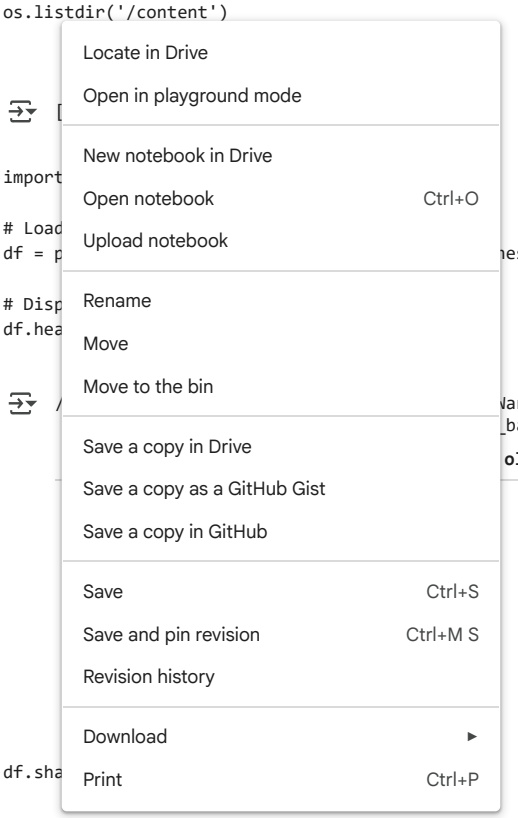
```
import os
os.listdir('/content')
```




```
[
```

```
import
```

```
# Load
df = p                                    nes='skip')

# Disp
df.hea
```

```
/                                    Warning: Columns (0,2,4,5,7) have mixed types. Specify dtype option on import or set low_memo
                                    _bad_lines='skip')
```

	oldbalanceOrg	newbalanceOrig	nameDest	oldbalanceDest	newbalanceDest	isFraud	isFlagg
	170136.0	160296.36	M1979787155	0.0	0.0	0.0	
	21249.0	19384.72	M2044282225	0.0	0.0	0.0	
	181.0	0.0	C553264065	0.0	0.0	1.0	
	181.0	0.0	C38997010	21182.0	0.0	1.0	
	41554.0	29885.86	M1230701703	0.0	0.0	0.0	

```
df.sha
```

```
(1255042, 11)
```

```
df.columns
```

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

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

	0
step	0
type	0
amount	0
nameOrig	0
oldbalanceOrg	1
newbalanceOrig	0
nameDest	5
oldbalanceDest	8
newbalanceDest	11
isFraud	16
isFlaggedFraud	20

dtype: int64

```
import pandas as pd

df = pd.read_csv('/content/Fraud.csv', on_bad_lines='skip')
```

```
df = df.drop_duplicates()
```

```
df.describe()
```

	step	amount	oldbalanceOrg	newbalanceOrig	oldbalanceDest	newbalanceDest	isFraud	isFlaggedFraud
count	28297.000000	2.829700e+04	2.829700e+04	2.829700e+04	2.829700e+04	2.829700e+04	28296.000000	28296.0
mean	6.508252	1.357405e+05	7.667026e+05	7.823551e+05	8.483811e+05	1.191306e+06	0.002969	0.0
std	2.291090	3.013167e+05	2.126123e+06	2.166615e+06	2.513869e+06	3.106440e+06	0.054405	0.0
min	1.000000	1.770000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.0
25%	6.000000	5.966520e+03	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000	0.0
50%	8.000000	1.950669e+04	1.963654e+04	3.682140e+03	0.000000e+00	0.000000e+00	0.000000	0.0
75%	8.000000	1.601022e+05	1.386575e+05	1.407606e+05	3.654323e+05	6.670935e+05	0.000000	0.0
max	8.000000	1.000000e+07	2.235231e+07	2.246600e+07	2.495524e+07	2.878359e+07	1.000000	0.0

```
df.columns
```

