between the amount attribute and the boxplot is visualised.

```
#oldbalanceOrg
sns.histplot(data=df,x='oldbalanceOrg')
```

```
<AxesSubplot:xlabel='oldbalanceOrg', ylabel='Count'>
```



By creating bins along the data's range and then drawing bars to reflect the number of observations that fall within the oldbalanceOrg attribute in the dataset.

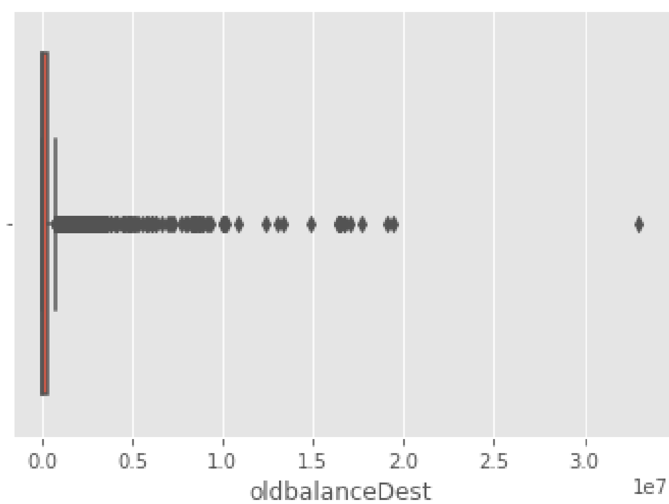
```
#nameDest
df['nameDest'].value_counts()
```

```
C1590550415    25
C985934102     22
C564160838     19
C451111351     17
C1023714065    15
               ..
M1113829504     1
M936219350      1
M178401052      1
M1888639813     1
C757947873      1
Name: nameDest, Length: 1870, dtype: int64
```

utilising the value counts() function here to determine how many times the nameDest column appears.

```
#oldbalanceDest
sns.boxplot(data=df,x='oldbalanceDest')
```

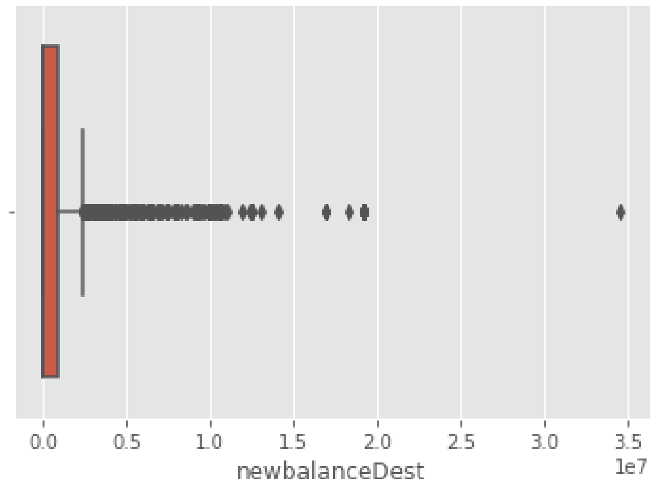
```
<AxesSubplot:xlabel='oldbalanceDest'>
```



Here, the relationship between the oldbalanceDest attribute and the boxplot is visualised.

```
#newbalanceDest
sns.boxplot(data=df,x='newbalanceDest')
```

```
<AxesSubplot:xlabel='newbalanceDest'>
```



Here, the relationship between the newbalanceDest attribute and the boxplot is visualised.

```
#isFraud:
sns.countplot(data=df,x='isFraud')
```

```
<AxesSubplot:xlabel='isFraud', ylabel='count'>
```

using the countplot approach here to count the number of instances in the dataset's target isFraud column.

```
df['isFraud'].value_counts()
```

```
0    1288
1    1142
Name: isFraud, dtype: int64
```

