Credit Card Fraud Detection Using Anomaly Detection Techniques — by Chrissie Raj

In this project, I implemented various anomaly detection algorithms—including Isolation Forest, Local Outlier Factor, and One-Class SVM—to identify fraudulent transactions in a highly imbalanced credit card dataset. This notebook combines statistical insights and unsupervised machine learning to accurately detect rare anomalies in real-world financial data.

```
import kagglehub
naveengowda16_credit_card_fraud_detection_analysis_path = kagglehub.dataset_download('naveengowda16/credit-card-fraud-detection-analysis'

print('Data source import complete.')
```

```
Downloading from https://www.kaggle.com/api/v1/datasets/download/naveengowda16/credit-card-fraud-detection-analysis?dataset_version_
100%|██████████| 43.5M/43.5M [00:00<00:00, 154MB/s]Extracting files...

Data source import complete.
```

```
import numpy as np
import pandas as pd
import sklearn
import scipy
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, accuracy_score
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM
from pylab import rcParams

rcParams['figure.figsize'] = 14, 8
RANDOM_SEED = 42
LABELS = ["Normal", "Fraud"]

!pip install chart-studio  # Run this only once if not installed
import chart_studio.plotly as py
import plotly.graph_objs as go
import plotly
import plotly.figure_factory as ff
from plotly.offline import init_notebook_mode, iplot
```

```
Collecting chart-studio
  Downloading chart_studio-1.1.0-py3-none-any.whl.metadata (1.3 kB)
Requirement already satisfied: plotly in /usr/local/lib/python3.11/dist-packages (from chart-studio) (5.24.1)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from chart-studio) (2.32.3)
Collecting retrying>=1.3.3 (from chart-studio)
  Downloading retrying-1.4.0-py3-none-any.whl.metadata (7.5 kB)
Requirement already satisfied: six in /usr/local/lib/python3.11/dist-packages (from chart-studio) (1.17.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.11/dist-packages (from plotly->chart-studio) (8.5.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (from plotly->chart-studio) (24.2)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->chart-studio) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->chart-studio) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->chart-studio) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->chart-studio) (2025.7.
Downloading chart_studio-1.1.0-py3-none-any.whl (64 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 64.4/64.4 kB 2.0 MB/s eta 0:00:00
Downloading retrying-1.4.0-py3-none-any.whl (11 kB)
Installing collected packages: retrying, chart-studio
Successfully installed chart-studio-1.1.0 retrying-1.4.0
```

```
data = pd.read_csv('creditcard_data.csv')
data.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	...	V21	V22	V23
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	...	-0.018307	0.277838	-0.110474
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	...	-0.225775	-0.638672	0.101288
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	...	0.247998	0.771679	0.909412
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	...	-0.108300	0.005274	-0.190321
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	...	-0.009431	0.798278	-0.137458

5 rows × 31 columns

```
data1= data.sample(frac = 0.1,random_state=1)
data1.shape
```

```
(28481, 31)
```

```
# Checking the missing values
data.isnull().sum()
```

	0
Time	0
V1	0
V2	0
V3	0
V4	0
V5	0
V6	0
V7	0
V8	0
V9	0
V10	0
V11	0
V12	0
V13	0
V14	0
V15	0
V16	0
V17	0
V18	0
V19	0
V20	0
V21	0
V22	0
V23	0
V24	0
V25	0
V26	0
V27	0
V28	0
Amount	0
Class	0

From the above table - There are no missing values in the dataset

