Financial Fraud Dataset

representation of mobile money transactions, meticulously crafted to mirror the complexities of real-world financial activities while integrating fraudulent behaviors for research purposes.

Derived from a simulator named PaySim, which utilizes aggregated data from actual financial logs of a mobile money service in an African country, this dataset aims to fill the gap in publicly available financial datasets for fraud detection studies. It encompasses a variety of transaction types including CASH-IN, CASH-OUT, DEBIT, PAYMENT, and TRANSFER over a simulated period of 30 days, providing a comprehensive environment for evaluating fraud detection methodologies.

Double-click (or enter) to edit

It contains only numerical input variables which are the result of a PCA transformation. Unfortunately, due to confidentiality issues, we cannot provide the original features and more background information about the data. Features V1, V2, … V28 are the principal components

These logs were provided by a multinational company that offers this financial service across more than 14 countries globally.

Dataset Description

step: Represents a unit of time in the real world, with 1 step equating to 1 hour. The total simulation spans 744 steps, equivalent to 30 days.

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

amount: The transaction amount in the local currency.

nameOrig: The customer initiating the transaction.

oldbalanceOrg: The initial balance before the transaction.

newbalanceOrig: The new balance after the transaction.

nameDest: The transaction's recipient customer.

oldbalanceDest: The initial recipient's balance before the transaction. Not applicable for customers identified by 'M' (Merchants).

newbalanceDest: The new recipient's balance after the transaction. Not applicable for 'M' (Merchants).

isFraud: Identifies transactions conducted by fraudulent agents aiming to deplete customer accounts through transfers and cash-outs.

isFlaggedFraud: Flags large-scale, unauthorized transfers between accounts, with any single transaction exceeding 200,000 being considered illegal.

```
!unzip /content/Synthetic_Financial_datasets_log.csv.zip
```

```
Archive:  /content/Synthetic_Financial_datasets_log.csv.zip
  inflating: Synthetic_Financial_datasets_log.csv
```

```
import pandas as pd
```

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

```
data.head(10)
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDes
0	1	PAYMENT	9839.64	C1231006815	170136.00	160296.36	M197978715
1	1	PAYMENT	1864.28	C1666544295	21249.00	19384.72	M204428222
2	1	TRANSFER	181.00	C1305486145	181.00	0.00	C55326406
3	1	CASH_OUT	181.00	C840083671	181.00	0.00	C3899701
4	1	PAYMENT	11668.14	C2048537720	41554.00	29885.86	M12307017C
5	1	PAYMENT	7817.71	C90045638	53860.00	46042.29	M57348727
6	1	PAYMENT	7107.77	C154988899	183195.00	176087.23	M40806911
7	1	PAYMENT	7861.64	C1912850431	176087.23	168225.59	M63332633
8	1	PAYMENT	4024.36	C1265012928	2671.00	0.00	M11769321C
9	1	DEBIT	5337.77	C712410124	41720.00	36382.23	C19560086

```
data.shape
```

```
(284807, 31)
```

```
data.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
```

```
data.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbala
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.36
mean	2.433972e+02	1.798619e+05	8.338831e+05	8.551137e+05	1.100702e+06	1.22
std	1.423320e+02	6.038582e+05	2.888243e+06	2.924049e+06	3.399180e+06	3.67
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.00
25%	1.560000e+02	1.338957e+04	0.000000e+00	0.000000e+00	0.000000e+00	0.00
50%	2.390000e+02	7.487194e+04	1.420800e+04	0.000000e+00	1.327057e+05	2.14
75%	3.350000e+02	2.087215e+05	1.073152e+05	1.442584e+05	9.430367e+05	1.11
max	7.430000e+02	9.244552e+07	5.958504e+07	4.958504e+07	3.560159e+08	3.56

```
data.isna().sum() #checks for missing values , and no missing values
```

