```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
df=pd.read_csv('/Fraud.csv')
```

```
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalan
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.00	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.00	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.00	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.00	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.00	
...	...	...	...	...	...	...	...	...	
627839	34	CASH_OUT	181122.71	C651585324	56404.0	0.00	C317509232	0.00	18
627840	34	PAYMENT	5412.56	C415515195	1062.0	0.00	M1995069219	0.00	
627841	34	CASH_OUT	95385.70	C1170559510	9269.0	0.00	C1427089316	27051.23	12
627842	34	PAYMENT	1116.95	C1354740357	8097.0	6980.05	M1783686963	0.00	
627843	34	TRANSFER	1911467.49	C1319626261	8214.0	0.00	C1756573049	545108.00	

627844 rows × 11 columns

```
df.head()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	0.0
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.0	0.0
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	0.0
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.0	0.0
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.0	0.0

```
df.tail()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
627839	34	CASH_OUT	181122.71	C651585324	56404.0	0.00	C317509232	0.00	181122.71	0.0	0.0
627840	34	PAYMENT	5412.56	C415515195	1062.0	0.00	M1995069219	0.00	0.00	0.0	0.0
627841	34	CASH_OUT	95385.70	C1170559510	9269.0	0.00	C1427089316	27051.23	122436.93	0.0	0.0
627842	34	PAYMENT	1116.95	C1354740357	8097.0	6980.05	M1783686963	0.00	0.00	0.0	0.0
627843	34	TRANSFER	1911467.49	C1319626261	8214.0	0.00	C1756573049	545108.00	NaN	NaN	NaN

```
df.isnull()
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
0	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...
627839	False	False	False	False	False	False	False	False	False	False	False
627840	False	False	False	False	False	False	False	False	False	False	False
627841	False	False	False	False	False	False	False	False	False	False	False
627842	False	False	False	False	False	False	False	False	False	False	False
627843	False	False	False	False	False	False	False	False	True	True	True

627844 rows × 11 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 627844 entries, 0 to 627843
Data columns (total 11 columns):
 #   Column          Non-Null Count   Dtype
---  ------          --------------   -----
 0   step            627844 non-null  int64
 1   type            627844 non-null  object
 2   amount          627844 non-null  float64
 3   nameOrig        627844 non-null  object
 4   oldbalanceOrg   627844 non-null  float64
 5   newbalanceOrig  627844 non-null  float64
 6   nameDest        627844 non-null  object
 7   oldbalanceDest  627844 non-null  float64
 8   newbalanceDest  627843 non-null  float64
 9   isFraud         627843 non-null  float64
 10  isFlaggedFraud  627843 non-null  float64
dtypes: float64(7), int64(1), object(3)
memory usage: 52.7+ MB
```

```
df.mean()
```

```
df.count()
```

```
df.count
```

```
pandas.core.frame.DataFrame.count
def count(axis: Axis=0, level: Level=None, numeric_only: bool=False)
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py
Count non-NA cells for each column or row.

The values `None`, `NaN`, `NaT`, and optionally `numpy.inf` (depending
on `pandas.options.mode.use_inf_as_na`) are considered NA.
```

```
df.columns
```

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

```
df.isFlaggedFraud
```

```
0         0.0
1         0.0
2         0.0
3         0.0
4         0.0
          ...
627839    0.0
627840    0.0
627841    0.0
627842    0.0
627843    NaN
Name: isFlaggedFraud, Length: 627844, dtype: float64
```

```
df.add_prefix
```

```
pandas.core.generic.NDFrame.add_prefix
def add_prefix(prefix: str) -> NDFrameT
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py
Prefix labels with string `prefix`.

For Series, the row labels are prefixed.
For DataFrame, the column labels are prefixed.
```

```
df.all
```

```
pandas.core.generic.NDFrame._add_numeric_operations.<locals>.all
def all(axis=0, bool_only=None, skipna=True, level=None, **kwargs)
```

```
   col1   col2
0  True   True
1  True  False

Default behaviour checks if values in each column all return True.

>>> df.all()
```

df.bool

```
pandas.core.generic.NDFrame.bool
def bool() -> bool_t
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py
Return the bool of a single element Series or DataFrame.

This must be a boolean scalar value, either True or False. It will raise a
ValueError if the Series or DataFrame does not have exactly 1 element, or that
element is not boolean (integer values 0 and 1 will also raise an exception).
```

df.convert_dtypes

```
pandas.core.generic.NDFrame.convert_dtypes
def convert_dtypes(infer_objects: bool_t=True, convert_string: bool_t=True, convert_integer:
bool_t=True, convert_boolean: bool_t=True, convert_floating: bool_t=True) -> NDFrameT
```

```
/usr/local/lib/python3.10/dist-packages/pandas/core/generic.py
Convert columns to best possible dtypes using dtypes supporting ``pd.NA``.

