+ Code + Text

```
import pandas as pd
data = pd.read_csv("/content/drive/MyDrive/financial_anomaly_data.csv")
```

```
data.shape
```

```
(217441, 7)
```

```
data.head()
```

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location
0	01-01-2023 08:00	TXN1127	ACC4	95071.92	MerchantH	Purchase	Tokyo
1	01-01-2023 08:01	TXN1639	ACC10	15607.89	MerchantH	Purchase	London
2	01-01-2023 08:02	TXN872	ACC8	65092.34	MerchantE	Withdrawal	London
3	01-01-2023 08:03	TXN1438	ACC6	87.87	MerchantE	Purchase	London
4	01-01-2023 08:04	TXN1338	ACC6	716.56	MerchantI	Purchase	Los Angeles

```
data.describe(include='all')
```

	Timestamp	TransactionID	AccountID	Amount	Merchant	TransactionType	Location
count	216960	216960	216960	216960.000000	216960	216960	216960
unique	216960	1999	15	NaN	10	3	5
top	01-01-2023 08:00	TXN838	ACC15	NaN	MerchantF	Transfer	San Francisco
freq	1	139	14701	NaN	21924	72793	43613
mean	NaN	NaN	NaN	50090.025108	NaN	NaN	NaN
std	NaN	NaN	NaN	29097.905016	NaN	NaN	NaN
min	NaN	NaN	NaN	10.510000	NaN	NaN	NaN
25%	NaN	NaN	NaN	25061.242500	NaN	NaN	NaN
50%	NaN	NaN	NaN	50183.980000	NaN	NaN	NaN
75%	NaN	NaN	NaN	75080.460000	NaN	NaN	NaN
max	NaN	NaN	NaN	978942.260000	NaN	NaN	NaN

```
data.columns
```

```
Index(['Timestamp', 'TransactionID', 'AccountID', 'Amount', 'Merchant',
       'TransactionType', 'Location'],
      dtype='object')
```

```
data.dtypes
```

```
Timestamp           object
TransactionID       object
AccountID           object
Amount             float64
Merchant            object
TransactionType     object
Location            object
dtype: object
```

```
# Check for missing values
data.isnull().sum()
```

```
Timestamp          481
TransactionID      481
AccountID          481
Amount             481
Merchant           481
TransactionType    481
Location           481
dtype: int64
```

```
data=data.dropna()
```

```
# Remove duplicates
data = data.drop_duplicates()
```

```
data.shape
```

```
(216960, 7)
```

```
# Convert Timestamp to datetime with specified format
data['Timestamp'] = pd.to_datetime(data['Timestamp'], format='%d-%m-%Y %H:%M', errors='coerce')

# Extract time-based features
data['Hour'] = data['Timestamp'].dt.hour
data['DayOfWeek'] = data['Timestamp'].dt.dayofweek
data['DayOfMonth'] = data['Timestamp'].dt.day
data['Month'] = data['Timestamp'].dt.month
```

```
data.head()
```

