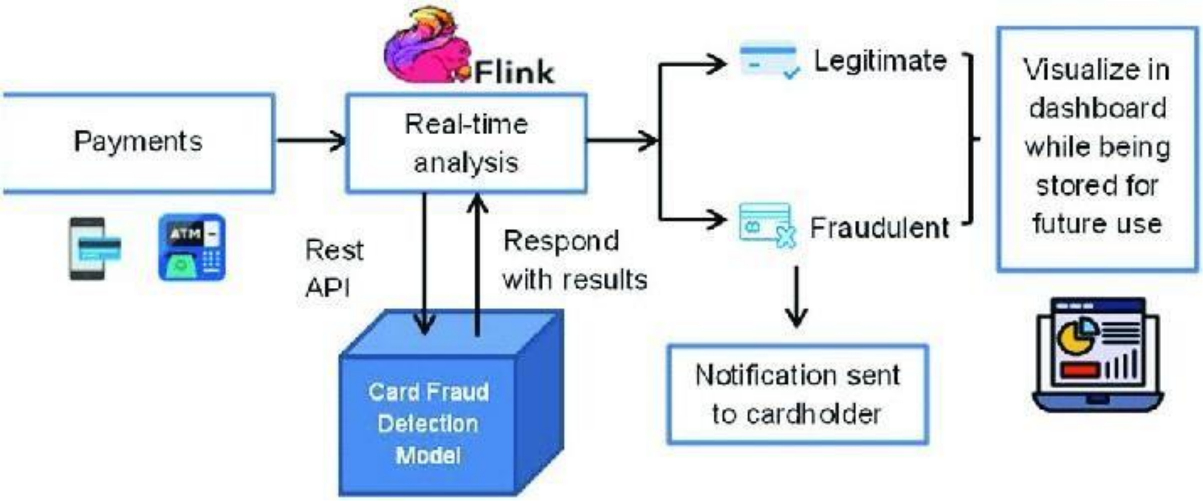
Online Payments Fraud Detection using ML

Project Description:

The growth in internet and e-commerce appears to involve the use of online credit/debit card transactions. The increase in the use of credit / debit cards is causing an increase in fraud. The frauds can be detected through various approaches, yet they lag in their accuracy and its own specific drawbacks. If there are any changes in the conduct of the transaction, the frauds are predicted and taken for further process. Due to large amount of data credit / debit card fraud detection problem is rectified by the proposed method.

We will be using classification algorithms such as Decision tree, Random forest, svm, and Extra tree classifier, xgboost Classifier**.**We will train and test the data with these algorithms. From this the best model is selected and saved in pkl format. We will be doing flask integration and IBM deployment.

TECHNICAL ARCHITECTURE:


Flink
Payments
Real-time analysis
Rest API
Respond with results
Card Fraud Detection Model
Legitimate
Fraudulent
Notification sent to cardholder
Visualize in dashboard while being stored for future use

Project Objectives:

By the end of this project you will:

- Know fundamental concepts and techniques used for machine learning.
- Gain a broad understanding about data.
- Have knowledge on pre-processing the data/transformation techniques on outlier and some visualization concepts.

Project Flow:

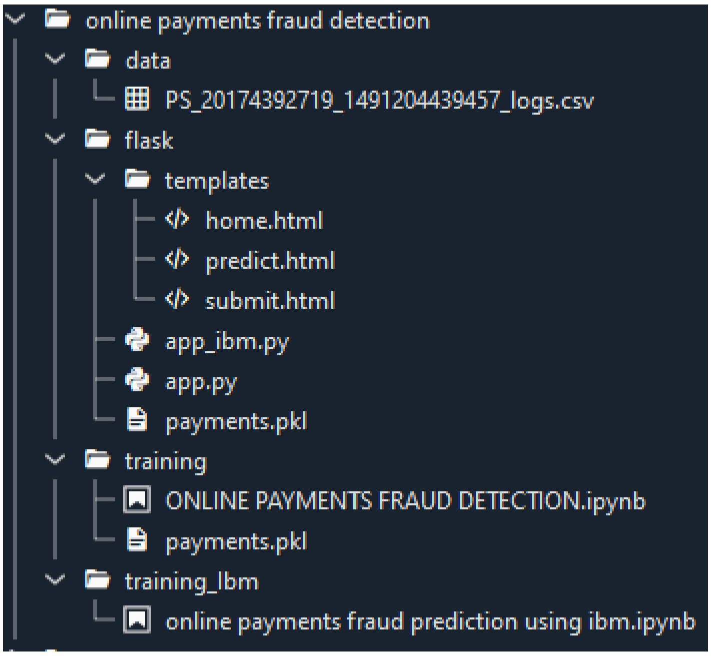
- User interacts with the UI to enter the input.
- Entered input is analyzed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Data collection
- visualizing and analyzing data
- Model building
- Application Building.

Project Structure:

Create the Project folder which contains files as shown below



```
online payments fraud detection
  data
    PS_20174392719_1491204439457_logs.csv
  flask
    templates
      home.html
      predict.html
      submit.html
    app_ibm.py
    app.py
    payments.pkl
  training
    ONLINE PAYMENTS FRAUD DETECTION.ipynb
    payments.pkl
  training_Ibm
    online payments fraud prediction using ibm.ipynb
```

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Model.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains model training files and the training_ibm folder contains IBM deployment files.

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.

Collect the dataset or create the dataset or Download the dataset:

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

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 the dataset.

Link: https://www.kaggle.com/datasets/rupakroy/online-payments-fraud-detection-dataset

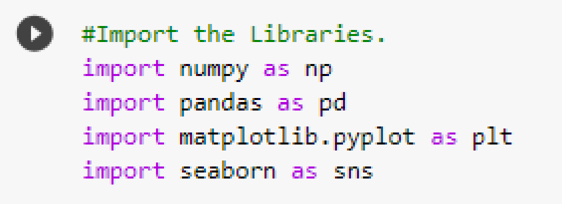
visualizing and analyzing data

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

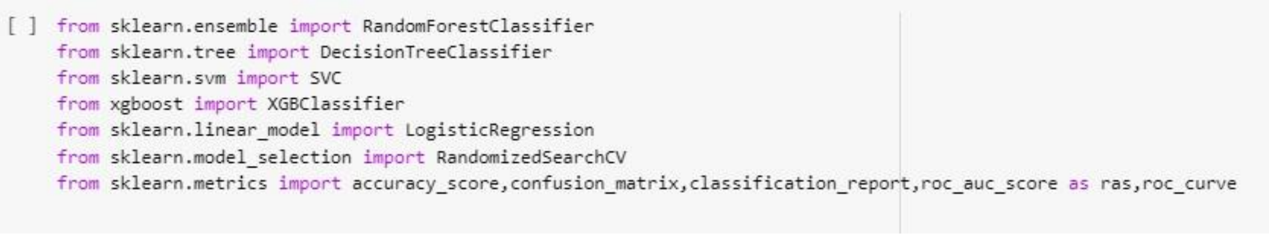
Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Importing the libraries