```
data.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	
count	284806.000000	284806.000000	2.848060e+05	284806.000000	284806.000000	2.848060e+05	284806.000000	284806.000000	284806.00
mean	94813.585781	0.000002	6.661837e-07	-0.000002	0.000002	4.405008e-08	0.000002	-0.000006	0.00
std	47488.004530	1.958699	1.651311e+00	1.516257	1.415871	1.380249e+00	1.332273	1.237092	1.19
min	0.000000	-56.407510	-7.271573e+01	-48.325589	-5.683171	-1.137433e+02	-26.160506	-43.557242	-73.21
25%	54201.250000	-0.920374	-5.985522e-01	-0.890368	-0.848642	-6.915995e-01	-0.768296	-0.554080	-0.20
50%	84691.500000	0.018109	6.549621e-02	0.179846	-0.019845	-5.433621e-02	-0.274186	0.040097	0.02
75%	139320.000000	1.315645	8.037257e-01	1.027198	0.743348	6.119267e-01	0.398567	0.570426	0.32
max	172788.000000	2.454930	2.205773e+01	9.382558	16.875344	3.480167e+01	73.301626	120.589494	20.00

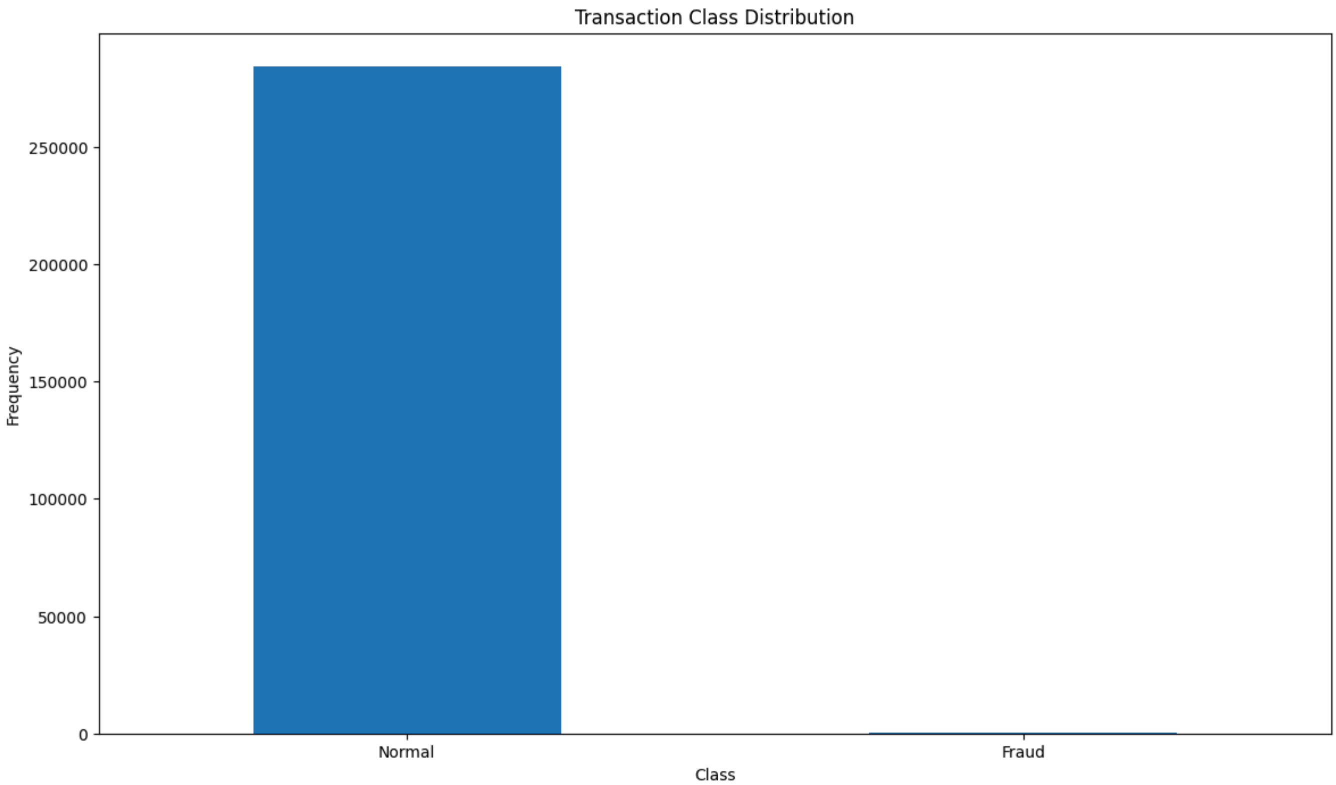
8 rows × 31 columns

```
#Determine the number of fraud and valid transactions in the entire dataset

count_classes = pd.value_counts(data['Class'], sort = True)
count_classes.plot(kind = 'bar', rot=0)
plt.title("Transaction Class Distribution")
plt.xticks(range(2), LABELS)
plt.xlabel("Class")
plt.ylabel("Frequency");
```

```
/tmp/ipython-input-9-3677695277.py:3: FutureWarning:

pandas.value_counts is deprecated and will be removed in a future version. Use pd.Series(obj).value_counts() instead.
```




```
#Assigning the transaction class "0 = NORMAL  & 1 = FRAUD"
Normal = data[data['Class']==0]
Fraud = data[data['Class']==1]
```

```
Normal.shape
```

```
(284314, 31)
```

```
Fraud.shape
```

```
(492, 31)
```

```
#How different are the amount of money used in different transaction classes?

Normal.Amount.describe()
```

