NLP-Based Customer Feedback Intent Clustering

Problem Statement

In customer feedback analysis, organizations often receive large volumes of unstructured text data. This makes it difficult to understand customer intent, pain points, and sentiment trends manually.

The goal of this project is to:

1. Automatically analyze customer feedback text.
2. Identify underlying intents or themes (e.g., product complaints, service praise, delivery issues).
3. Perform sentiment analysis to understand emotional tone.
4. Use multiple clustering algorithms to find optimal intent groups.
5. Compare clustering models and visualize results to derive meaningful insights.

Block 1 – Setup and Imports

This block installs all the libraries you'll use (data handling, NLP, clustering, and visualization).

```
!pip install pandas numpy matplotlib seaborn nltk sentence-transformers scikit-learn textblob p

import pandas as pd
import numpy as np
import re
import nltk
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sentence_transformers import SentenceTransformer
from sklearn.cluster import KMeans, DBSCAN
from sklearn.manifold import TSNE
from textblob import TextBlob
import hdbscan

nltk.download('stopwords')
nltk.download('wordnet')

print("✅ Environment ready")
```

```
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.12/dist-packages (from matpl
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (frc
Requirement already satisfied: click in /usr/local/lib/python3.12/dist-packages (from nltk) (8.
Requirement already satisfied: joblib in /usr/local/lib/python3.12/dist-packages (from nltk) (1
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from nltk) (4.6
Requirement already satisfied: transformers<5.0.0,>=4.41.0 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: torch>=1.11.0 in /usr/local/lib/python3.12/dist-packages (from s
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from sentence-
Requirement already satisfied: huggingface-hub>=0.20.0 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: typing_extensions>=4.5.0 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from huggir
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.12/dist-packages (frc
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.12/dist-packages (from hug
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from huggir
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from pythor
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torc
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch>
Requirement already satisfied: jinja2 in /usr/local/lib/python3.12/dist-packages (from torch>=1
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dis
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/c
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dis
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-pack
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-p
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from t
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-pa
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from re
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (f
✅ Environment ready
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Block 2 – Load Dataset & Split Columns

Your dataset has one combined column. We'll split it into multiple columns (like Text, Sentiment, Source, etc.) using string operations.

```
from google.colab import files
uploaded = files.upload()

file_name = list(uploaded.keys())[0]
```

```
df = pd.read_csv(file_name)
print("✅ Dataset loaded. Shape:", df.shape)

# Assume only 1 column
col = df.columns[0]
print(f"Detected single column: {col}")

# Split by comma (or adjust separator if needed)
split_cols = ['Text', 'Sentiment', 'Source', 'DateTime', 'UserID', 'Location', 'ConfidenceScore']

def split_row(row):
    if pd.isna(row):  # Skip NaN values
        return pd.Series([None]*len(split_cols), index=split_cols)
    parts = [p.strip() for p in str(row).split(',')] # Convert to string before splitting
    if len(parts) < len(split_cols):
        parts += [None]*(len(split_cols)-len(parts))
    return pd.Series(parts[:len(split_cols)], index=split_cols)

df = df[col].apply(split_row)
print("✅ Column split into:", df.columns.tolist())
display(df.head())
```

Choose files sentiment-analysis.csv

sentiment-analysis.csv(text/csv) - 13281 bytes, last modified: 15/10/2025 - 100% done