Show hidden output

```
from sklearn.preprocessing import StandardScaler

# Initialize the scaler
scaler = StandardScaler()

# Normalize the 'Amount' column
data['Amount'] = scaler.fit_transform(data[['Amount']])
```

```
data = pd.get_dummies(data, columns=['Merchant', 'TransactionType', 'Location'])

print(data.head())
```

```
             Timestamp TransactionID AccountID    Amount  Hour  DayOfWeek  \
0 2023-01-01 08:00:00       TXN1127      ACC4  1.545884     8          6
1 2023-01-01 08:01:00       TXN1639     ACC10 -1.185041     8          6
2 2023-01-01 08:02:00        TXN872      ACC8  0.515582     8          6
3 2023-01-01 08:03:00       TXN1438      ACC6 -1.718415     8          6
4 2023-01-01 08:04:00       TXN1338      ACC6 -1.696809     8          6

   DayOfMonth  Month  Merchant_MerchantA  Merchant_MerchantB  ...  \
0           1      1               False               False  ...
1           1      1               False               False  ...
2           1      1               False               False  ...
3           1      1               False               False  ...
4           1      1               False               False  ...

   Merchant_MerchantI  Merchant_MerchantJ  TransactionType_Purchase  \
0               False               False                      True
1               False               False                      True
2               False               False                     False
3               False               False                      True
4                True               False                      True

   TransactionType_Transfer  TransactionType_Withdrawal  Location_London  \
0                     False                       False            False
1                     False                       False             True
2                     False                        True             True
3                     False                       False             True
4                     False                       False            False

   Location_Los Angeles  Location_New York  Location_San Francisco  \
0                 False              False                   False
1                 False              False                   False
2                 False              False                   False
3                 False              False                   False
4                  True              False                   False

   Location_Tokyo
0            True
1           False
2           False
3           False
4           False

[5 rows x 26 columns]
```

```
# Calculate statistical summaries for each account
account_profiles = data.groupby('AccountID').agg({
    'Amount': ['mean', 'std', 'min', 'max', 'sum', 'count'],
    'Hour': ['mean', 'std'],
    'DayOfWeek': ['mean', 'std'],
    'Month': ['mean', 'std']
}).reset_index()

# Flatten multi-level columns
account_profiles.columns = ['_'.join(col).strip() for col in account_profiles.columns.values]
print(account_profiles.head())
```

```
  AccountID_  Amount_mean  Amount_std  Amount_min  Amount_max  Amount_sum  \
0       ACC1     0.011633    0.995594   -1.721073    4.716995  167.112682
1      ACC10    -0.012375    0.989686   -1.720899    1.714771 -177.735518
2      ACC11    -0.014704    0.997443   -1.721062    3.468661 -212.410998
3      ACC12     0.006269    1.007645   -1.721067   22.750383   90.411585
4      ACC13     0.015423    0.990247   -1.720974    1.715109  222.418199

   Amount_count  Hour_mean  Hour_std  DayOfWeek_mean  DayOfWeek_std  \
0         14365  11.610233  6.927493        2.980160       2.004370
1         14362  11.488859  6.919930        2.955856       1.998607
2         14446  11.542503  6.948485        2.970096       2.014710
3         14421  11.532834  6.912840        2.986131       2.014497
4         14421  11.507524  6.928866        2.980237       2.013708

   Month_mean  Month_std
0    3.020049   1.421633
1    3.008007   1.421141
2    3.022082   1.429452
3    3.018584   1.428485
4    3.019347   1.411897
```

```
from sklearn.cluster import KMeans, DBSCAN
from sklearn.ensemble import IsolationForest
from sklearn.neighbors import LocalOutlierFactor

# Prepare data for modeling (exclude non-numeric columns)
model_data = data.select_dtypes(include=[float, int])

# K-means Clustering
kmeans = KMeans(n_clusters=5, random_state=42)
kmeans_labels = kmeans.fit_predict(model_data)

# DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(model_data)

# Isolation Forest
iso_forest = IsolationForest(contamination=0.01, random_state=42)
iso_labels = iso_forest.fit_predict(model_data)

# Local Outlier Factor
lof = LocalOutlierFactor(n_neighbors=20, contamination=0.01)
lof_labels = lof.fit_predict(model_data)
```