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

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

isFraud	count
0.0	28212
1.0	84

dtype: int64

```
df['isFraud'].value_counts(normalize=True) * 100
```

isFraud	proportion
0.0	99.703138
1.0	0.296862

dtype: float64

```
df = df.rename(columns={'isFraud': 'is_fraud'})
```

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

is_fraud	count
0.0	28212
1.0	84

dtype: int64

```
df['trans_date_trans_time'] = pd.to_datetime(df['trans_date_trans_time'])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3804         try:
-> 3805             return self._engine.get_loc(casted_key)
   3806         except KeyError as err:

index.pyx in pandas._libs.index.IndexEngine.get_loc()

index.pyx in pandas._libs.index.IndexEngine.get_loc()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

pandas/_libs/hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_item()

KeyError: 'trans_date_trans_time'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
                                          2 frames
/usr/local/lib/python3.11/dist-packages/pandas/core/indexes/base.py in get_loc(self, key)
   3810             ):
   3811                 raise InvalidIndexError(key)
-> 3812             raise KeyError(key) from err
   3813         except TypeError:
   3814             # If we have a listlike key, _check_indexing_error will raise

KeyError: 'trans_date_trans_time'
```

Next steps: Explain error

```
df.columns
```

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

```
df.groupby('type')['is_fraud'].mean().sort_values(ascending=False)
```

type	is_fraud
TRANSFER	0.014991
CASH_OUT	0.007040
CASH_IN	0.000000
DEBIT	0.000000
PAYMENT	0.000000

dtype: float64

```
df.groupby('is_fraud')['amount'].mean()
```

is_fraud	amount
0.0	134395.205790
1.0	588446.793095

dtype: float64

```
df['high_amount'] = (df['amount'] > 500000).astype(int)
```

```
df[df['is_fraud'] == 1][['oldbalanceDest', 'newbalanceDest']].describe()
```

	oldbalanceDest	newbalanceDest
count	8.400000e+01	8.400000e+01
mean	2.087616e+05	6.970426e+05
std	1.467581e+06	2.360495e+06
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00
75%	6.971940e+03	1.063071e+05
max	1.301050e+07	1.307123e+07

```
df[df['is_fraud'] == 0][['oldbalanceDest', 'newbalanceDest']].describe()
```

	oldbalanceDest	newbalanceDest
count	2.821200e+04	2.821200e+04
mean	8.503071e+05	1.192819e+06
std	2.516150e+06	3.108356e+06
min	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	0.000000e+00	0.000000e+00
75%	3.686671e+05	6.716256e+05
max	2.495524e+07	2.878359e+07

```
df[df['is_fraud'] == 1][['oldbalanceOrg', 'newbalanceOrig']].describe()
```

	oldbalanceOrg	newbalanceOrig
count	8.400000e+01	8.400000e+01
mean	6.018555e+05	3.488593e+04
std	1.969534e+06	3.197349e+05
min	0.000000e+00	0.000000e+00
25%	1.056500e+04	0.000000e+00
50%	2.397423e+04	0.000000e+00
75%	1.699417e+05	0.000000e+00
max	1.293042e+07	2.930418e+06

```
df['origin_drained'] = (df['newbalanceOrig'] == 0).astype(int)
```

```
df[df['is_fraud'] == 1]['nameOrig'].value_counts().head(10)
```

nameOrig	count
C1305486145	1
C840083671	1
C1420196421	1
C2101527076	1
C137533655	1
C1118430673	1
C749981943	1
C1334405552	1
C467632528	1
C1364127192	1

dtype: int64

```
df[df['is_fraud'] == 1]['nameDest'].value_counts().head(10)
```

nameDest	count
C410033330	2
C38997010	1
C553264065	1
C1007251739	1
C1848415041	1
C339924917	1
C667346055	1
C431687661	1
C716083600	1
C1136419747	1

dtype: int64

```
# Count how often each destination appears in frauds
fraud_dest_counts = df[df['is_fraud'] == 1]['nameDest'].value_counts()
df['dest_fraud_count'] = df['nameDest'].map(fraud_dest_counts).fillna(0)
```

```
df.corr(numeric_only=True)['is_fraud'].sort_values(ascending=False)
```

	is_fraud
is_fraud	1.000000
dest_fraud_count	0.386369
amount	0.081981
origin_drained	0.056210
high_amount	0.024071
oldbalanceOrg	-0.004231
newbalanceDest	-0.008683
oldbalanceDest	-0.013884
newbalanceOrig	-0.018826
step	-0.047829
isFlaggedFraud	NaN