```
Saving sentiment-analysis.csv to sentiment-analysis (1).csv
✅ Dataset loaded. Shape: (98, 1)
Detected single column: Text, Sentiment, Source, Date/Time, User ID, Location, Confidence Score
✅ Column split into: ['Text', 'Sentiment', 'Source', 'DateTime', 'UserID', 'Location', 'Confide
```

	Text	Sentiment	Source	DateTime	UserID	Location	ConfidenceScore
0	"I love this product!"	Positive	Twitter	2023-06-15 09:23:14	@user123	New York	0.85
1	"The service was terrible."	Negative	Yelp Reviews	2023-06-15 11:45:32	user456	Los Angeles	0.65
2	"This movie is amazing!"	Positive	IMDb	2023-06-15 14:10:22	moviefan789	London	0.92
3	"I'm so disappointed with their customer suppo...	Negative	Online Forum	2023-06-15 17:35:11	forumuser1	Toronto	0.78

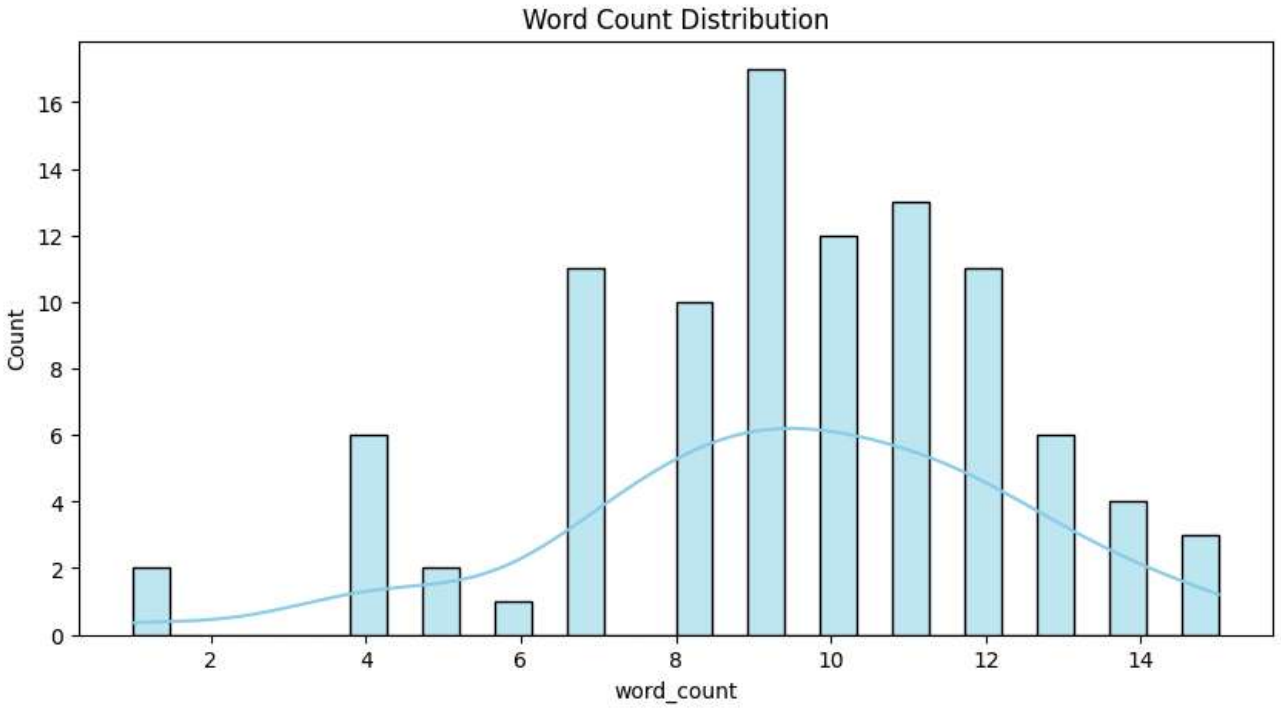
Block 3 – Exploratory Data Analysis (EDA)

Understand the dataset structure before modeling. We check missing values, text length, and sentiment distribution.

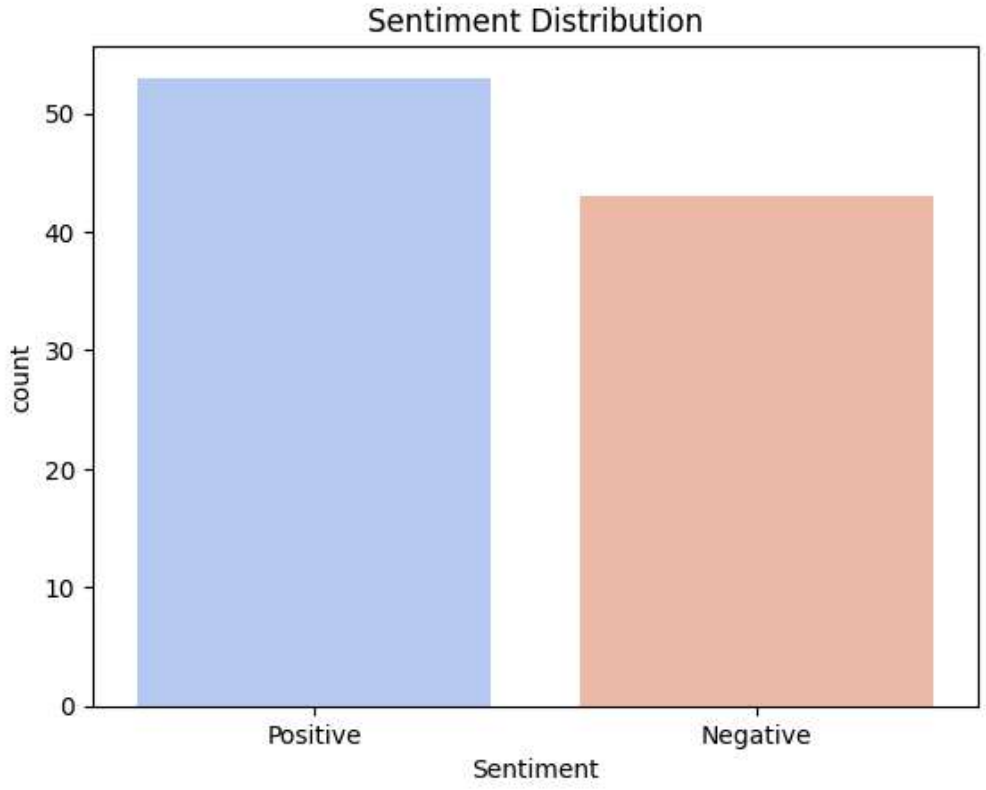
```
df['text_length'] = df['Text'].astype(str).apply(len)
df['word_count'] = df['Text'].astype(str).apply(lambda x: len(x.split()))