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



```
#Import the Libraries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

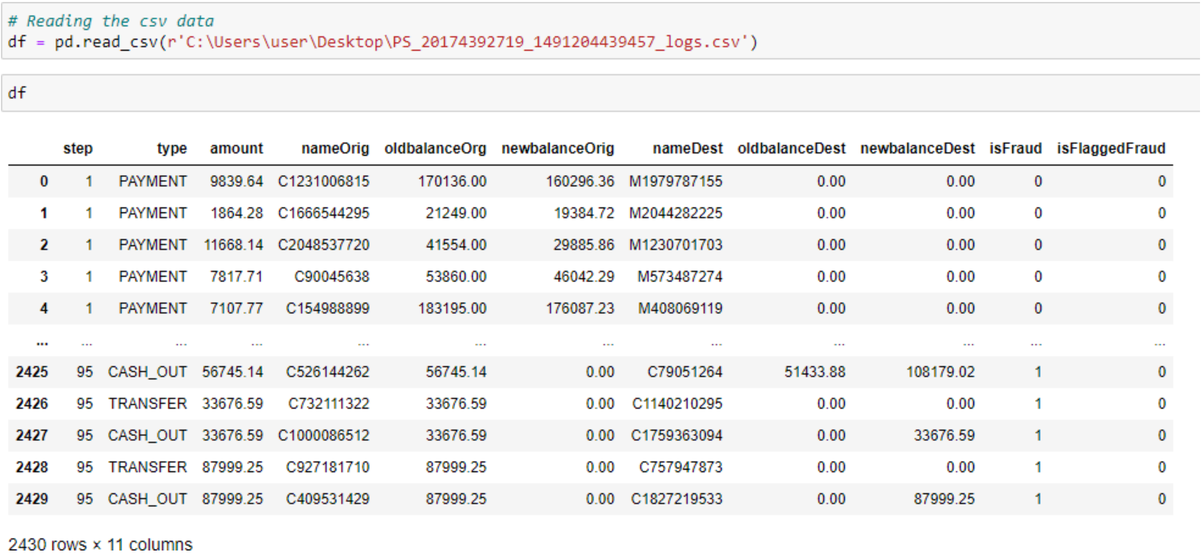


```
[ ] from sklearn.ensemble import RandomForestClassifier
    from sklearn.tree import DecisionTreeClassifier
    from sklearn.svm import SVC
    from xgboost import XGBClassifier
    from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import RandomizedSearchCV
    from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_auc_score as ras,roc_curve
```

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.

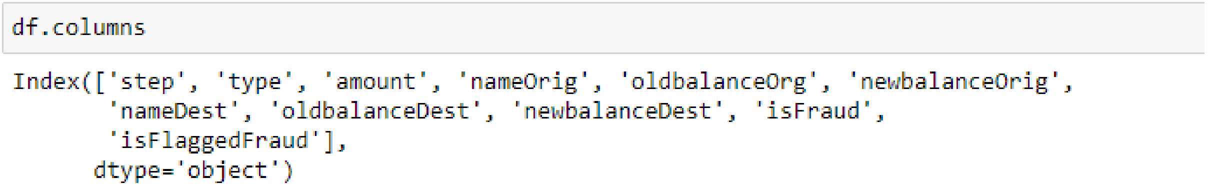


```
# Reading the csv data
df = pd.read_csv(r'C:\Users\user\Desktop\PS_20174392719_1491204439457_logs.csv')
```

```
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

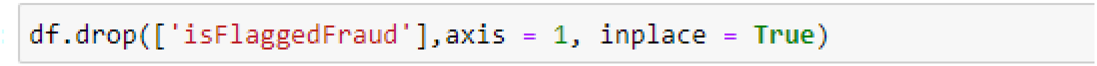
2430 rows × 11 columns



```
df.columns
```

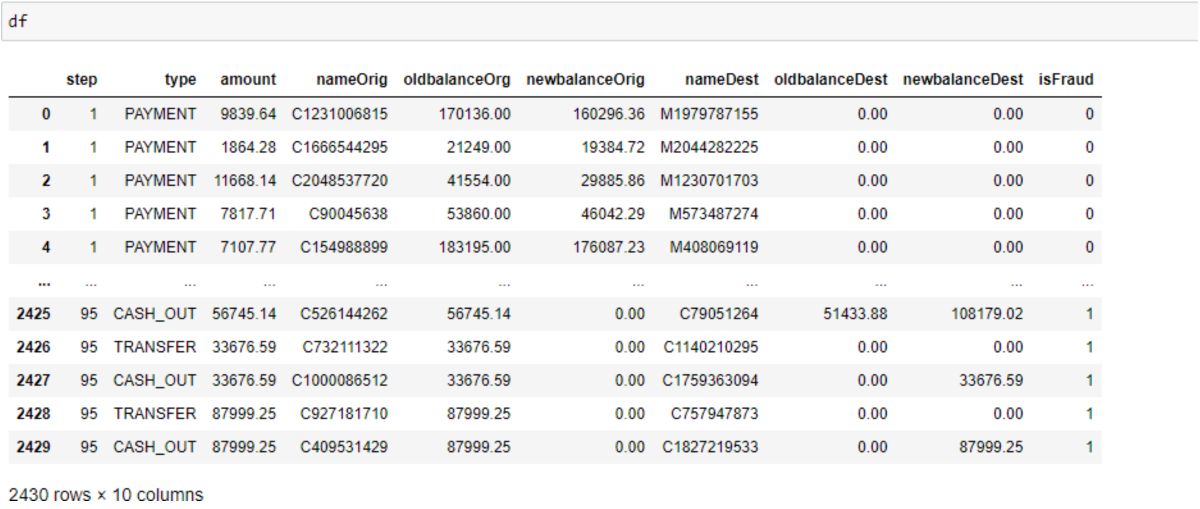
```
Index(['step', 'type', 'amount', 'nameOrig', 'oldbalanceOrg', 'newbalanceOrig',
       'nameDest', 'oldbalanceDest', 'newbalanceDest', 'isFraud',
       'isFlaggedFraud'],
      dtype='object')
```

Here, the input features in the dataset are known using the df.columns function.



```
df.drop(['isFlaggedFraud'],axis = 1, inplace = True)
```