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

```
#Finding the unique values in each columns

for col in data.columns:
  print(data[col].value_counts())
```

```
44707.62           1
29850.29           1
2251.93            1
165200.06          1
Name: count, Length: 2682586, dtype: int64
nameDest
C1286084959    113
C985934102     109
C665576141     105
C2083562754    102
C1590550415    101
              ...
M295304806       1
M33419717        1
M1940055334      1
M335107734       1
M1757317128      1
Name: count, Length: 2722362, dtype: int64
oldbalanceDest
0.00           2704388
10000000.00        615
20000000.00        219
30000000.00         86
40000000.00         31
                ...
2039554.04           1
587552.25            1
1326910.11           1
230693.29            1
851586.36            1
Name: count, Length: 3614697, dtype: int64
newbalanceDest
0.00           2439433
10000000.00         53
971418.91           32
19169204.93         29
16532032.16         25
                ...
1347758.15           1
3878719.83           1
1605826.83           1
592930.77            1
2580880.68           1
Name: count, Length: 3555499, dtype: int64
isFraud
0    6354407
1       8213
Name: count, dtype: int64
isFlaggedFraud
0    6362604
1         16
Name: count, dtype: int64
```

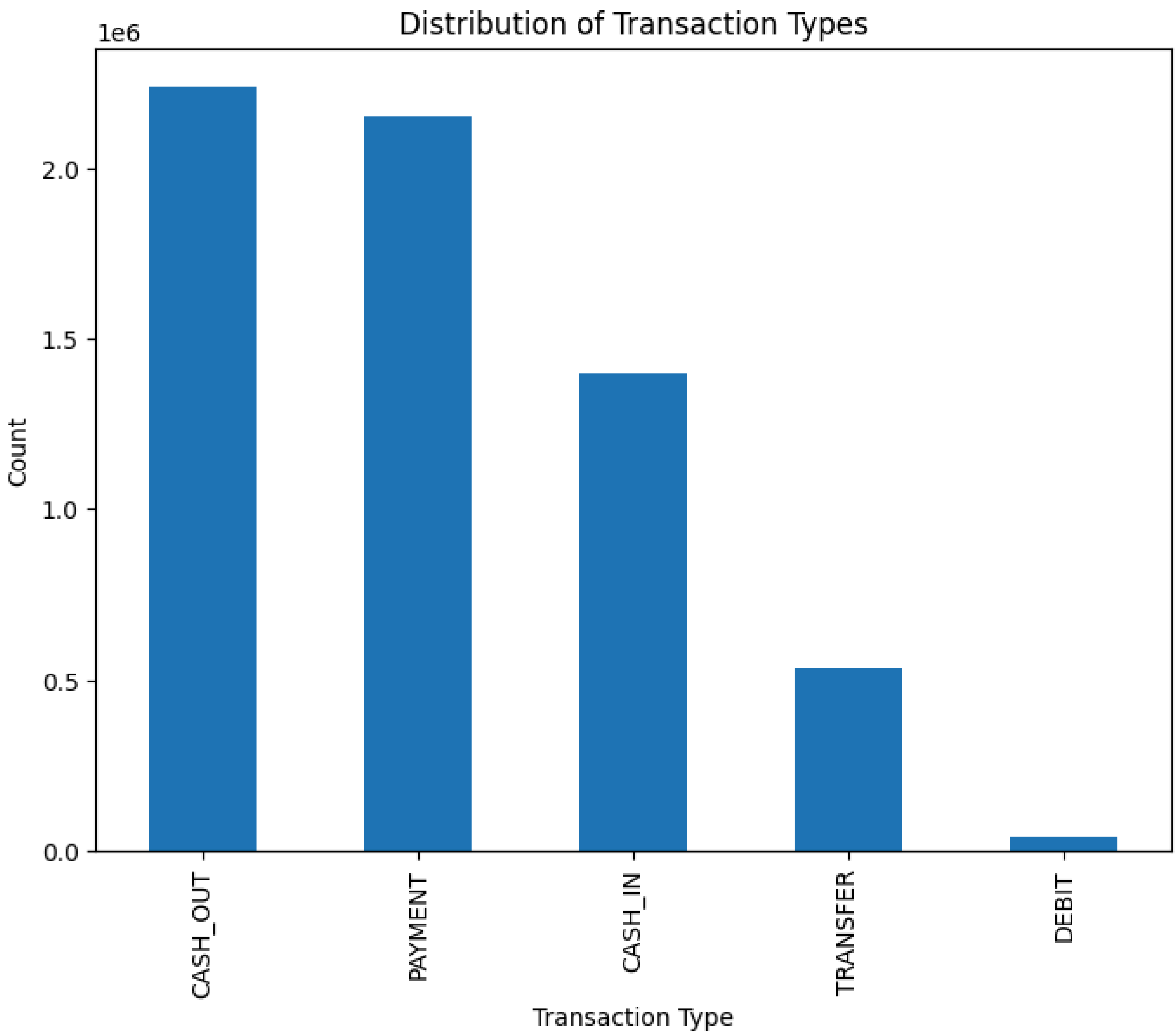
EDA

```
data.head()
```

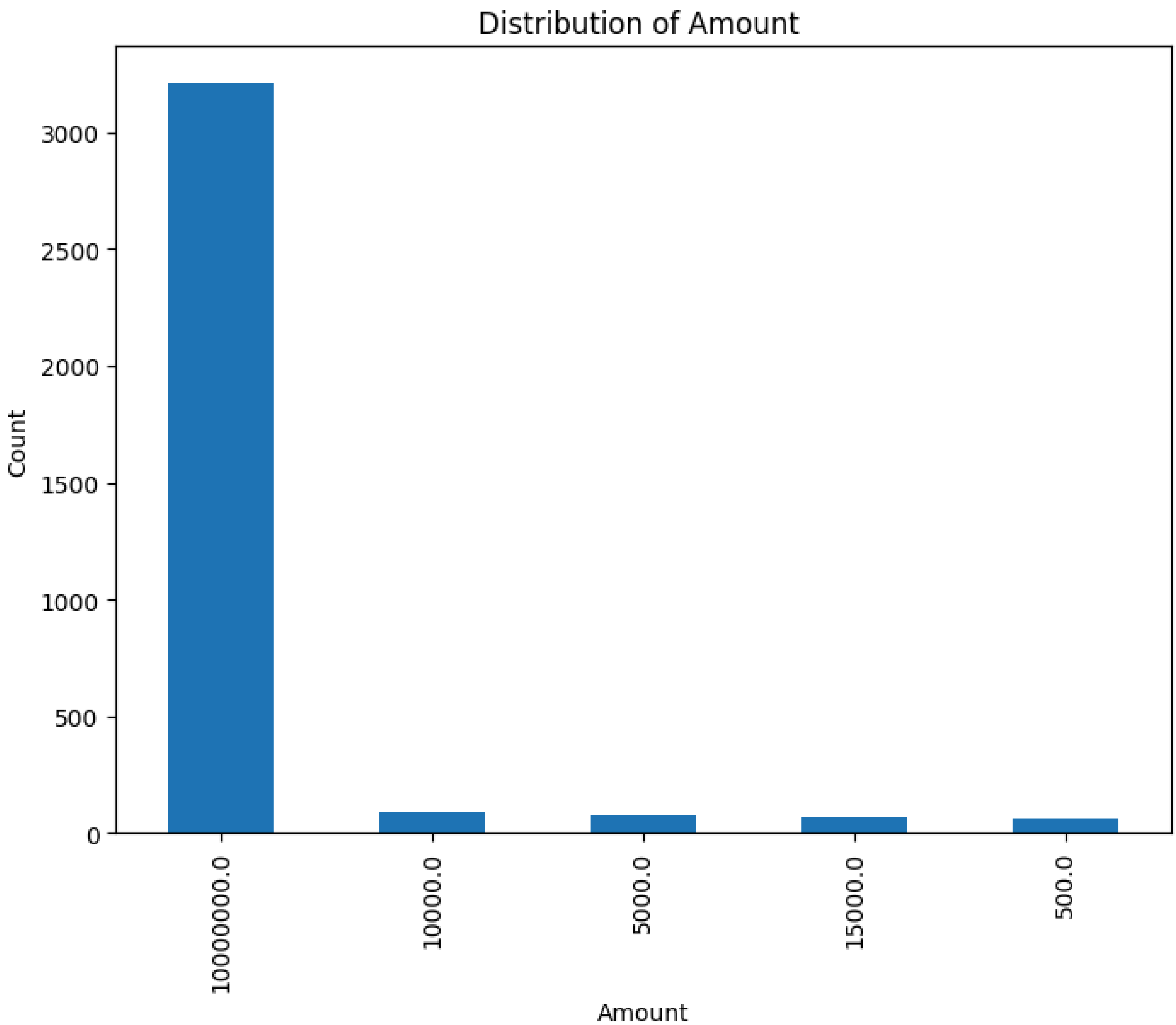
	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDes
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M197978715
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M204428222
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C55326406
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C3899701
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M12307017C