Here, we're using the value counts method to figure out how many classes there are in the dataset's target isFraud column.

```
df.loc[df['isFraud']==0,'isFraud'] = 'is not Fraud'
df.loc[df['isFraud']==1,'isFraud'] = 'is Fraud'
```

```
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M1979787155	0.00	0.00	is not Fraud
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	is not Fraud
2	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	is not Fraud
3	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	M573487274	0.00	0.00	is not Fraud
4	1	PAYMENT	7107.77	C154988899	183195.00	176087.23	M408069119	0.00	0.00	is not Fraud
...	...	...	...	...	...	...	...	...	...	...
2425	95	CASH_OUT	56745.14	C526144262	56745.14	0.00	C79051264	51433.88	108179.02	is Fraud
2426	95	TRANSFER	33676.59	C732111322	33676.59	0.00	C1140210295	0.00	0.00	is Fraud
2427	95	CASH_OUT	33676.59	C1000086512	33676.59	0.00	C1759363094	0.00	33676.59	is Fraud
2428	95	TRANSFER	87999.25	C927181710	87999.25	0.00	C757947873	0.00	0.00	is Fraud
2429	95	CASH_OUT	87999.25	C409531429	87999.25	0.00	C1827219533	0.00	87999.25	is Fraud

2430 rows × 10 columns

converting 0-means: is not fraud and 1-means: is fraud using the loc technique here

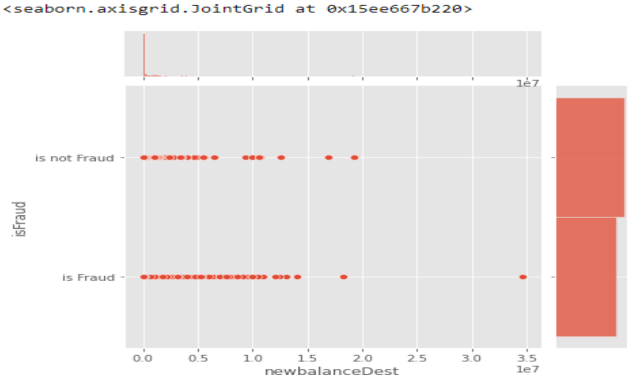
Bivariate Analysis

To find the relation between two features we use bivariate analysis. Here we are visualising the relationship between newbalanceDest and isFraud.

jointplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.jointplot(data=df,x='newbalanceDest',y='isFraud')
```

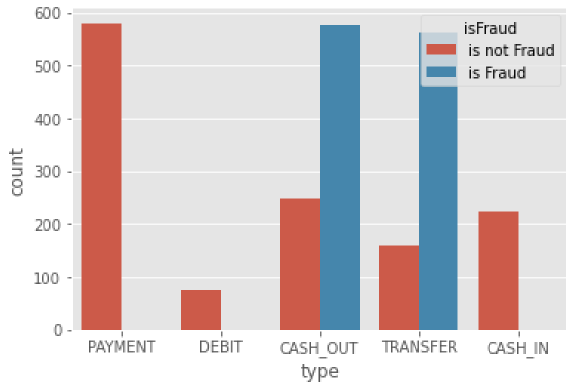
<seaborn.axisgrid.JointGrid at 0x15ee667b220>



Here we are visualising the relationship between type and isFraud.countplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.countplot(data=df,x='type',hue='isFraud')
```

```
<AxesSubplot:xlabel='type', ylabel='count'>
```



Here we are visualising the relationship between isFraud and step.boxtplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.boxplot(data=df,x='isFraud',y='step')
```

```
<AxesSubplot:xlabel='isFraud', ylabel='step'>
```

Here we are visualising the relationship between isFraud and amount.boxtplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.boxplot(data=df,x='isFraud',y='amount')
```

```
<AxesSubplot:xlabel='isFraud', ylabel='amount'>
```

Here we are visualising the relationship between isFraud and oldbalanceOrg. boxtplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.boxplot(data=df,x='isFraud',y='oldbalanceOrg')
```

```
<AxesSubplot:xlabel='isFraud', ylabel='oldbalanceOrg'>
```

Here we are visualising the relationship between isFraud and newbalanceOrig. boxtplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