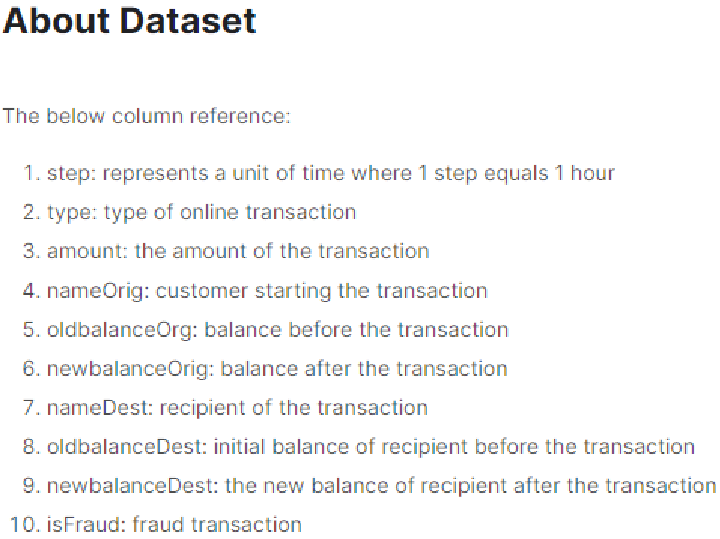
here, the dataset's superfluous columns are being removed using the drop method.



```
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	0
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M2044282225	0.00	0.00	0
2	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M1230701703	0.00	0.00	0
3	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	M573487274	0.00	0.00	0
4	1	PAYMENT	7107.77	C154988899	183195.00	176087.23	M408069119	0.00	0.00	0
...	...	...	...	...	...	...	...	...	...	...
2425	95	CASH_OUT	56745.14	C526144262	56745.14	0.00	C79051264	51433.88	108179.02	1
2426	95	TRANSFER	33676.59	C732111322	33676.59	0.00	C1140210295	0.00	0.00	1
2427	95	CASH_OUT	33676.59	C1000086512	33676.59	0.00	C1759363094	0.00	33676.59	1
2428	95	TRANSFER	87999.25	C927181710	87999.25	0.00	C757947873	0.00	0.00	1
2429	95	CASH_OUT	87999.25	C409531429	87999.25	0.00	C1827219533	0.00	87999.25	1

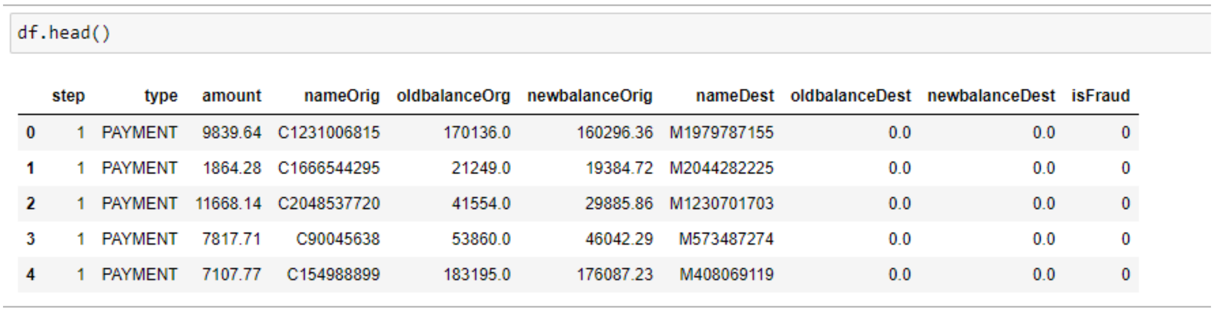
2430 rows × 10 columns



About Dataset

The below column reference:

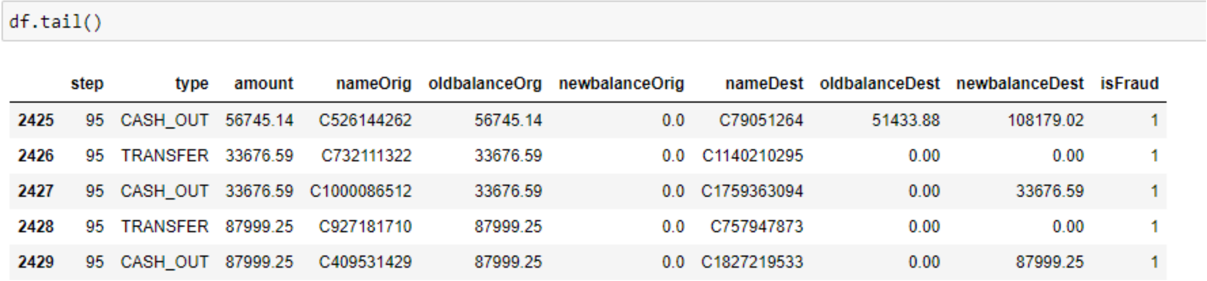
1. step: represents a unit of time where 1 step equals 1 hour
2. type: type of online transaction
3. amount: the amount of the transaction
4. nameOrig: customer starting the transaction
5. oldbalanceOrg: balance before the transaction
6. newbalanceOrig: balance after the transaction
7. nameDest: recipient of the transaction
8. oldbalanceDest: initial balance of recipient before the transaction
9. newbalanceDest: the new balance of recipient after the transaction
10. isFraud: fraud transaction



```
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0	0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0	0
2	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0	0
3	1	PAYMENT	7817.71	C90045638	53860.0	46042.29	M573487274	0.0	0.0	0
4	1	PAYMENT	7107.77	C154988899	183195.0	176087.23	M408069119	0.0	0.0	0

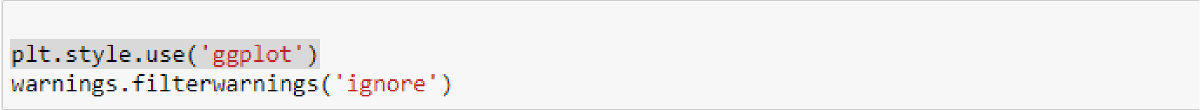
above, the dataset's first five values are loaded using the head method.



```
df.tail()
```

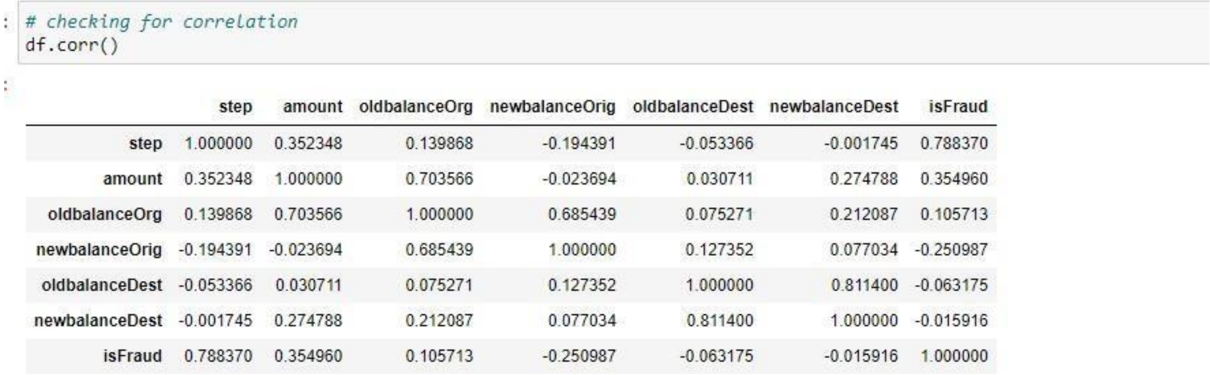
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud
2425	95	CASH_OUT	56745.14	C526144262	56745.14	0.0	C79051264	51433.88	108179.02	1
2426	95	TRANSFER	33676.59	C732111322	33676.59	0.0	C1140210295	0.00	0.00	1
2427	95	CASH_OUT	33676.59	C1000086512	33676.59	0.0	C1759363094	0.00	33676.59	1
2428	95	TRANSFER	87999.25	C927181710	87999.25	0.0	C757947873	0.00	0.00	1
2429	95	CASH_OUT	87999.25	C409531429	87999.25	0.0	C1827219533	0.00	87999.25	1

above, the dataset's last five values are loaded using the tail method.



```
plt.style.use('ggplot')
warnings.filterwarnings('ignore')
```