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize=(8,6))
data['type'].value_counts().plot(kind='bar')
plt.xlabel('Transaction Type')
plt.ylabel('Count')
plt.title('Distribution of Transaction Types')
plt.show()
```




```
plt.figure(figsize=(8,6))
data['amount'].value_counts().sort_values(ascending=False).head().plot(kind='bar') #Just
plt.xlabel('Amount')
plt.ylabel('Count')
plt.title('Distribution of Amount ')
plt.show()
```

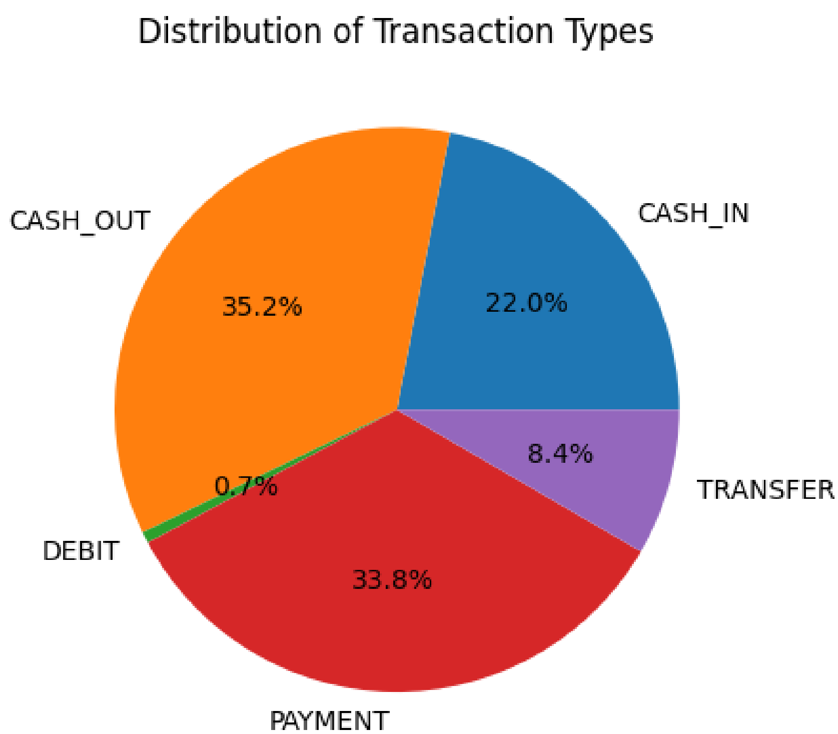


10000000.00 occurs 3207, Is this a default value or a system generate d value or are others outliers?

```
counts = data.groupby('type').count()['amount']
print(counts)
```

```
type
CASH_IN      1399284
CASH_OUT     2237500
DEBIT          41432
PAYMENT      2151495
TRANSFER      532909
Name: amount, dtype: int64
```