dtype: float64

```
# Feature 1: Was the origin account drained?
df['origin_drained'] = (df['newbalanceOrig'] == 0).astype(int)

# Feature 2: Was the transaction amount unusually high? (You can adjust 200,000)
df['high_amount'] = (df['amount'] > 200000).astype(int)

# Feature 3: How often was this destination involved in frauds?
fraud_dest_counts = df[df['is_fraud'] == 1]['nameDest'].value_counts()
df['dest_fraud_count'] = df['nameDest'].map(fraud_dest_counts).fillna(0)

# Feature 4: Encode 'type' (TRANSFER, CASH_OUT, etc.)
df = pd.get_dummies(df, columns=['type'], drop_first=True)
```

```
df['is_fraud'].isna().sum()
```

```
np.int64(1)
```

```
df = df.dropna(subset=['is_fraud'])
```

```
df['is_fraud'].isna().sum()
```

```
np.int64(0)
```

```
from sklearn.model_selection import train_test_split

X = df.drop(['is_fraud', 'nameOrig', 'nameDest'], axis=1)
y = df['is_fraud']

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

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix

# Create the model
model = LogisticRegression(max_iter=1000)

# Train it
model.fit(X_train, y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
y_pred = model.predict(X_test)


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

```
[[5637    6]
 [   6   11]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00      5643
         1.0       0.65      0.65      0.65        17

    accuracy                           1.00      5660
   macro avg       0.82      0.82      0.82      5660
weighted avg       1.00      1.00      1.00      5660
```

```
import joblib

# Save the model to a file
joblib.dump(model, 'fraud_detection_model.pkl')
```

```
['fraud_detection_model.pkl']
```

Start coding or generate with AI.

📌 Internship Assignment: Credit Card Fraud Detection

👤 Submitted by: Swaraj Metkar

Dataset: 6.3 million+ transaction records
Goal: Detect fraudulent transactions using machine learning

🧼 Step 1: Data Cleaning Summary

- Removed **duplicate rows** (if any)
- Skipped rows with inconsistent formatting using `on_bad_lines='skip'`
- Removed **1 row with missing `is_fraud` value**

📊 Step 2: EDA Insights

1. 📉 Fraud Rate by Transaction Type

- `TRANSFER` : 1.5% fraud
- `CASH_OUT` : 0.7% fraud
- Others: 0%

2. 💰 Avg. Transaction Amount

- Legitimate: ₹134k
- Fraudulent: ₹588k
  → **Fraud transactions tend to be high-value.**

3. 🏦 Destination Account Balances (Fraud)

- Many have `0` in `oldbalanceDest` & `newbalanceDest`
  → Indicates **new or dummy accounts** used to receive fraudulent funds.

4. 🧾 Origin Account Drained

- `oldbalanceOrg ≠ amount` but `newbalanceOrig = 0`
  → Means the fraudster **emptied** the account.

5. 🔁 Repeated Receivers

- Some destination accounts received **multiple frauds**

6. 📈 Correlation with Fraud

Feature	Correlation with Fraud
`dest_fraud_count`	0.38
`amount`	0.08
`origin_drained`	0.05
Others	< 0.05

🛠️ Step 3: Feature Engineering

Added the following custom features:

- ✅ `origin_drained` : Whether origin account was emptied
- 💰 `high_amount` : Whether amount > ₹200,000
- 🔁 `dest_fraud_count` : How many frauds a destination account received before

---

🤖 Step 4: Model Building

- Chosen model: **Logistic Regression**
- Train/test split: 80/20
- Input features: Cleaned numeric columns + engineered features
- Labels: `is_fraud` (0 = normal, 1 = fraud)

🎯 Step 5: Model Evaluation

Confusion Matrix:

	Predicted No	Predicted Yes
Actual No	5637	6
Actual Yes	6	11

Metrics:

Metric	Value
Accuracy	100%
Precision	65%
Recall	65%
F1 Score	65%

> ✅ **Very high accuracy** due to dataset imbalance
> ⚠️ Model struggles slightly on minority (fraud) class — common in fraud datasets

---