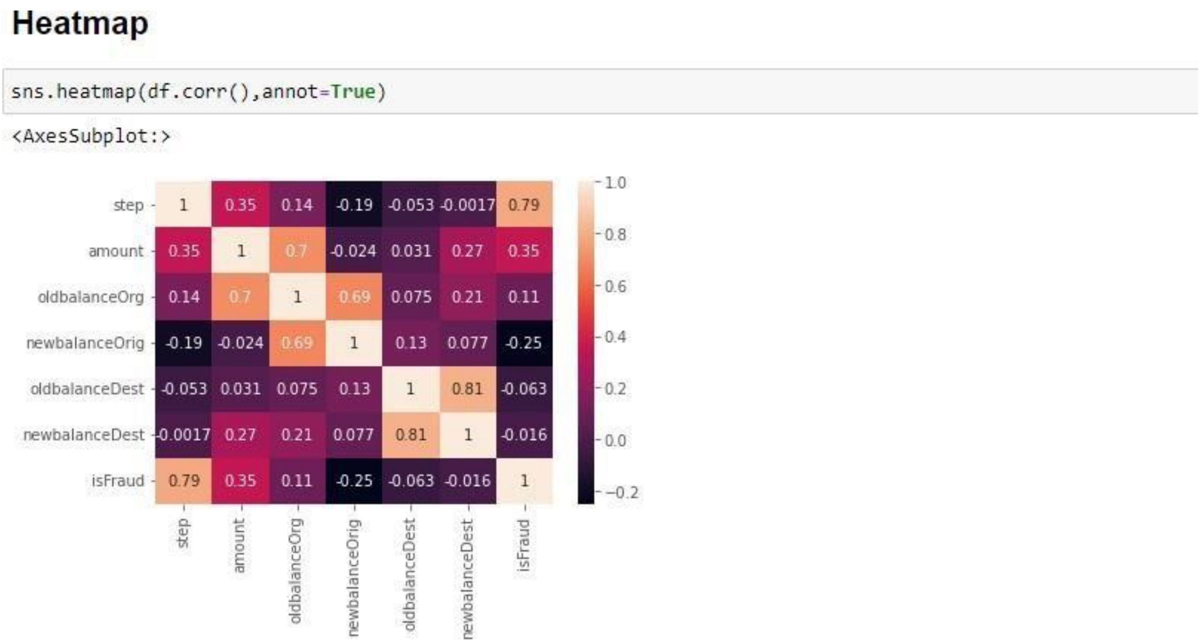
utilising Style use here The Ggplot approach Setting "styles"—basically stylesheets that resemble matplotlibrc files—is a fundamental feature of mpltools. The "ggplot" style, which modifies the style to resemble ggplot, is demonstrated in this dataset.



```
# checking for correlation
df.corr()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
step	1.000000	0.352348	0.139868	-0.194391	-0.053366	-0.001745	0.788370
amount	0.352348	1.000000	0.703566	-0.023694	0.030711	0.274788	0.354960
oldbalanceOrg	0.139868	0.703566	1.000000	0.685439	0.075271	0.212087	0.105713
newbalanceOrig	-0.194391	-0.023694	0.685439	1.000000	0.127352	0.077034	-0.250987
oldbalanceDest	-0.053366	0.030711	0.075271	0.127352	1.000000	0.811400	-0.063175
newbalanceDest	-0.001745	0.274788	0.212087	0.077034	0.811400	1.000000	-0.015916
isFraud	0.788370	0.354960	0.105713	-0.250987	-0.063175	-0.015916	1.000000

utilizing the core function to examine the dataset's correlation



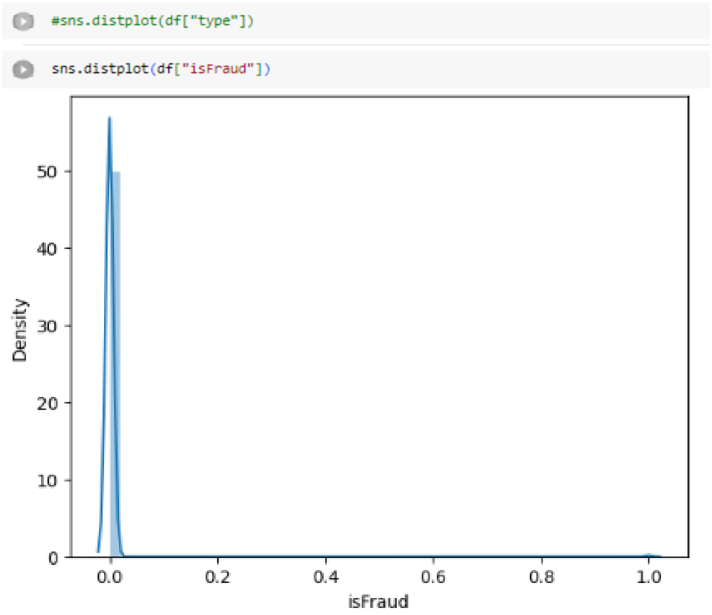
Heatmap

```
sns.heatmap(df.corr(),annot=True)
```

```
<AxesSubplot:>
```

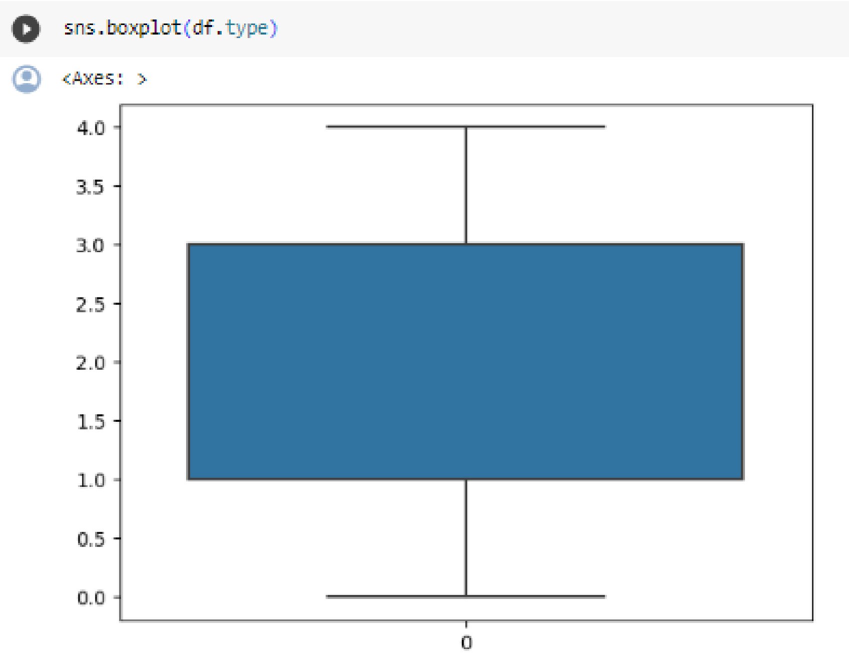
	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
step	1	0.35	0.14	-0.19	-0.053	-0.0017	0.79
amount	0.35	1	0.7	-0.024	0.031	0.27	0.35
oldbalanceOrg	0.14	0.7	1	0.69	0.075	0.21	0.11
newbalanceOrig	-0.19	-0.024	0.69	1	0.13	0.077	-0.25
oldbalanceDest	-0.053	0.031	0.075	0.13	1	0.81	-0.063
newbalanceDest	-0.0017	0.27	0.21	0.077	0.81	1	-0.016
isFraud	0.79	0.35	0.11	-0.25	-0.063	-0.016	1

Here, a heatmap is used to understand the relationship between the input attributes and the anticipated goal value.