```
plt.pie(counts, labels=counts.index, autopct='%1.1f%%')
plt.title('Distribution of Transaction Types')
plt.show()
```

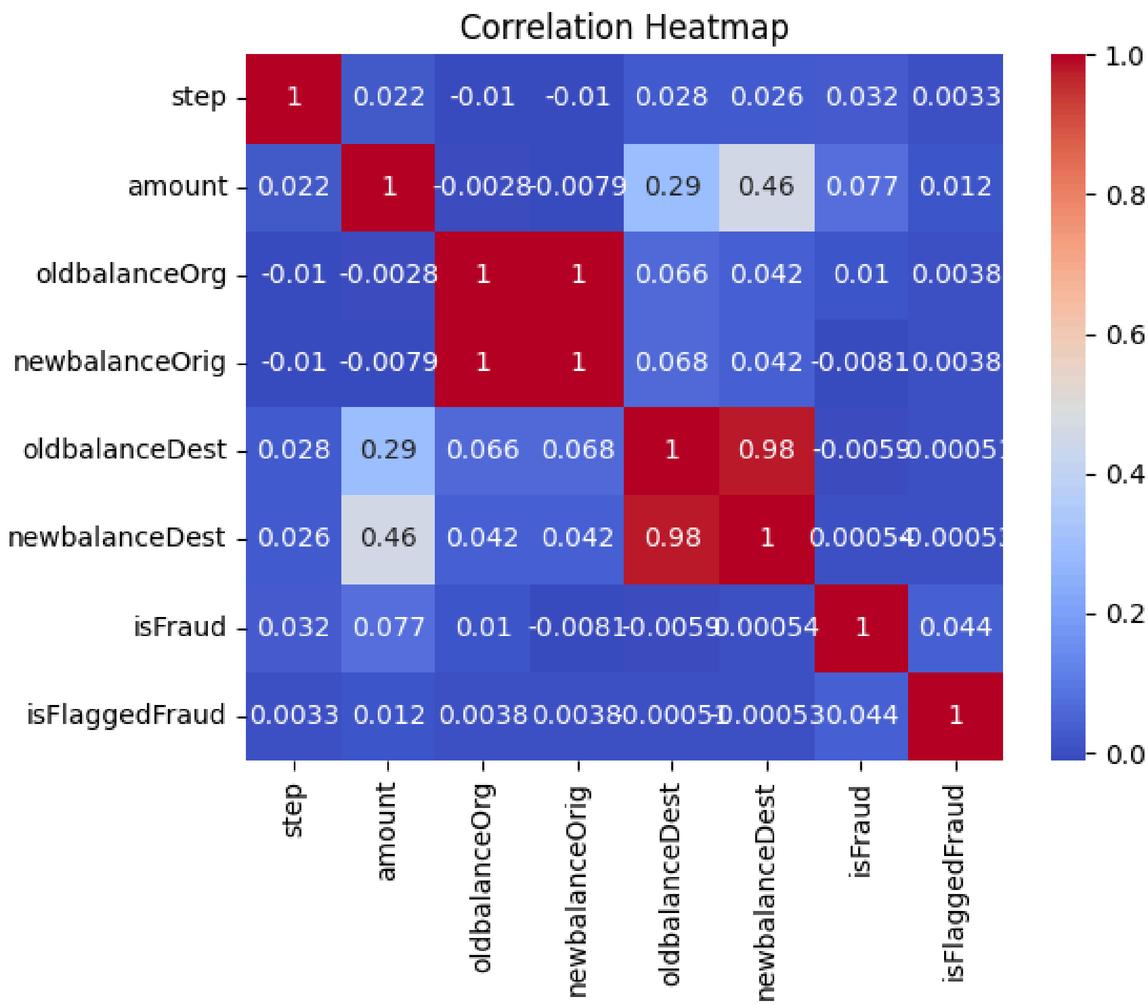



```
# Cash out , cash in and payment may have more fraud possiblities than others
#Cashin - depotsit into an acc, cashout, payment - move out of account, debit - atm with
```

```
data.groupby(['type','isFraud']).count()
```

type	isFraud	step	amount	nameOrig	oldbalanceOrg	newbalanceOrig	name
CASH_IN	0	1399284	1399284	1399284	1399284	1399284	13
CASH_OUT	0	2233384	2233384	2233384	2233384	2233384	22
	1	4116	4116	4116	4116	4116	
DEBIT	0	41432	41432	41432	41432	41432	
PAYMENT	0	2151495	2151495	2151495	2151495	2151495	21
TRANSFER	0	528812	528812	528812	528812	528812	5
	1	4097	4097	4097	4097	4097	

```
#Fraud transaction present in cash_out and transfer!!! These may be mostly social engine

numeric_cols = data.select_dtypes(include=['int','float']).columns
numeric_data = data[numeric_cols]

correlation_mat= numeric_data.corr()


sns.heatmap(correlation_mat, annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```




```
# Analyzing correlation marix -- Amount has more common with new_Balance_Dest which make
# oldbalanceOrigin and newBalance Orig is almost 1 which make sense because oldOrigin is

#Is FRaud(dependent) and Transaction Amount(independent) has large amount of linearity(m


#IsFraud and isFlaggedFrayd while an association exists, it's moderate
```

```
#We can drop oldBalanceOrig and NewBalance_orig, oldBalanceDest, and newBalance Dest

data.drop(['oldbalanceOrg','oldbalanceDest','newbalanceOrig','newbalanceDest', 'nameDest
```

```
data.head() #data way cleaner
```