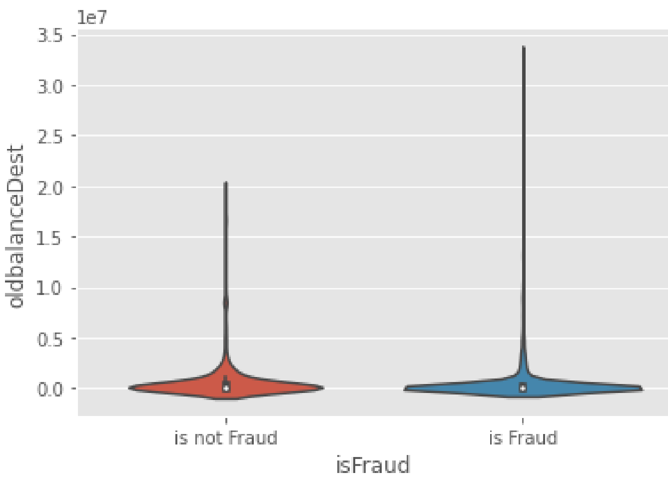
```
sns.boxplot(data=df,x='isFraud',y='newbalanceOrig')
```

```
<AxesSubplot:xlabel='isFraud', ylabel='newbalanceOrig'>
```

Here we are visualising the relationship between isFraud and oldbalanceDest. violinplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.violinplot(data=df,x='isFraud',y='oldbalanceDest')
```

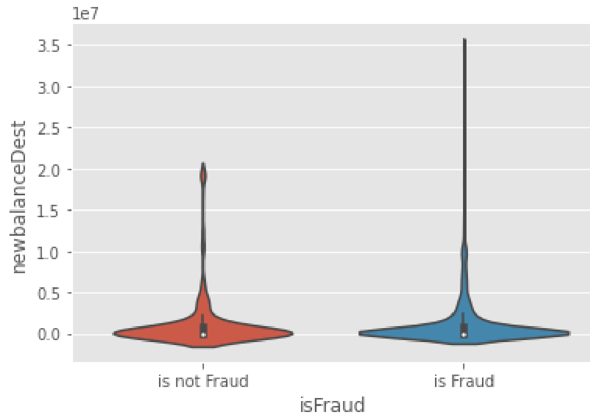
```
<AxesSubplot:xlabel='isFraud', ylabel='oldbalanceDest'>
```



Here we are visualising the relationship between isFraud and newbalanceDest. violinplot is used here. As a 1st parameter we are passing x value and as a 2nd parameter we are passing hue value.

```
sns.violinplot(data=df,x='isFraud',y='newbalanceDest')
```

```
<AxesSubplot:xlabel='isFraud', ylabel='newbalanceDest'>
```



Descriptive Analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
df.describe(include='all')
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
count	2430.000000	2430	2.430000e+03	2430	2.430000e+03	2.430000e+03	2430	2.430000e+03	2.430000e+03	2430
unique	NaN	5	NaN	2430	NaN	NaN	1870	NaN	NaN	2
top	NaN	CASH_OUT	NaN	C1231006815	NaN	NaN	C1590550415	NaN	NaN	is not Fraud
freq	NaN	827	NaN	1	NaN	NaN	25	NaN	NaN	1288
mean	23.216049	NaN	6.258361e+05	NaN	9.849040e+05	4.392755e+05	NaN	5.797246e+05	1.127075e+06	NaN
std	29.933036	NaN	1.503866e+06	NaN	2.082361e+06	1.520978e+06	NaN	1.891192e+06	2.907401e+06	NaN
min	1.000000	NaN	8.730000e+00	NaN	0.000000e+00	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN
25%	1.000000	NaN	9.018493e+03	NaN	8.679630e+03	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN
50%	1.000000	NaN	1.058692e+05	NaN	8.096250e+04	0.000000e+00	NaN	0.000000e+00	0.000000e+00	NaN
75%	45.000000	NaN	4.096098e+05	NaN	7.606258e+05	1.247804e+04	NaN	3.096195e+05	9.658701e+05	NaN
max	95.000000	NaN	1.000000e+07	NaN	1.990000e+07	9.987287e+06	NaN	3.300000e+07	3.460000e+07	NaN

Data Pre-Processing

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

Handling missing values
Handling Object data label encoding
Splitting dataset into training and test set

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

```
# Shape of csv data
df.shape
```

```
(2430, 10)
```

Here, I'm using the shape approach to figure out how big my dataset is

