REAL-TIME DATA STREAM CLUSTERING USING MUDI-DENSITY BASED CLUSTERING

Project by:

[Ujjwal Chaudhary]

[Shashank Acharya]

Date: [13-02-2025]

PROJECT OBJECTIVE

- Analyze real-time credit card transaction dataset using clustering techniques.
- Simulate real-time data processing by chunking the dataset.
- Use DBSCAN for clustering to detect patterns and anomalies.
- Identify outliers and customer behavior trends for fraud detection and risk assessment.

DATASET OVERVIEW

- Credit card dataset with features like:

  - BILL_AMT1 to BILL_AMT6 (Monthly bill amounts)

  - LIMIT_BAL (Credit limit)

  - default payment next month (Target variable)

[109]:
```python
#reading the data inside the dataset
import pandas as pd

# Load the dataset
data = pd.read_csv('Downloads/credit card dataset.csv')

# Print column names
print(data.columns)
```

```
Index(['ID', 'LIMIT_BAL', 'SEX', 'EDUCATION', 'MARRIAGE', 'AGE', 'PAY_0',
       'PAY_2', 'PAY_3', 'PAY_4', 'PAY_5', 'PAY_6', 'BILL_AMT1', 'BILL_AMT2',
       'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6', 'PAY_AMT1',
       'PAY_AMT2', 'PAY_AMT3', 'PAY_AMT4', 'PAY_AMT5', 'PAY_AMT6',
       'default payment next month'],
      dtype='object')
```

[110]:
```python
data.head(10)
```

[110]:

	ID	LIMIT_BAL	SEX	EDUCATION	MARRIAGE	AGE	PAY_0	PAY_2	PAY_3	PAY_4	...	BILL_AMT4	BILL_AMT5	BILL_AMT6	PAY_AMT1	PAY_AMT2	PAY_AMT3	PAY
0	1	20000	2	2	1	24	2	2	-1	-1	...	0	0	0	0	689	0	
1	2	120000	2	2	2	26	-1	2	0	0	...	3272	3455	3261	0	1000	1000	
2	3	90000	2	2	2	34	0	0	0	0	...	14331	14948	15549	1518	1500	1000	
3	4	50000	2	2	1	37	0	0	0	0	...	28314	28959	29547	2000	2019	1200	
4	5	50000	1	2	1	57	-1	0	-1	0	...	20940	19146	19131	2000	36681	10000	
5	6	50000	1	1	2	37	0	0	0	0	...	19394	19619	20024	2500	1815	657	
6	7	500000	1	1	2	29	0	0	0	0	...	542653	483003	473944	55000	40000	38000	
7	8	100000	2	2	2	23	0	-1	-1	0	...	221	-159	567	380	601	0	
8	9	140000	2	3	1	28	0	0	2	0	...	12211	11793	3719	3329	0	432	
9	10	20000	1	3	2	35	-2	-2	-2	-2	...	0	13007	13912	0	0	0	

10 rows × 25 columns

FEATURE ENGINEERING

- CUR (Credit Utilization Ratio): Average bill amount / Credit limit.
- PDT (Payment Delay Trend): Weighted sum of payment delays.
- TOA (Total Outstanding Amount): Sum of all bill amounts.
- RPB (Recent Payment Behavior): Average of recent payment statuses.

```
# Feature engineering
def feature_engineering(df):
    # Create new features from raw data
    df['CUR'] = df[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']].mean(axis=1) / df['LIMIT_BAL']
    df['PDT'] = (df['PAY_0'] * 3 + df['PAY_2'] * 2.5 + df['PAY_3'] * 2 + df['PAY_4'] * 1.5 + df['PAY_5'] * 1 + df['PAY_6'] * 0.5)
    df['TOA'] = df[['BILL_AMT1', 'BILL_AMT2', 'BILL_AMT3', 'BILL_AMT4', 'BILL_AMT5', 'BILL_AMT6']].sum(axis=1)
    df['RPB'] = (df['PAY_0'] + df['PAY_2']) / 2
    df['Defaults'] = df['default payment next month']
    return df

# Apply feature engineering
data = feature_engineering(data)
```

	CUR	PDT	TOA	RPB	Defaults
count	30000.000000	30000.000000	3.000000e+04	30000.000000	30000.000000
mean	0.373048	-1.459667	2.698617e+05	-0.075233	0.221200
std	0.351890	10.372223	3.795643e+05	1.061231	0.415062
min	-0.232590	-21.000000	-3.362590e+05	-2.000000	0.000000
25%	0.029997	-8.500000	2.868800e+04	-0.500000	0.000000
50%	0.284834	0.000000	1.263110e+05	0.000000	0.000000
75%	0.687929	0.000000	3.426265e+05	0.000000	0.000000
max	5.364308	66.500000	5.263883e+06	7.500000	1.000000