	step	type	amount	isFraud	isFlaggedFraud
0	1	PAYMENT	9839.64	0	0
1	1	PAYMENT	1864.28	0	0
2	1	TRANSFER	181.00	1	0
3	1	CASH_OUT	181.00	1	0
4	1	PAYMENT	11668.14	0	0

```
from sklearn. preprocessing import LabelEncoder, StandardScaler
le = LabelEncoder()
data['type'] = le.fit_transform(data['type'])
```

```
X = data.drop('isFraud', axis=1)
y = data['isFraud']
```

```
sc = StandardScaler()
X = sc.fit_transform(X)
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42
```

ML Modelling

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score, cla
```

```
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
y_pred = lr_model.predict(X_test)
```

```
accuracy_lr = accuracy_score(y_test, y_pred)
precision_lr = precision_score(y_test, y_pred)
recall_lr = recall_score(y_test, y_pred)
classification_report_lr = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy_lr}")
print(f"Precision: {precision_lr}")
print(f"Recall: {recall_lr}")
print("Classification Report:")
print(classification_report_lr)
```

```
Accuracy: 0.9987086032693031
Precision: 0.1590909090909091
Recall: 0.002874743326488706
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906351
           1       0.16      0.00      0.01      2435

    accuracy                           1.00   1908786
   macro avg       0.58      0.50      0.50   1908786
weighted avg       1.00      1.00      1.00   1908786
```

Logistic model has a very high accuracy in modelling the data of 99.87%, however it's precision and recall of positive classes are very low at 16% and 0.29%, which means it struggles to detect fraud

```
#Logistic Regression we can say underfits or is unable to classify fraud, we can try neu
#and cannot be a good use for detecting fraud or lesser data, so let's check out decisio
```

```
from sklearn.tree import DecisionTreeClassifier

dtc = DecisionTreeClassifier(max_depth=20)
dtc.fit(X_train, y_train)

y_pred = dtc.predict(X_test)
```

```
accuracy_dtc = accuracy_score(y_test, y_pred)
precision_dtc = precision_score(y_test, y_pred)
recall_dtc = recall_score(y_test, y_pred)
classification_report_dtc = classification_report(y_test, y_pred)

print("Decision Trees")
print(f"Accuracy : {accuracy_dtc}")
print(f"Precision: {precision_dtc}")
print(f"Recall: {recall_dtc}")
print("Classification Report:")
print(classification_report_dtc)
```

```
Decision Trees
Accuracy : 0.998973693227004
Precision: 0.6619047619047619
Recall: 0.3995893223819302
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906351
           1       0.66      0.40      0.50      2435

    accuracy                           1.00   1908786
   macro avg       0.83      0.70      0.75   1908786
weighted avg       1.00      1.00      1.00   1908786
```

```
### As we can see, decision tree performs amazingly it's able to give a accuracy of 0.99
## and 40% recall
```

```
# plt.figure()

# tree.plot_tree(dtc,
#                    feature_names = data.columns,
#                    filled = True)
# plt.savefig('testfig.svg', format='svg')
```

Start coding or generate with AI.

```
<Figure size 640x480 with 0 Axes>
```

The above tree is pretty messy (As there are 100 estimators)

```
# What about random forests?

from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier(n_estimators = 10, max_depth=3)
rfc.fit(X_train, y_train)

y_pred = rfc.predict(X_test)
```

```
accuracy_rfc = accuracy_score(y_test, y_pred)
precision_rfc = precision_score(y_test, y_pred)
recall_rfc = recall_score(y_test, y_pred)
classification_report_rfc = classification_report(y_test, y_pred)

print("Random Forest")
print(f"Accuracy : {accuracy_rfc}")
print(f"Precision: {precision_rfc}")
print(f"Recall: {recall_rfc}")
print("Classification Report:")
print(classification_report_rfc)
```

```
Random Forest
Accuracy : 0.9988065712971491
Precision: 1.0
Recall: 0.06447638603696099
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00   1906351
           1       1.00      0.06      0.12      2435

    accuracy                           1.00   1908786
   macro avg       1.00      0.53      0.56   1908786
weighted avg       1.00      1.00      1.00   1908786
```