```
df.drop(['nameOrig','nameDest'],axis=1,inplace=True)
df.columns
```

```
Index(['step', 'type', 'amount', 'oldbalanceOrg', 'newbalanceOrig',
       'oldbalanceDest', 'newbalanceDest', 'isFraud'],
      dtype='object')
```

```
df.head()
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9.194174	170136.0	160296.36	0.0	0.0	is not Fraud
1	1	PAYMENT	7.530630	21249.0	19384.72	0.0	0.0	is not Fraud
2	1	PAYMENT	9.364617	41554.0	29885.86	0.0	0.0	is not Fraud
3	1	PAYMENT	8.964147	53860.0	46042.29	0.0	0.0	is not Fraud
4	1	PAYMENT	8.868944	183195.0	176087.23	0.0	0.0	is not Fraud

here, the dataset's superfluous columns (nameOrig,nameDest) are being removed using the drop method.

Checking For Null Values

Isnull is used (). sum() to check your database for null values. Using the df.info() function, the data type can be determined.

```
# Finding null values
df.isnull().sum()
```

```
step              0
type              0
amount            0
oldbalanceOrg     0
newbalanceOrig    0
oldbalanceDest    0
newbalanceDest    0
isFraud           0
dtype: int64
```

For checking the null values, data.isnull() function is used. To sum those null values we use the .sum() function to it. From the above image we found that there are no null values present in our dataset.So we can skip handling of missing values step.

```
df.info()
```

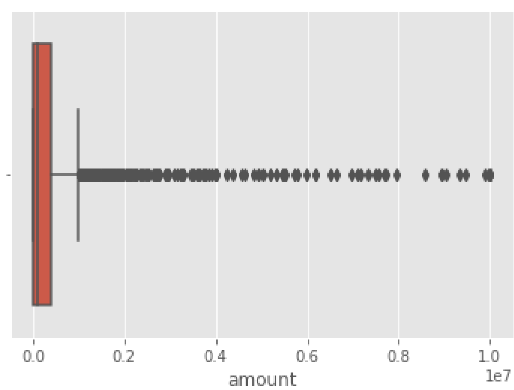
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2430 entries, 0 to 2429
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   step            2430 non-null   int64  
 1   type            2430 non-null   object 
 2   amount          2430 non-null   float64
 3   oldbalanceOrg   2430 non-null   float64
 4   newbalanceOrig  2430 non-null   float64
 5   oldbalanceDest  2430 non-null   float64
 6   newbalanceDest  2430 non-null   float64
 7   isFraud         2430 non-null   object 
dtypes: float64(5), int64(1), object(2)
memory usage: 152.0+ KB
```

determining the types of each attribute in the dataset using the info() function

Handling Outliers

```
sns.boxplot(df['amount'])
```

```
<AxesSubplot:xlabel='amount'>
```



Here, a boxplot is used to identify outliers in the dataset's amount attribute.

Remove the Outliers

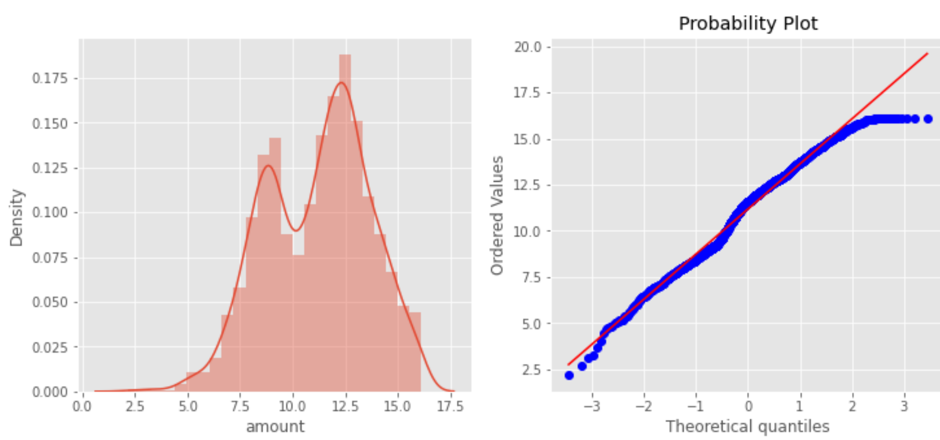
```
from scipy import stats
print(stats.mode(df['amount']))
print(np.mean(df['amount']))
```