FORMULAE

CUR → Current Utilization Rate
PDT → Payment Delay Trend
TOA → Total Outstanding Amount
RPB → Recent Payment Behaviour

$$CUR = \frac{\text{Mean}(\text{BILL\_AMT1}, \text{BILL\_AMT2}, \text{BILL\_AMT3}, \text{BILL\_AMT4}, \text{BILL\_AMT5}, \text{BILL\_AMT6})}{\text{LIMIT\_BAL}}$$

$$PDT = (PAY_0 \times 3) + (PAY_2 \times 2.5) + (PAY_3 \times 2) + (PAY_4 \times 1.5) + (PAY_5 \times 1) + (PAY_6 \times 0.5)$$

$$TOA = BILL\_AMT1 + BILL\_AMT2 + BILL\_AMT3 + BILL\_AMT4 + BILL\_AMT5 + BILL\_AMT6$$

$$RPB = \frac{PAY_0 + PAY_2}{2}$$

PREPROCESSING AND STANDARDIZATION

- Standardized data using StandardScaler.
- Checked and replaced NaN and infinite values with zeros.
- Ensured features were numerical and cleaned missing values.

[114]:

```python
from sklearn.preprocessing import StandardScaler
import numpy as np
import pandas as pd

# Standardize the data
scaler = StandardScaler()
data = scaler.fit_transform(df)  # df is your original DataFrame

# Convert back to DataFrame with original column names
data_scaled = pd.DataFrame(data, columns=df.columns)

# Print the first 10 rows of the scaled data
print(data_scaled.head(10))
```

```
        CUR       PDT       TOA       RPB  Defaults
0 -0.877698  0.574589 -0.690692  1.955529  1.876378
1 -0.992741  0.429970 -0.665997  0.542053  1.876378
2 -0.525178  0.140731 -0.443170  0.070894 -0.532942
3  1.131240  0.140731 -0.101507  0.070894 -0.532942
4 -0.024396 -0.341334 -0.422920 -0.400265 -0.532942
5  1.195466  0.140731 -0.083644  0.070894 -0.532942
6  1.520814  0.140731  6.467347  0.070894 -0.532942
7 -0.996269 -0.341334 -0.675458 -0.400265 -0.532942
8 -0.839523  0.526383 -0.539179  0.070894 -0.532942
9 -0.422648 -1.739322 -0.640067 -1.813742 -0.532942
```

REAL-TIME DATA SIMULATION

- Simulated real-time data by splitting the dataset into 100 chunks.
- Each chunk contains approximately equal rows for analysis.
- Processed each chunk sequentially to mimic real-time streaming.

```
[118]: # Split data into 100 chunks
       chunks = np.array_split(data_scaled, 100)  # Each chunk contains 300 rows (approx.)
```

CLUSTERING METHODOLOGY

- Applied DBSCAN (Density-Based Spatial Clustering of Applications with Noise).
- Parameters:

  - eps: Maximum distance between two points to be in the same cluster.

  - min_samples: Minimum number of points to form a dense region.
- Labeled noise points (outliers) as -1.

```
from sklearn.cluster import DBSCAN

# Use DBSCAN to merge micro-clusters
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(micro_clusters[:, :-1])  # Ignore weights during clustering
```

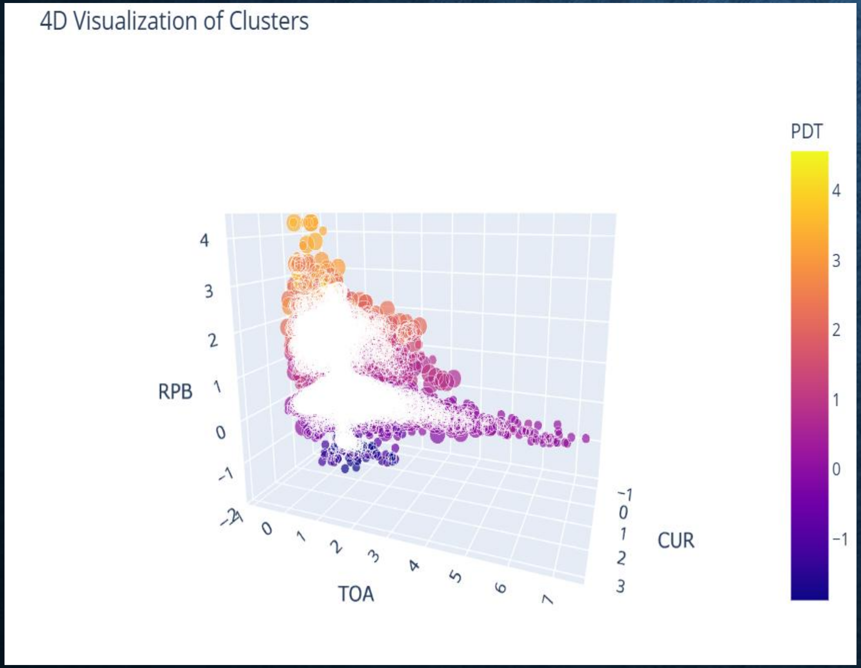
```
print(f"Shape of micro_clusters: {np.array(micro_clusters).shape}")

merged_clusters = pd.DataFrame(micro_clusters, columns=['CUR', 'PDT', 'TOA', 'RPB', 'Default', 'cluster_label'])
```