```
#sns.distplot(df["type"])
```

```
sns.distplot(df["isFraud"])
```

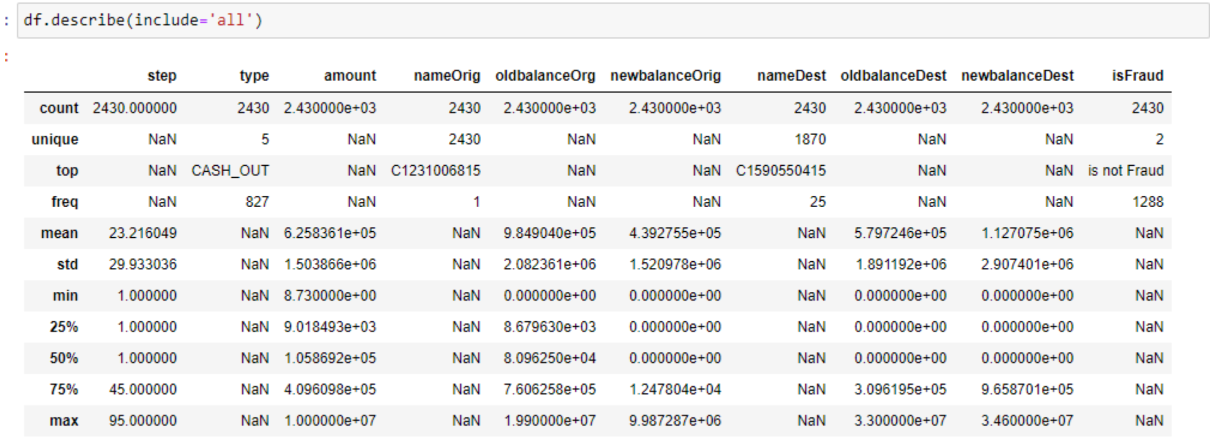


```
sns.boxplot(df.type)
```

```
<Axes: >
```

Descriptive analysis

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas have 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.

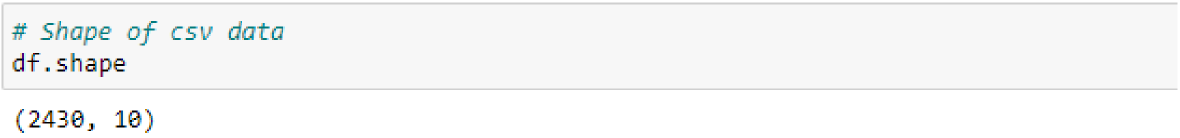
The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

Handling missing values

Handling Object data label encoding

Splitting dataset into training and test set

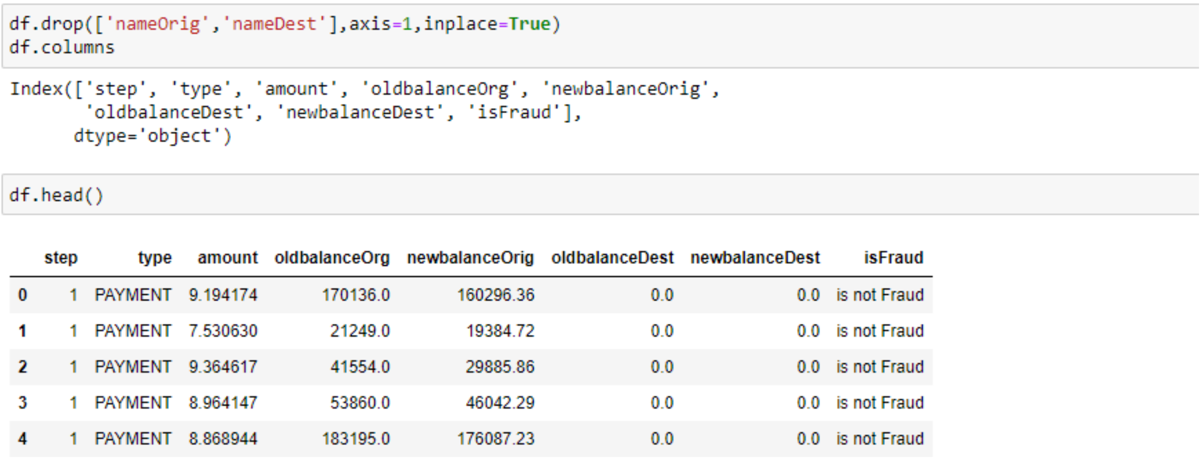
Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.



```
# 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')
```

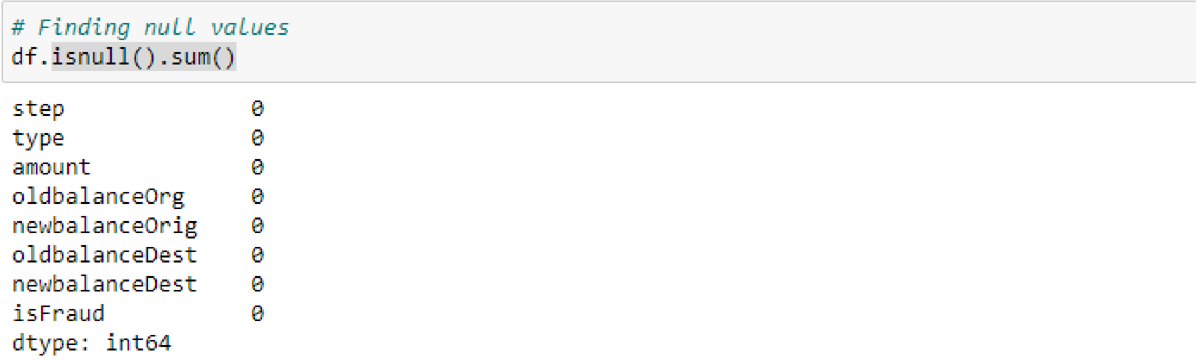
```
df.head()
```

	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

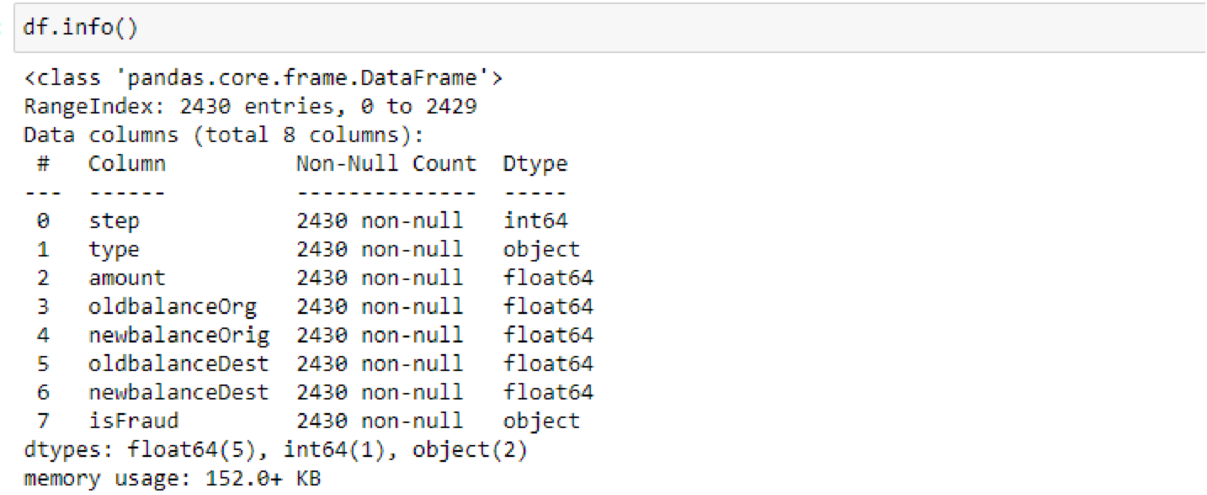
Isnull is used (). sum() to check your database for null values. Using the df.info() function, the data type can be determined.



```
# 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.