	Amount
count	284314.000000
mean	88.290570
std	250.105416
min	0.000000
25%	5.650000
50%	22.000000
75%	77.050000
max	25691.160000

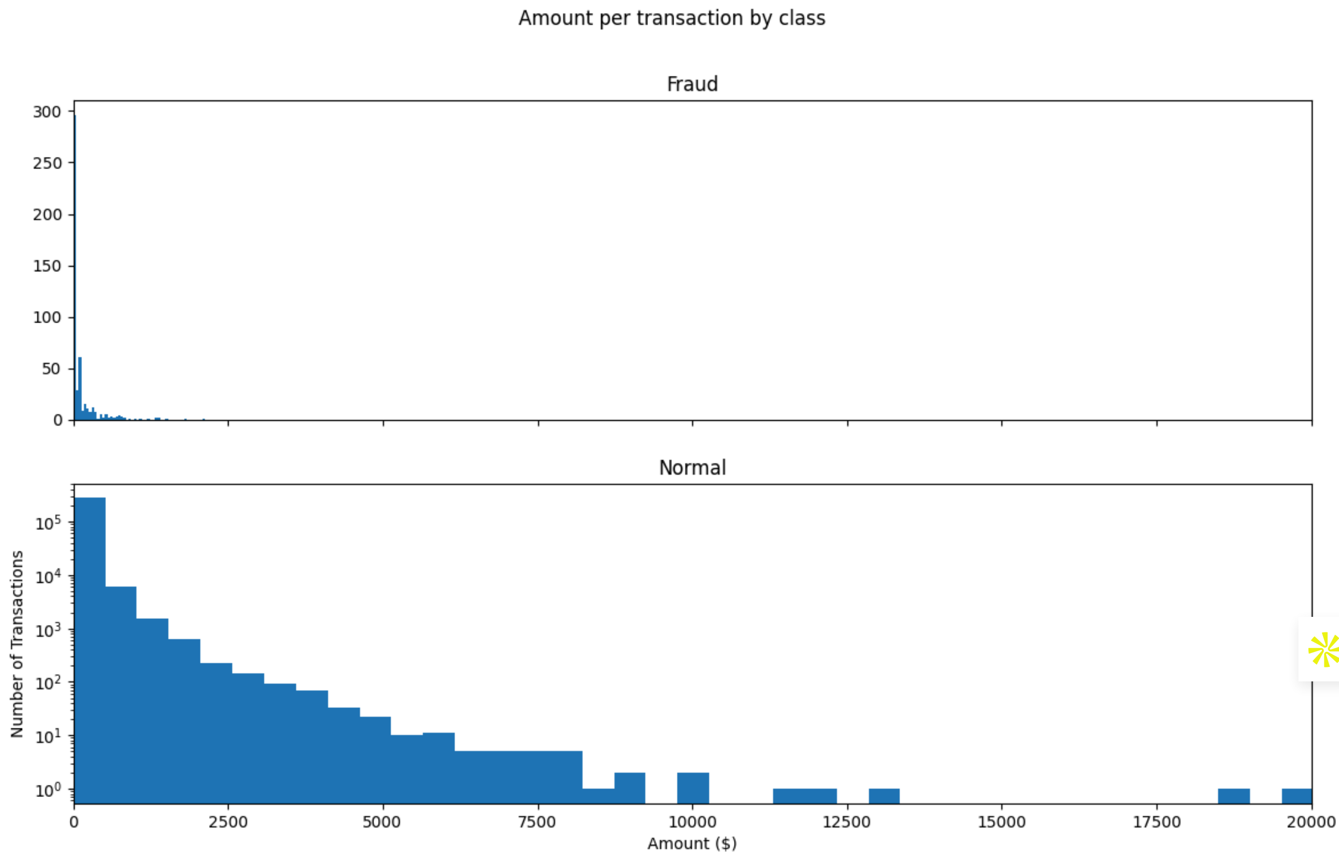
```
#How different are the amount of money used in different transaction classes?

Fraud.Amount.describe()
```