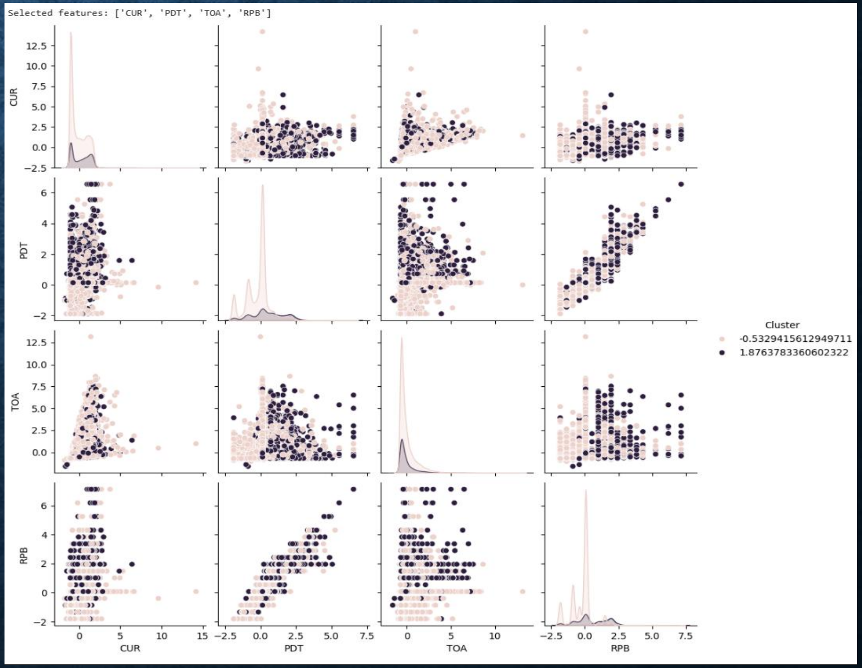
```
Shape of micro_clusters: (10000, 6)
```

```
dbscan = DBSCAN(eps=0.5, min_samples=5)
final_labels = dbscan.fit_predict(micro_clusters[:, :-1])  # Ignore weight column
merged_clusters['final_cluster_label'] = final_labels
```