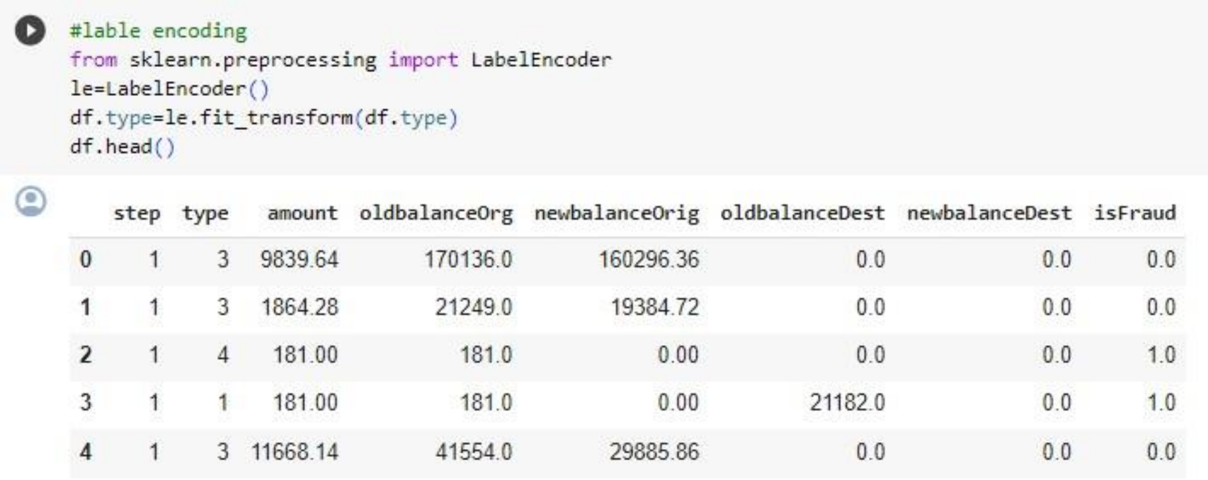


```
df.info()
```

```
<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.

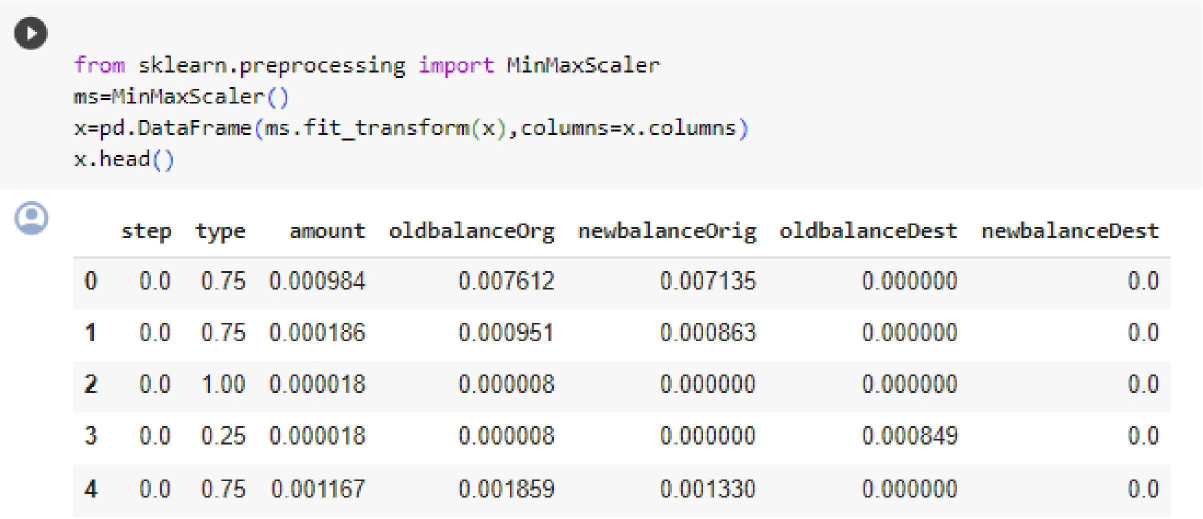
Object data label encoding



```
#lable encoding
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
df.type=le.fit_transform(df.type)
df.head()
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud
0	1	3	9839.64	170136.0	160296.36	0.0	0.0	0.0
1	1	3	1864.28	21249.0	19384.72	0.0	0.0	0.0
2	1	4	181.00	181.0	0.00	0.0	0.0	1.0
3	1	1	181.00	181.0	0.00	21182.0	0.0	1.0
4	1	3	11668.14	41554.0	29885.86	0.0	0.0	0.0

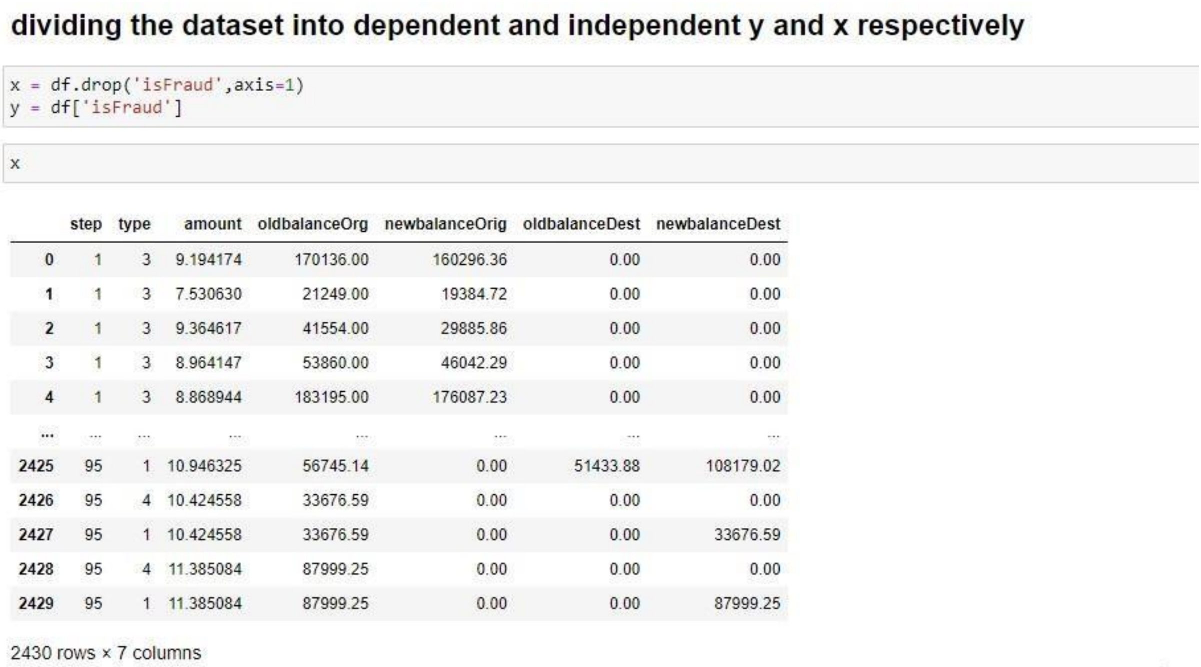
FEATURE SCALING:



```
from sklearn.preprocessing import MinMaxScaler
ms=MinMaxScaler()
x=pd.DataFrame(ms.fit_transform(x),columns=x.columns)
x.head()
```

	step	type	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest
0	0.0	0.75	0.000984	0.007612	0.007135	0.000000	0.0
1	0.0	0.75	0.000186	0.000951	0.000863	0.000000	0.0
2	0.0	1.00	0.000018	0.000008	0.000000	0.000000	0.0
3	0.0	0.25	0.000018	0.000008	0.000000	0.000849	0.0
4	0.0	0.75	0.001167	0.001859	0.001330	0.000000	0.0