```
!pip install dtreeviz
```

```
Collecting dtreeviz
  Downloading dtreeviz-2.2.2-py3-none-any.whl (91 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 91.8/91.8 kB 2.5 MB/s eta 0:00:00
Requirement already satisfied: graphviz>=0.9 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (fro
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: colour in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: pytest in /usr/local/lib/python3.10/dist-packages (fr
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-pa
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dis
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-pack
Requirement already satisfied: iniconfig in /usr/local/lib/python3.10/dist-packages
Requirement already satisfied: pluggy<2.0,>=0.12 in /usr/local/lib/python3.10/dist-p
Requirement already satisfied: exceptiongroup>=1.0.0rc8 in /usr/local/lib/python3.10
Requirement already satisfied: tomli>=1.0.0 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.10/dist-packag
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packa
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dis
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (
```

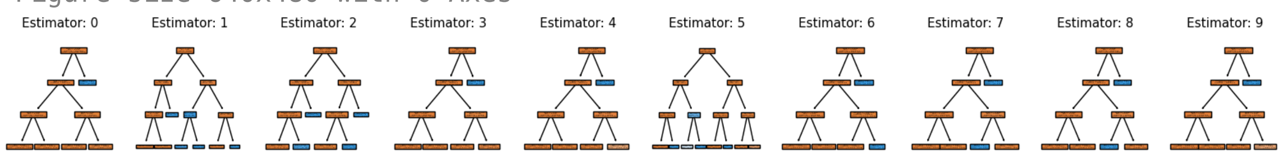
```
Installing collected packages: dtreeviz
Successfully installed dtreeviz-2.2.2
```

```
len(rfc.estimators_)
```

```
10
```

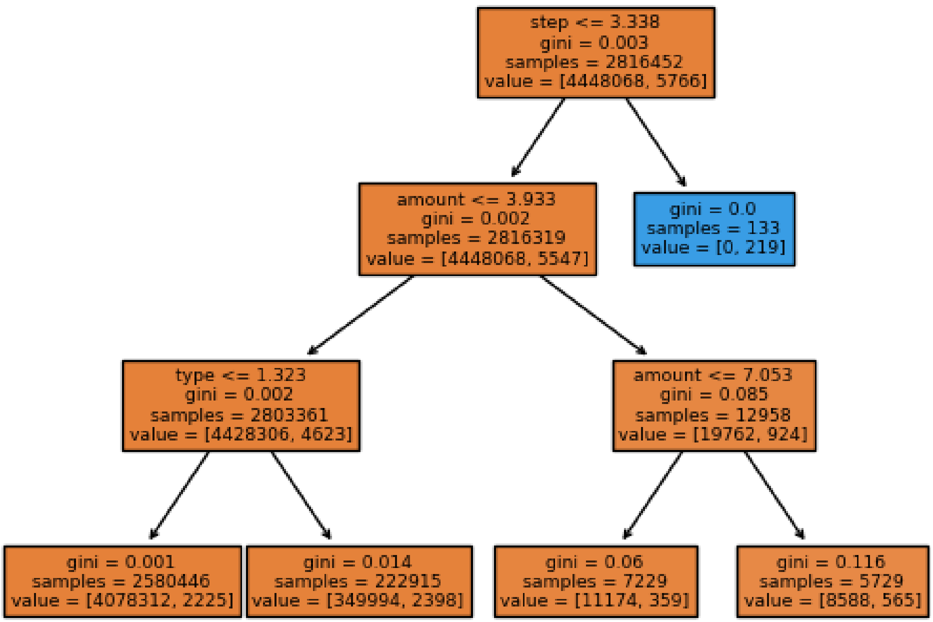
```
from sklearn import tree
plt.figure()
fig, axes = plt.subplots(nrows = 1,ncols = 10,figsize = (20,2))
for index in range(0, 10):
    tree.plot_tree(rfc.estimators_[index],
                   feature_names = data.columns,
                   filled = True,
                   ax = axes[index]);

    axes[index].set_title('Estimator: ' + str(index), fontsize = 11)

plt.savefig('random_forest.svg', format='svg')
```

```
<Figure size 640x480 with 0 Axes>
```




```
tree.plot_tree(rfc.estimators_[3],
                   feature_names = data.columns,
                   filled = True)
```