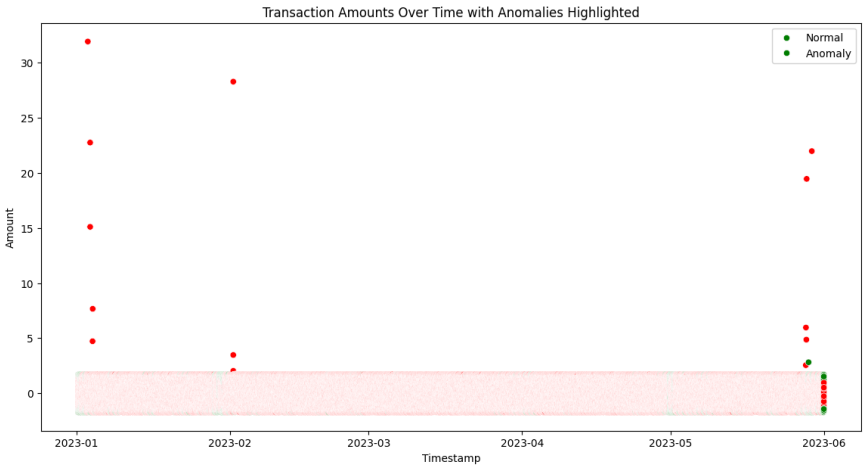
```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:1416: FutureWarning: The default value of `n_init` will change fr
  super()._check_params_vs_input(X, default_n_init=10)
```

```
# Add anomaly labels to the data
data['KMeans_Labels'] = kmeans_labels
data['DBSCAN_Labels'] = dbscan_labels
data['IsolationForest_Labels'] = iso_labels
data['LOF_Labels'] = lof_labels

# Summarize the results
anomaly_counts = {
    'KMeans': (kmeans_labels == -1).sum(),
    'DBSCAN': (dbscan_labels == -1).sum(),
    'IsolationForest': (iso_labels == -1).sum(),
    'LOF': (lof_labels == -1).sum()
}
print(anomaly_counts)
```

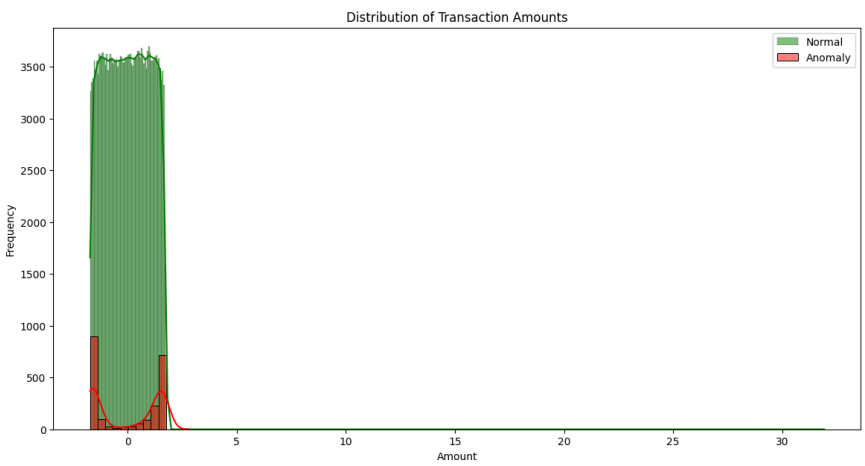
```
{'KMeans': 0, 'DBSCAN': 20, 'IsolationForest': 2170, 'LOF': 2170}
```

```
# Export the data with anomaly labels to CSV for Tableau
data.to_csv('anomaly_detection_results.csv', index=False)
```

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

# Scatter plot of transaction amounts over time with anomalies highlighted
plt.figure(figsize=(14, 7))
sns.scatterplot(x=data['Timestamp'], y=data['Amount'], hue=data['IsolationForest_Labels'], palette=['green', 'red'])
plt.title('Transaction Amounts Over Time with Anomalies Highlighted')
plt.xlabel('Timestamp')
plt.ylabel('Amount')
plt.legend(['Normal', 'Anomaly'])
plt.show()
```




```
plt.figure(figsize=(14, 7))
sns.histplot(data[data['IsolationForest_Labels'] == 1]['Amount'], color='green', kde=True, label='Normal')
sns.histplot(data[data['IsolationForest_Labels'] == -1]['Amount'], color='red', kde=True, label='Anomaly')
plt.title('Distribution of Transaction Amounts')
plt.xlabel('Amount')
plt.ylabel('Frequency')
plt.legend()
plt.show()
```




```
plt.figure(figsize=(14, 7))
sns.boxplot(x='IsolationForest_Labels', y='Amount', data=data, palette=['green', 'red'])
plt.title('Box Plot of Transaction Amounts by Anomaly Label')
plt.xlabel('Anomaly Label')
plt.ylabel('Amount')
plt.show()
```

```
<ipython-input-56-c0726b5252f6>:2: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.

  sns.boxplot(x='IsolationForest_Labels', y='Amount', data=data, palette=['green', 'r
```

Start coding or generate with AI.