.. versionadded:: 1.0.0

Parameters
```

df.drop_duplicates

```
pandas.core.frame.DataFrame.drop_duplicates
def drop_duplicates(subset: Hashable | Sequence[Hashable] | None=None, keep: Literal['first',
'last', False]='first', inplace: bool=False, ignore_index: bool=False) -> DataFrame | None
```

```
To remove duplicates and keep last occurrences, use   keep  .

>>> df.drop_duplicates(subset=['brand', 'style'], keep='last')
    brand style  rating
1  Yum Yum   cup     4.0
2  Indomie   cup     3.5
4  Indomie  pack     5.0
```

column10_duplicates=df.drop_duplicates("isFraud")

column10_duplicates

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalar
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.0	
627843	34	TRANSFER	1911467.49	C1319626261	8214.0	0.00	C1756573049	545108.0	

Next steps: Generate code with column10_duplicates View recommended plots

```
columnn11_duplicates=df.drop_duplicates("isFlaggedFraud")
```

```
columnn11_duplicates
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalar
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.0	
627843	34	TRANSFER	1911467.49	C1319626261	8214.0	0.00	C1756573049	545108.0	

Next steps: Generate code with columnn11_duplicates View recommended plots

```
column10_duplicates=df.drop_duplicates("newbalanceDest")
```

```
column10_duplicates
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalar
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.00	
9	1	DEBIT	5337.77	C712410124	41720.0	36382.23	C195600860	41898.00	4
10	1	DEBIT	9644.94	C1900366749	4465.0	0.00	C997608398	10845.00	15
15	1	CASH_OUT	229133.94	C905080434	15325.0	0.00	C476402209	5083.00	5
21	1	DEBIT	9302.79	C1566511282	11299.0	1996.21	C1973538135	29832.00	1
...	...	...	...	...	...	...	...	...	
627836	34	CASH_OUT	71502.29	C1640594013	202393.0	130890.71	C476112429	149720.81	22
627837	34	TRANSFER	453962.90	C776080832	31.0	0.00	C762277956	838214.27	129
627839	34	CASH_OUT	181122.71	C651585324	56404.0	0.00	C317509232	0.00	18
627841	34	CASH_OUT	95385.70	C1170559510	9269.0	0.00	C1427089316	27051.23	12
627843	34	TRANSFER	1911467.49	C1319626261	8214.0	0.00	C1756573049	545108.00	

225320 rows × 11 columns

```
df
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalai
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.00	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.00	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.00	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.00	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.00	
...	...	...	...	...	...	...	...	...	
627839	34	CASH_OUT	181122.71	C651585324	56404.0	0.00	C317509232	0.00	18
627840	34	PAYMENT	5412.56	C415515195	1062.0	0.00	M1995069219	0.00	
627841	34	CASH_OUT	95385.70	C1170559510	9269.0	0.00	C1427089316	27051.23	12
627842	34	PAYMENT	1116.95	C1354740357	8097.0	6980.05	M1783686963	0.00	
627843	34	TRANSFER	1911467.49	C1319626261	8214.0	0.00	C1756573049	545108.00	

627844 rows × 11 columns

```
df.dropna
```

pandas.core.frame.DataFrame.dropna

```
def dropna(axis: Axis=0, how: str | NoDefault=no_default, thresh: int | NoDefault=no_default,
subset: IndexLabel=None, inplace: bool=False) -> DataFrame | None
```

/usr/local/lib/python3.10/dist-packages/pandas/core/frame.py

```
Remove missing values.

See the :ref:`User Guide <missing_data>` for more on which values are
considered missing, and how to work with missing data.
```

```
df.dropna(subset=["isFraud"])
```

	step	type	amount	nameOrig	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanc
0	1	PAYMENT	9839.64	C1231006815	170136.0	160296.36	M1979787155	0.00	
1	1	PAYMENT	1864.28	C1666544295	21249.0	19384.72	M2044282225	0.00	
2	1	TRANSFER	181.00	C1305486145	181.0	0.00	C553264065	0.00	
3	1	CASH_OUT	181.00	C840083671	181.0	0.00	C38997010	21182.00	
4	1	PAYMENT	11668.14	C2048537720	41554.0	29885.86	M1230701703	0.00	
...	...	...	...	...	...	...	...	...	
627838	34	PAYMENT	14672.30	C1576429270	36705.0	22032.70	M208142191	0.00	
627839	34	CASH_OUT	181122.71	C651585324	56404.0	0.00	C317509232	0.00	181
627840	34	PAYMENT	5412.56	C415515195	1062.0	0.00	M1995069219	0.00	
627841	34	CASH_OUT	95385.70	C1170559510	9269.0	0.00	C1427089316	27051.23	122
627842	34	PAYMENT	1116.95	C1354740357	8097.0	6980.05	M1783686963	0.00	