```
[Text(0.625, 0.875, 'step <= 3.338\ngini = 0.003\nsamples = 2816452\nvalue =
[4448068, 5766]'),
 Text(0.5, 0.625, 'amount <= 3.933\ngini = 0.002\nsamples = 2816319\nvalue =
[4448068, 5547]'),
 Text(0.25, 0.375, 'type <= 1.323\ngini = 0.002\nsamples = 2803361\nvalue =
[4428306, 4623]'),
 Text(0.125, 0.125, 'gini = 0.001\nsamples = 2580446\nvalue = [4078312, 2225]'),
 Text(0.375, 0.125, 'gini = 0.014\nsamples = 222915\nvalue = [349994, 2398]'),
 Text(0.75, 0.375, 'amount <= 7.053\ngini = 0.085\nsamples = 12958\nvalue =
[19762, 924]'),
 Text(0.625, 0.125, 'gini = 0.06\nsamples = 7229\nvalue = [11174, 359]'),
 Text(0.875, 0.125, 'gini = 0.116\nsamples = 5729\nvalue = [8588, 565]'),
 Text(0.75, 0.625, 'gini = 0.0\nsamples = 133\nvalue = [0, 219]')]
```




```
performance = pd.DataFrame({
    'models': [ 'Logistic Regression', 'Decision Tree', 'Random Forest'],
    'accuracy': [accuracy_lr, accuracy_dtc, accuracy_rfc],
    'precision': [precision_lr, precision_dtc, precision_rfc],
    'recall': [recall_lr, recall_dtc, recall_rfc]
})

performance
```

	models	accuracy	precision	recall
0	Logistic Regression	0.998709	0.159091	0.002875
1	Decision Tree	0.998974	0.661905	0.399589
2	Random Forest	0.998807	1.000000	0.064476

Next steps: Generate code with performance View recommended plots

widget for display

```
!pip install gradio
```

```
ERROR: pip's dependency resolver does not currently take into account all the pac
spacy 3.7.4 requires typer<0.10.0,>=0.3.0, but you have typer 0.12.3 which is inc
weasel 0.3.4 requires typer<0.10.0,>=0.3.0, but you have typer 0.12.3 which is ir
Successfully installed aiofiles-23.2.1 dnspython-2.6.1 email_validator-2.1.1 fast
```

```
data.describe()
```

	step	type	amount	isFraud	isFlaggedFraud
count	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06	6.362620e+06
mean	2.433972e+02	1.714150e+00	1.798619e+05	1.290820e-03	2.514687e-06
std	1.423320e+02	1.350117e+00	6.038582e+05	3.590480e-02	1.585775e-03
min	1.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
25%	1.560000e+02	1.000000e+00	1.338957e+04	0.000000e+00	0.000000e+00
50%	2.390000e+02	1.000000e+00	7.487194e+04	0.000000e+00	0.000000e+00
75%	3.350000e+02	3.000000e+00	2.087215e+05	0.000000e+00	0.000000e+00
max	7.430000e+02	4.000000e+00	9.244552e+07	1.000000e+00	1.000000e+00

```
import gradio as gr

def cleanup(var):
  return "True" if var == "1" else "False"

def greet(step,type1,amount,isFlaggedFraud):
    isFlaggedFraud = 1 if isFlaggedFraud == "True" else 0
    type1 = le.transform([type1])[0]
    output = sc.transform([[step,type1,amount,isFlaggedFraud]])
    dtc_output = dtc.predict(output)[0]
    rfc_output = rfc.predict(output)[0]
    lr_output = lr_model.predict(output)[0]
    output = "Decision Tree says "+ cleanup(str(dtc_output))+" and Random Forest says "+

    return output

demo = gr.Interface(
    fn=greet,
    inputs=[gr.Slider(1,743),
    gr.Radio(["CASH_IN","CASH_OUT","DEBIT","PAYMENT","TRANSFER"]),
    "number",
    gr.Radio(["True","False"])
    ],
    outputs=["text"]
)

demo.launch()
```

```
Setting queue=True in a Colab notebook requires sharing enabled. Setting `share=

Colab notebook detected. To show errors in colab notebook, set debug=True in lau
Running on public URL: https://d786240e6270e9dc0d.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades
```