using label encoder to encode the dataset's object type



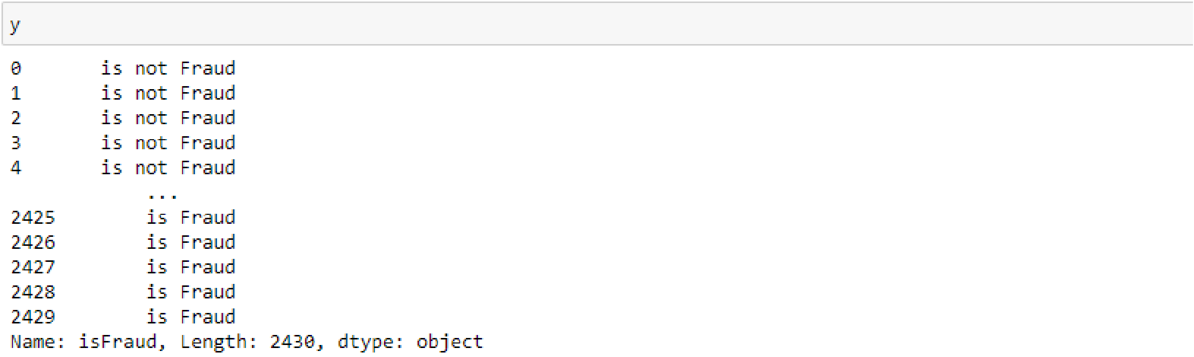
dividing the dataset into dependent and independent y and x respectively

```
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



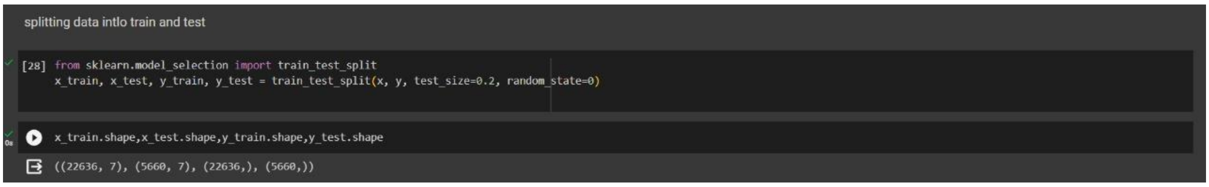
```
y
```

```
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.



splitting data intlo train and test

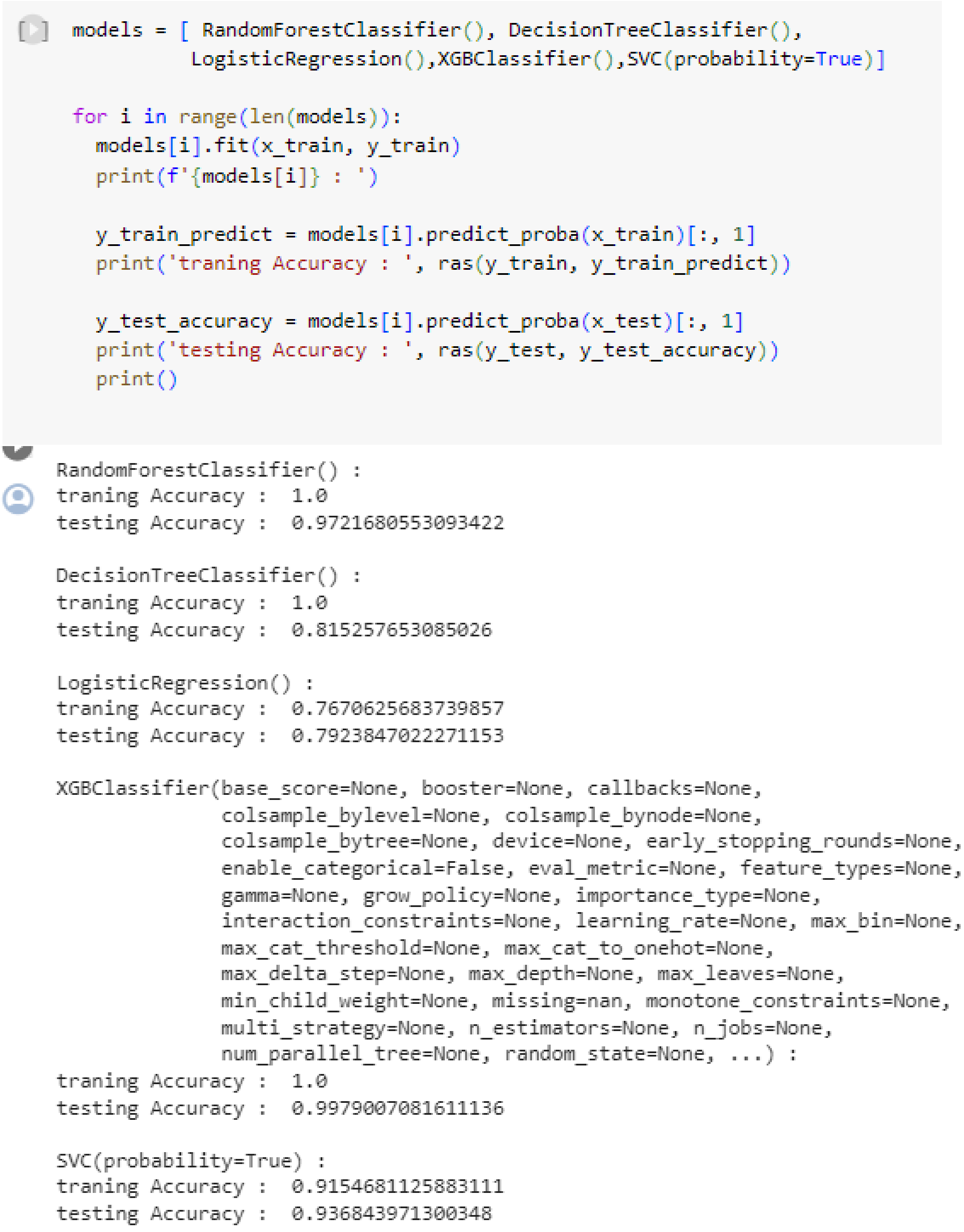
```
[28] from sklearn.model_selection import train_test_split
     x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=0)
```

```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

```
((22636, 7), (5660, 7), (22636,), (5660,))
```

Model Building

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying four classification algorithms. The best model is saved based on its performance.



```
models = [ RandomForestClassifier(), DecisionTreeClassifier(),
           LogisticRegression(),XGBClassifier(),SVC(probability=True)]

for i in range(len(models)):
  models[i].fit(x_train, y_train)
  print(f'{models[i]} : ')

  y_train_predict = models[i].predict_proba(x_train)[:, 1]
  print('traning Accuracy : ', ras(y_train, y_train_predict))

  y_test_accuracy = models[i].predict_proba(x_test)[:, 1]
  print('testing Accuracy : ', ras(y_test, y_test_accuracy))
  print()
```

```
RandomForestClassifier() :
traning Accuracy :  1.0
testing Accuracy :  0.9721680553093422

DecisionTreeClassifier() :
traning Accuracy :  1.0
testing Accuracy :  0.815257653085026

LogisticRegression() :
traning Accuracy :  0.7670625683739857
testing Accuracy :  0.7923847022271153

XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...) :
traning Accuracy :  1.0
testing Accuracy :  0.9979007081611136