```
ModeResult(mode=array([10000000.]), count=array([14]))
625836.0974156366
```

```
q1 = np.quantile(df['amount'],0.25)
q3 = np.quantile(df['amount'],0.75)

IQR = q3-q1

upper_bound = q3+(1.5*IQR)
lower_bound = q1-(1.5*IQR)

print('q1 :',q1)
print('q3 :',q3)
print('IQR :',IQR)
print('Upper Bound :',upper_bound)
print('Lower Bound :',lower_bound)
print('Skewed data :',len(df[df['amount']>upper_bound]))
print('Skewed data :',len(df[df['amount']<lower_bound]))
```

```
# To handle outliers transformation techniques are used.

def transformationPlot(feature):
    plt.figure(figsize=(12,5))
    plt.subplot(1,2,1)
    sns.distplot(feature)
    plt.subplot(1,2,2)
    stats.probplot(feature,plot=plt)
```

```
transformationPlot(np.log(df['amount']))
```




```
df['amount']=np.log(df['amount'])
```

Here, transformationPlot is used to plot the dataset's outliers for the amount property.

Object Data Labelencoding

```
from sklearn.preprocessing import LabelEncoder

la = LabelEncoder()
df['type'] = la.fit_transform(df['type'])
```

```
df['type'].value_counts()
```

```
1    827
4    724
3    580
0    224
2     75
Name: type, dtype: int64
```

using label encoder to encode the dataset's object type

dividing the dataset into dependent and independent y and x respectively

```
x = df.drop('isFraud',axis=1)
y = df['isFraud']
```

```
x
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	1	3	9.194174	170136.00	160296.36	0.00	0.00
1	1	3	7.530630	21249.00	19384.72	0.00	0.00
2	1	3	9.364617	41554.00	29885.86	0.00	0.00
3	1	3	8.964147	53860.00	46042.29	0.00	0.00
4	1	3	8.868944	183195.00	176087.23	0.00	0.00
...	...	...	...	...	...	...	...
2425	95	1	10.946325	56745.14	0.00	51433.88	108179.02
2426	95	4	10.424558	33676.59	0.00	0.00	0.00
2427	95	1	10.424558	33676.59	0.00	0.00	33676.59
2428	95	4	11.385084	87999.25	0.00	0.00	0.00
2429	95	1	11.385084	87999.25	0.00	0.00	87999.25

2430 rows × 7 columns

```
y
```

```
0       is not Fraud
1       is not Fraud
2       is not Fraud
3       is not Fraud
4       is not Fraud
            ...
2425        is Fraud
2426        is Fraud
2427        is Fraud
2428        is Fraud
2429        is Fraud
Name: isFraud, Length: 2430, dtype: object
```

Splitting Data Into Train And Test

Now let's split the Dataset into train and test setsChanges: first split the dataset into x and y and then split the data set.

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And my target variable is passed. For splitting training and testing data we are using the train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

Train test split¶

```
from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=0,test_size=0.2)
```

```
print(x_train.shape)
print(x_test.shape)
print(y_test.shape)
print(y_train.shape)
```

```
(1944, 7)
(486, 7)
(486,)
(1944,)
```

Evaluating Performance Of The Model And Saving The Model

From sklearn, accuracy_score is used to evaluate the score of the model. On the parameters, we have given svc (model name), x, y, cv (as 5 folds). Our model is performing well. So, we are saving the model is svc by pickle.dump().

```
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
svc= SVC()
svc.fit(x_train,y_train)
y_test_predict4=svc.predict(x_test)
test_accuracy=accuracy_score(y_test,y_test_predict4)
test_accuracy
```

```
0.7901234567901234
```

```
y_train_predict4=svc.predict(x_train)
train_accuracy=accuracy_score(y_train,y_train_predict4)
train_accuracy
```

```
0.8009259259259259
```

```
import pickle
pickle.dump(svc,open('payments.pkl','wb'))
```