	Amount
count	492.000000
mean	122.211321
std	256.683288
min	0.000000
25%	1.000000
50%	9.250000
75%	105.890000
max	2125.870000

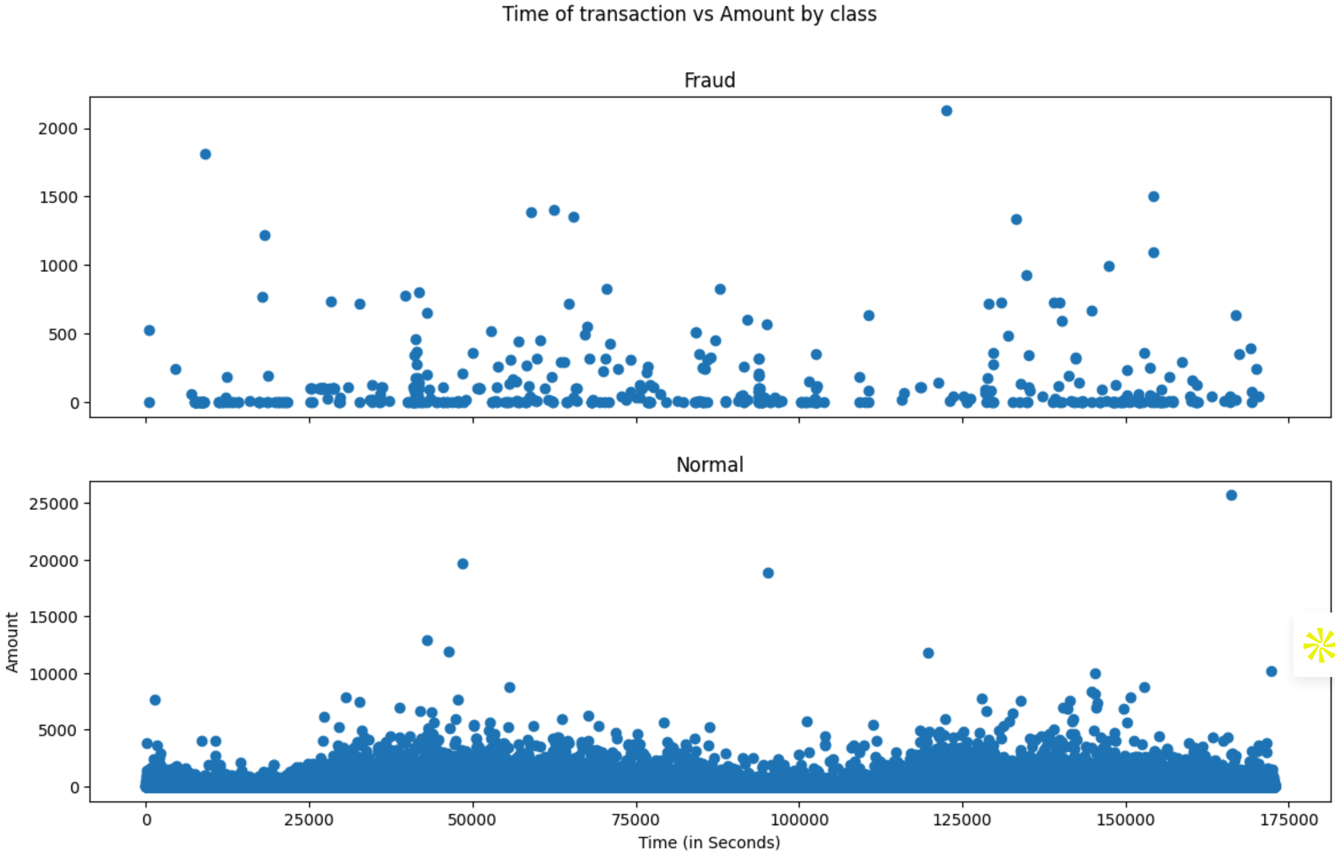
```
#Let's have a more graphical representation of the data

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Amount per transaction by class')
bins = 50
ax1.hist(Fraud.Amount, bins = bins)
ax1.set_title('Fraud')
ax2.hist(Normal.Amount, bins = bins)
ax2.set_title('Normal')
plt.xlabel('Amount ($)')
plt.ylabel('Number of Transactions')
plt.xlim((0, 20000))
plt.yscale('log')
plt.show();
```




```
#Graphical representation of the data

f, (ax1, ax2) = plt.subplots(2, 1, sharex=True)
f.suptitle('Time of transaction vs Amount by class')
ax1.scatter(Fraud.Time, Fraud.Amount)
ax1.set_title('Fraud')
ax2.scatter(Normal.Time, Normal.Amount)
ax2.set_title('Normal')
plt.xlabel('Time (in Seconds)')
plt.ylabel('Amount')
plt.show();
```




```
init_notebook_mode(connected=True)
plotly.offline.init_notebook_mode(connected=True)
```

```
# Create a trace

trace = go.Scatter(
    x = Fraud.Time,
    y = Fraud.Amount,
    mode = 'markers'
)
data = [trace]

plotly.offline.iplot({
    "data": data
})
```

```
data1.shape
```

```
(28481, 31)
```

```
#Determine the number of fraud and valid transactions in the dataset.

Fraud = data1[data1['Class']==1]
Valid = data1[data1['Class']==0]
outlier_fraction = len(Fraud)/float(len(Valid))


#Now let us print the outlier fraction and no of Fraud and Valid Transaction cases

print(outlier_fraction)