627843 rows × 11 columns

```
df.isnull().sum()
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    1
isFraud           1
isFlaggedFraud    1
dtype: int64
```

```
df.dropna(thresh=0.8, axis=1, inplace=True)
```

```
df["outlier"] = 0
```

```
# Calculate Q1, Q2, Q3
Q1 = df["amount"].quantile(0.25)
Q2 = df["amount"].quantile(0.5)
Q3 = df["amount"].quantile(0.75)

# Calculate IQR
IQR = Q3 - Q1

# Identify outliers
df["outlier"] = np.where(
    (df["amount"] < Q1 - 1.5 * IQR) | (df["amount"] > Q3 + 1.5 * IQR),
    1,
    0,
)
```

```
missing_values = df.isnull().sum()

# Print the number of missing values in each column
print(missing_values)
```

```
step              0
type              0
amount            0
nameOrig          0
oldbalanceOrg     0
newbalanceOrig    0
nameDest          0
oldbalanceDest    0
newbalanceDest    1
isFraud           1
isFlaggedFraud    1
outlier           0
dtype: int64
```

```
# Drop columns with a high percentage of missing values
df.dropna(thresh=0.8, axis=1, inplace=True)

# Impute missing values in remaining columns
df["isFlaggedFraud"].fillna(df["isFlaggedFraud"].mean(), inplace=True)
```

```
# Drop columns with a high percentage of missing values
df.dropna(thresh=0.8, axis=1, inplace=True)

# Impute missing values in remaining columns
df["newbalanceDest"].fillna(df["newbalanceDest"].mean(), inplace=True)
```

```
# Drop columns with a high percentage of missing values
df.dropna(thresh=0.8, axis=1, inplace=True)

# Impute missing values in remaining columns
df["isFraud"].fillna(df["isFraud"].mean(), inplace=True)
```

```
df["type"] = df["type"].astype("str")
```

```
df["type"] = df["type"].replace("CASH_IN", 0)
```

```
df["type"] = df["type"].astype("str")
```

```
df["type"] = df["type"].replace("PAYMENT", 0)
```

```
df["type"] = df["type"].astype("str")
df["type"] = df["type"].replace("TRANSFER", 0)
```

```
df["type"] = df["type"].astype("str")
df["type"] = df["type"].replace("CASH_OUT", 0)
```

```
Start coding or generate with AI.
```

Start coding or generate with AI.

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

# Load the data
df = pd.read_csv("/Fraud.csv")

# Preprocess the data
df.dropna(inplace=True)
df["outliers"] = np.where(
    (df["amount"] < Q1 - 1.5 * IQR) | (df["amount"] > Q3 + 1.5 * IQR),
    1,
    0,
)

# Split the data into training and testing sets
X = df.drop(["isFraud", "outliers"], axis=1)
y = df["isFraud"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train the model
model = RandomForestClassifier()
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)

from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

Start coding or generate with AI.

```
df = pd.read_csv("/Fraud.csv")

# Calculate correlation between each variable and the target variable
correlations = df.corr()["isFraud"]

# Select variables with a strong correlation with the target variable
selected_variables = correlations[abs(correlations) > 0.5].index.tolist()
```

```
<ipython-input-71-2f178dbfbd22>:6: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid
  correlations = df.corr()["isFraud"]
```

```
# Examine the relationship between pairs of variables
for i in selected_variables:
    for j in selected_variables:
        if i != j:
            sns.scatterplot(x=df[i], y=df[j])
            plt.show()
```

```
df = pd.read_csv("/Fraud.csv")

# Select features for the model
selected_features = ["amount", "oldbalanceOrg", "newbalanceOrig", "oldbalanceDest", "newbalanceDest"]

# Define X and y
X = df[selected_features]
y = df["isFraud"]

# Split the data into training and testing sets
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=42)
```

```
# Build a model using the selected variables
X = df[selected_variables]
y = df["isFraud"]

model = RandomForestClassifier()
model.fit(X, y)

# Evaluate the model
y_pred = model.predict(X)

from sklearn.metrics import classification_report

print(classification_report(y, y_pred))
```

```
mse = (y_test-y_train)**2
print(f"MSE: {mse.mean():0.2f} (+/- {mse.std():0.2f})")
```

```
MSE: nan (+/- nan)
```

```
mae = np.abs(y_test-y_train)
print(f"MAE: {mae.mean():0.2f} (+/- {mae.std():0.2f})")
```

```
MAE: nan (+/- nan)
```

```
mse = (y_test-y_train)**2
rmse = np.sqrt(mse.mean())
print(f"RMSE: {rmse:0.2f}")
```

```
RMSE: nan
```

Start coding or generate with AI.

```
df = pd.read_csv("/Fraud.csv")

# Select features for the model
selected_features = ["amount", "oldbalanceOrg", "newbalanceOrig", "oldbalanceDest", "newbalanceDest"]

# Split the data into training and testing sets
from sklearn.model_selection import train_test_split
```