SVC(probability=True) :
traning Accuracy :  0.9154681125883111
testing Accuracy :  0.936843971300348
```

Classifiers

Several classifiers have been implemented in this project, each encapsulated within a dedicated function. The RandomForestClassifier function takes training and test data as parameters, initializing the algorithm and training the model using the .fit() function. The test data is then predicted with the .predict() function, and the results are saved in a new variable. Model evaluation involves generating a confusion matrix and classification report. Similarly, the DecisionTreeClassifier and ExtraTreeClassifier functions follow a similar structure, initializing their respective algorithms, training the models, and evaluating performance. The SupportVectorClassifier function applies the Support Vector Machine algorithm, and the xgboostClassifier function employs the XGBoost algorithm. In both cases, training data is utilized to fit the model, and predictions on test data are made using the .predict() function. Evaluation metrics, including confusion matrices and classification reports, are generated for each classifier. Additionally, a note on preprocessing using the LabelEncoder from the sklearn library is mentioned, providing a comprehensive overview of the implemented classifiers and their respective evaluation methodologies.

Compare the model

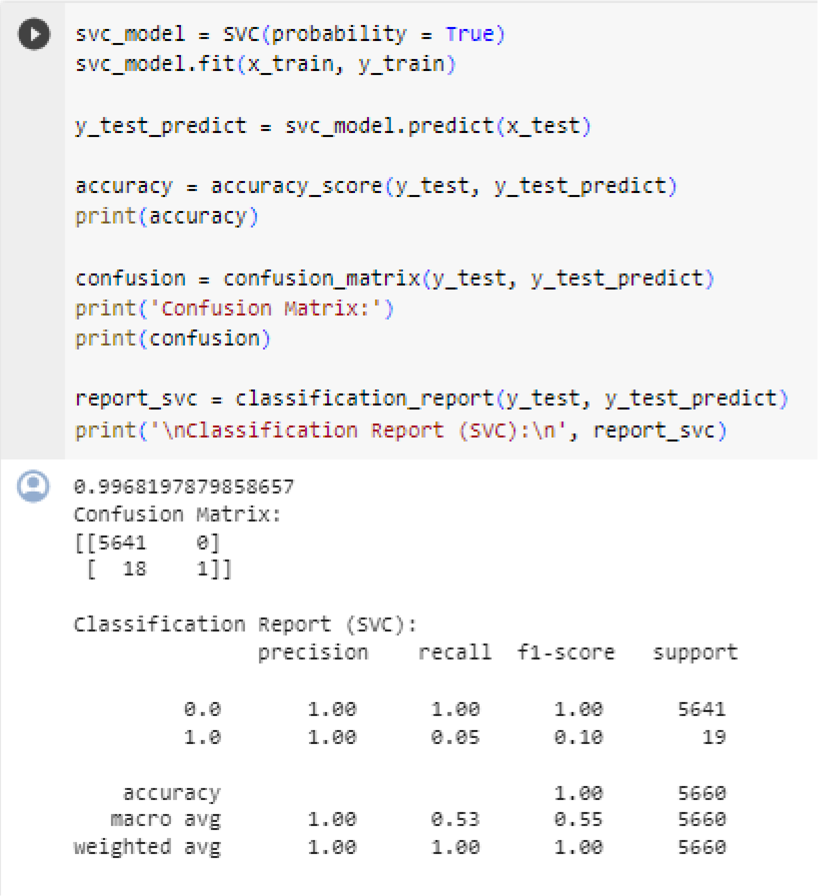
To compare the performance of the four models mentioned earlier, a "compareModel" function has been defined. Upon invoking this function, the output displays the results of each model. Notably, the Support Vector Classifier (SVC) stands out as the top-performing model among the five. The accompanying image reveals a 79% accuracy for the SVC, indicating its superior predictive capability.

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().

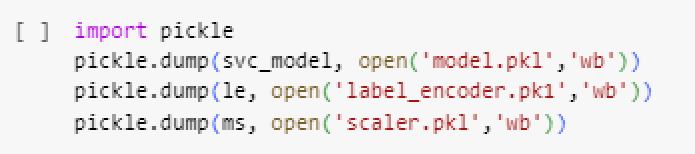
Application Building

Within this segment, our focus lies in constructing a web application seamlessly integrated with the previously developed model. A user-friendly interface has been designed for users to input values for predictions. These entered values are then forwarded to the pre-trained model, and the resulting predictions are presented on the user interface. The tasks encompassed in this section include the creation of HTML pages and the development of server-side scripts.



```python
svc_model = SVC(probability = True)
svc_model.fit(x_train, y_train)

y_test_predict = svc_model.predict(x_test)

accuracy = accuracy_score(y_test, y_test_predict)
print(accuracy)

confusion = confusion_matrix(y_test, y_test_predict)
print('Confusion Matrix:')
print(confusion)

report_svc = classification_report(y_test, y_test_predict)
print('\nClassification Report (SVC):\n', report_svc)
```

```
0.9968197879858657
Confusion Matrix:
[[5641    0]
 [  18    1]]

Classification Report (SVC):
               precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      5641
         1.0       1.00      0.05      0.10        19

    accuracy                           1.00      5660
   macro avg       1.00      0.53      0.55      5660
weighted avg       1.00      1.00      1.00      5660
```



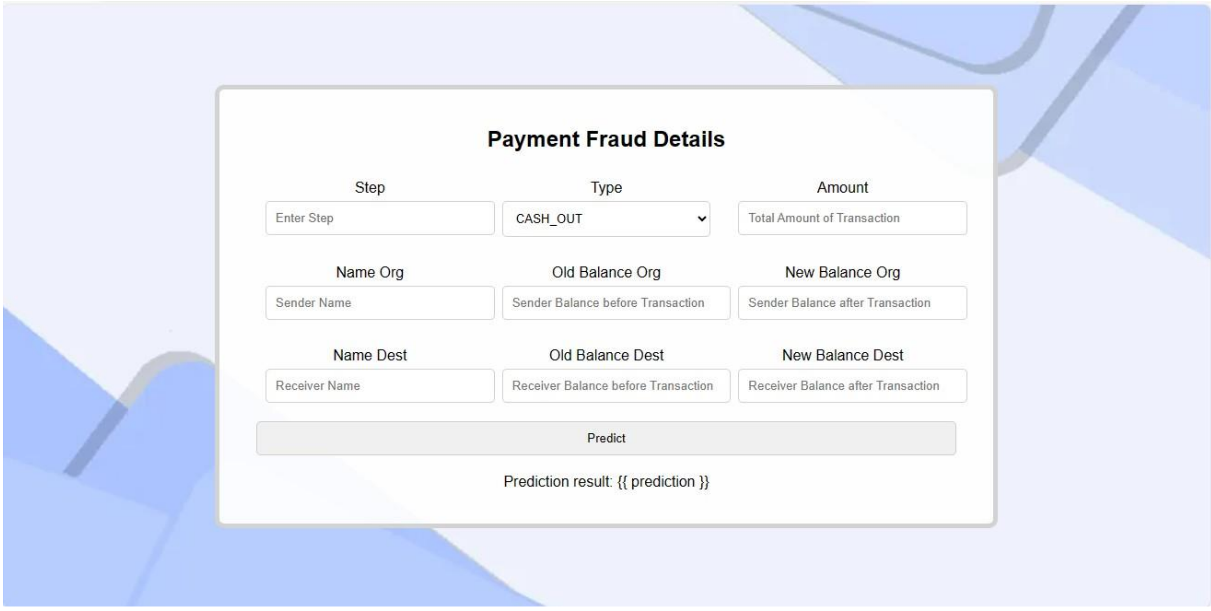
```
[ ]  import pickle
     pickle.dump(svc_model, open('model.pkl','wb'))
     pickle.dump(le, open('label_encoder.pkl','wb'))
     pickle.dump(ms, open('scaler.pkl','wb'))
```

Building Html Pages:

For this project create three HTML files namely

- login.html
- register.html
- predict.html

Initially, users will encounter the login page, providing them with the option to log into their accounts. For those new to the application, a registration process is available for creating an account. Following a successful login, users will be directed to the home screen, where they can input payment values and utilize the system to predict outcomes.



Payment Fraud Details
Step
Enter Step
Amount
Total Amount of Transaction
Name Org
Sender Name
Old Balance Org
Sender Balance before Transaction
New Balance Org
Sender Balance after Transaction
Name Dest
Receiver Name
Old Balance Dest
Receiver Balance before Transaction
New Balance Dest
Receiver Balance after Transaction
Predict

Type: CASH_OUT

Prediction result: {{ prediction }}

Upon clicking the "predict" button, users will receive the outcome in the prediction result section.