print("Fraud Cases : {}".format(len(Fraud)))
print("Valid Cases : {}".format(len(Valid)))
```

```
0.0016529506928325245
Fraud Cases : 47
Valid Cases : 28434
```

```
#Correlation Matrix

correlation_matrix = data1.corr()
fig = plt.figure(figsize=(12,9))
sns.heatmap(correlation_matrix,vmax=0.8,square = True)
plt.show()
```


```
#Get all the columns from the dataframe

columns = data1.columns.tolist()
# Filter the columns to remove data we do not want
columns = [c for c in columns if c not in ["Class"]]
# Store the variable we are predicting
target = "Class"
# Define a random state
state = np.random.RandomState(42)
X = data1[columns]
Y = data1[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))
# Print the shapes of X & Y
print(X.shape)
print(Y.shape)
```

```
(28481, 30)
(28481,)
```


```
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor
from sklearn.svm import OneClassSVM

# Define outlier fraction and random seed
outlier_fraction = len(data1[data1['Class'] == 1]) / len(data1)
state = 42

# Feature matrix (X) should exclude the target column 'Class'
X = data1.drop(['Class'], axis=1).values
y_true = data1['Class'].values  # Ground truth

# Define the outlier detection methods
classifiers = {
    "Isolation Forest": IsolationForest(
        n_estimators=100,
        max_samples=len(X),
        contamination=outlier_fraction,
        random_state=state,
        verbose=0
    ),

    "Local Outlier Factor": LocalOutlierFactor(
        n_neighbors=20,
        algorithm='auto',
```