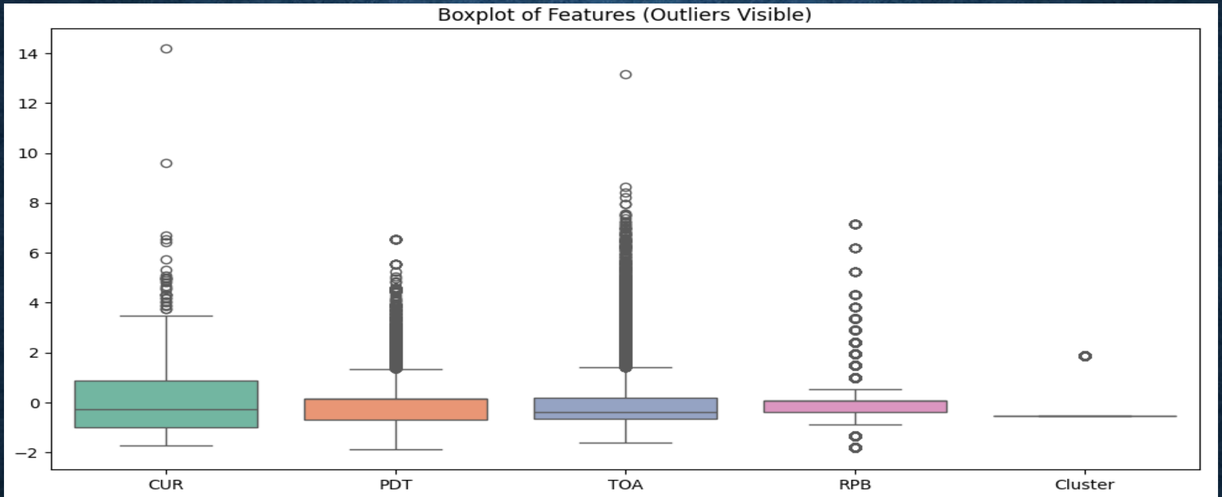
RESULTS & VISUALIZATIONS

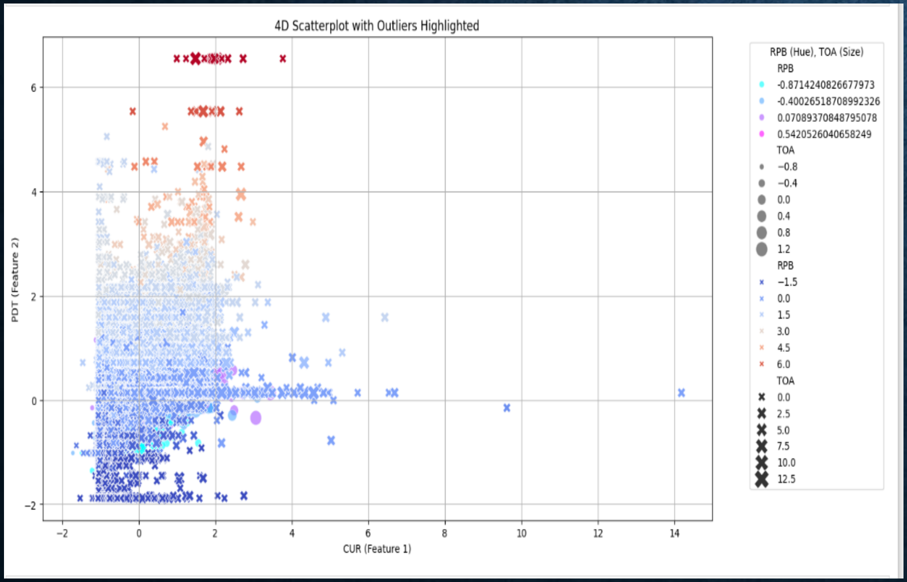
- Identified clusters representing customer behavior and anomaly detected.
- Outliers detected as potential anomalies or fraud cases.
- Visualized clusters using scatter plots and 4D and dataframe.

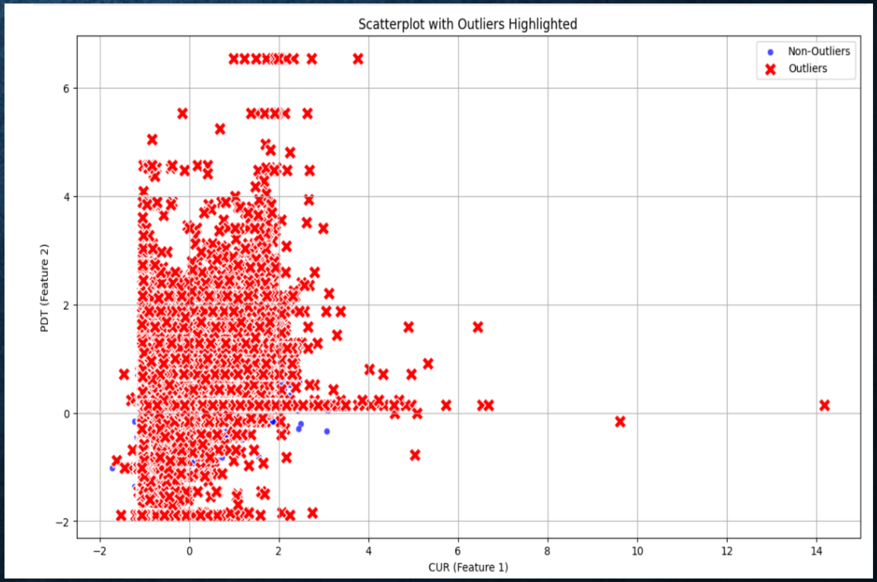




OUTLINERS & VISUALIZATION







KEY INSIGHTS

- CUR and PDT features were highly influential in clustering.
- Detected distinct customer behavior patterns.
- Noise points indicate unusual patterns, useful for anomaly detection.

CONCLUSION & FUTURE WORK

- Real-time clustering provides valuable insights for credit risk and fraud detection.
- Future improvements:

  - Use advanced clustering algorithms (e.g., MuDi).

  - Integrate real-time streaming systems (Kafka, Spark).