**1. Feature Engineering**

We’ll extract features that represent liquidity, volatility, and volume behavior. Here’s a breakdown:

**Order Book-Based Features**

These come from depth20, i.e., the top 20 levels of the order book.

* **Bid-Ask Spread**

python

CopyEdit

spread = ask\_prices[0] - bid\_prices[0]

* **Microprice**  
  Gives a weighted average price near the top:

python

CopyEdit

microprice = (bid\_prices[0] \* ask\_qty[0] + ask\_prices[0] \* bid\_qty[0]) / (bid\_qty[0] + ask\_qty[0])

* **Order Book Imbalance (Level 1)**  
  Measures aggressiveness on one side of the market:

python

CopyEdit

imbalance\_lvl1 = (bid\_qty[0] - ask\_qty[0]) / (bid\_qty[0] + ask\_qty[0])

* **Cumulative Depth (Top 20)**  
  Shows depth of liquidity on both sides:

python

CopyEdit

cum\_bid\_qty = np.sum(bid\_qty)

cum\_ask\_qty = np.sum(ask\_qty)

* **Sloped Depth**  
  Shows how liquidity decays:

python

CopyEdit

bid\_slope = (bid\_qty[0] - bid\_qty[-1]) / 20

ask\_slope = (ask\_qty[0] - ask\_qty[-1]) / 20

* **Trade Wipe Level**  
  Average number of levels cleared in a window by aggressive trades (requires tracking executed trades vs resting orders).

**Price & Volatility Features**

Use mid-price = (best bid + best ask) / 2.

* **Rolling Return (log)**

python

CopyEdit

log\_return = np.log(mid\_price / mid\_price.shift(1))

* **Rolling Volatility**

python

CopyEdit

rolling\_vol\_10s = log\_return.rolling(window=10).std()

rolling\_vol\_30s = log\_return.rolling(window=30).std()

**Volume Features**

Use aggTrade data.

* **Volume Imbalance**

python

CopyEdit

vol\_imbalance = (buy\_volume - sell\_volume) / (buy\_volume + sell\_volume)

* **Cumulative Volume (10s, 30s)**

python

CopyEdit

cum\_vol\_10s = volume.rolling(window=10).sum()

cum\_vol\_30s = volume.rolling(window=30).sum()

* **VWAP & VWAP Shift**

python

CopyEdit

vwap = (price \* volume).sum() / volume.sum()

vwap\_shift = vwap.diff()

**2. Data Normalization**

Normalize all features to standardize scale.

* **Z-score normalization**

python

CopyEdit

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

Optional: Apply **PCA** or **UMAP** for dimension reduction before clustering.

**3. Clustering**

Try multiple clustering algorithms and compare.

* **K-Means**

python

CopyEdit

from sklearn.cluster import KMeans

inertia = []

for k in range(2, 10):

kmeans = KMeans(n\_clusters=k)

kmeans.fit(X\_scaled)

inertia.append(kmeans.inertia\_)

* **HDBSCAN**

python

CopyEdit

import hdbscan

clusterer = hdbscan.HDBSCAN(min\_cluster\_size=30)

labels = clusterer.fit\_predict(X\_scaled)

* **Gaussian Mixture Models**

python

CopyEdit

from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n\_components=4)

labels = gmm.fit\_predict(X\_scaled)

Evaluate using:

python

CopyEdit

from sklearn.metrics import silhouette\_score, davies\_bouldin\_score

silhouette = silhouette\_score(X\_scaled, labels)

db\_score = davies\_bouldin\_score(X\_scaled, labels)

**4. Regime Labeling and Analysis**

After clustering:

* Assign the regime label to each timestamp.
* Analyze per cluster:
  + Average bid-ask spread
  + Average volatility
  + Average trade volume
  + Directionality (check if prices trend up/down over time)

Map regime types manually:

* Regime A: Trending + Liquid + Stable
* Regime B: Mean-Reverting + Illiquid + Volatile

You can use statistical summaries:

python

CopyEdit

df.groupby("regime").agg({

"spread": "mean",

"rolling\_vol\_30s": "mean",

"cum\_vol\_30s": "mean",

...

})

**5. Visualization**

* **Regime Over Time** Plot cluster labels over time:

python

CopyEdit

plt.plot(time\_index, regime\_labels)

* **Overlay with Price**

python

CopyEdit

plt.plot(price)

plt.plot(regime\_labels \* scale\_factor) # For overlaying

* **t-SNE or UMAP 2D Cluster Plot**

python

CopyEdit

from sklearn.manifold import TSNE

tsne = TSNE(n\_components=2)

X\_2d = tsne.fit\_transform(X\_scaled)

plt.scatter(X\_2d[:, 0], X\_2d[:, 1], c=regime\_labels)

**6. Regime Change Insights**

Study regime transition matrix:

python

CopyEdit

from sklearn.metrics import confusion\_matrix

transition\_matrix = pd.crosstab(df.regime.shift(), df.regime, normalize='index')

This gives probability that regime A is followed by regime B.