print("Average word count:", df['word_count'].mean())
```

```
plt.figure(figsize=(10,5))
sns.histplot(df['word_count'], bins=30, kde=True, color='skyblue')
plt.title("Word Count Distribution")
plt.show()

# Sentiment distribution
sns.countplot(data=df, x='Sentiment', palette='coolwarm')
plt.title("Sentiment Distribution")
plt.show()
```

```
Average word count: 9.448979591836734
```




```
/tmp/ipython-input-3592463459.py:12: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign

  sns.countplot(data=df, x='Sentiment', palette='coolwarm')
```



Block 4 – Text Preprocessing

We clean text (lowercase, remove punctuation, stopwords, and lemmatize). This ensures uniform input for embeddings.

```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess(text):
    text = str(text).lower()
    text = re.sub(r'[^a-z\s]', '', text)
    tokens = text.split()
    tokens = [lemmatizer.lemmatize(w) for w in tokens if w not in stop_words]
    return " ".join(tokens)

df['clean_text'] = df['Text'].apply(preprocess)
df[['Text','clean_text']].head()
```

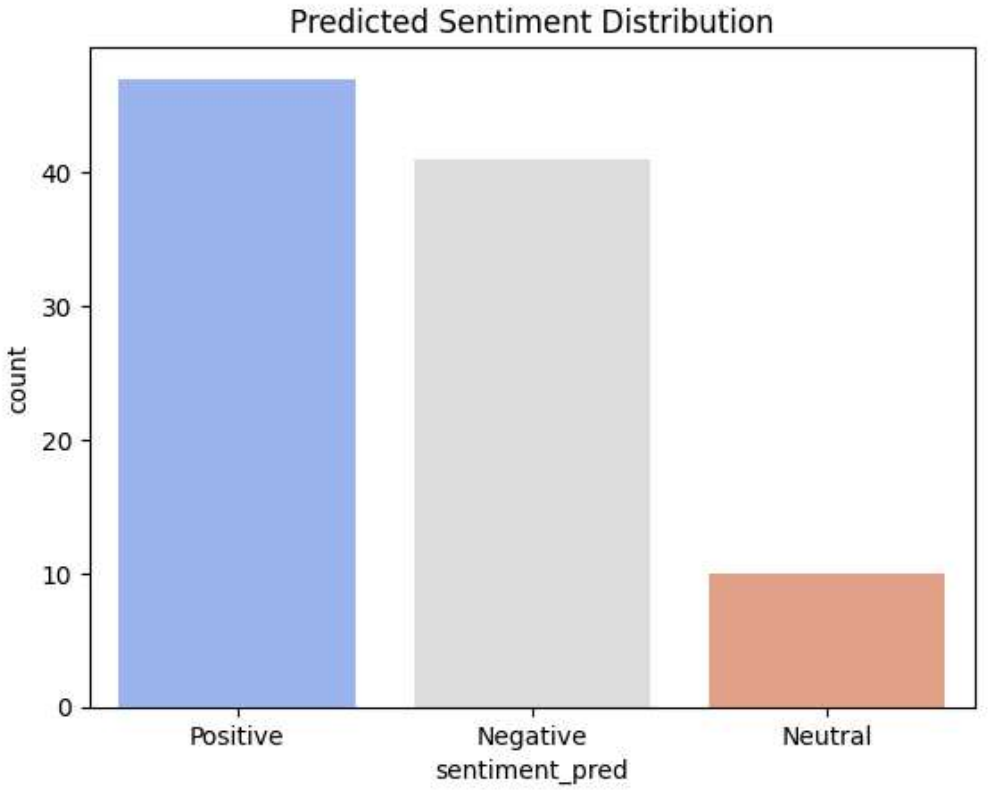
	Text	clean_text
0	"I love this product!"	love product
1	"The service was terrible."	service terrible
2	"This movie is amazing!"	movie amazing
3	"I'm so disappointed with their customer suppo...	im disappointed customer support
4	"Just had the best meal of my life!"	best meal life

Block 5 – Sentiment Analysis

Add sentiment scores to understand emotional polarity in the feedback.

```
def get_sentiment(text):
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity
    if polarity > 0: return "Positive", polarity
    elif polarity < 0: return "Negative", polarity
    else: return "Neutral", polarity

df[['sentiment_pred', 'sentiment_score']] = df['clean_text'].apply(lambda x: pd.Series(get_sent

sns.countplot(data=df, x='sentiment_pred', palette='coolwarm')
plt.title("Predicted Sentiment Distribution")
plt.show()
```

```
/tmp/ipython-input-125302670.py:10: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign t

  sns.countplot(data=df, x='sentiment_pred', palette='coolwarm')
```



Block 6 – Embedding Generation

Convert text into vector representations using Sentence-BERT, which captures semantic meaning.

```
model = SentenceTransformer('all-MiniLM-L6-v2')
embeddings = model.encode(df['clean_text'], show_progress_bar=True)
print("✅ Embeddings shape:", embeddings.shape)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingf
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or dat
  warnings.warn(
Batches: 100%                                                 4/4 [00:00<00:00,  4.21it/s]
✅ Embeddings shape: (98, 384)
```

Blocks 7–12 – Clustering Models

We apply multiple clustering algorithms to identify intent groups.

Each block uses the same embeddings, but different clustering logic.

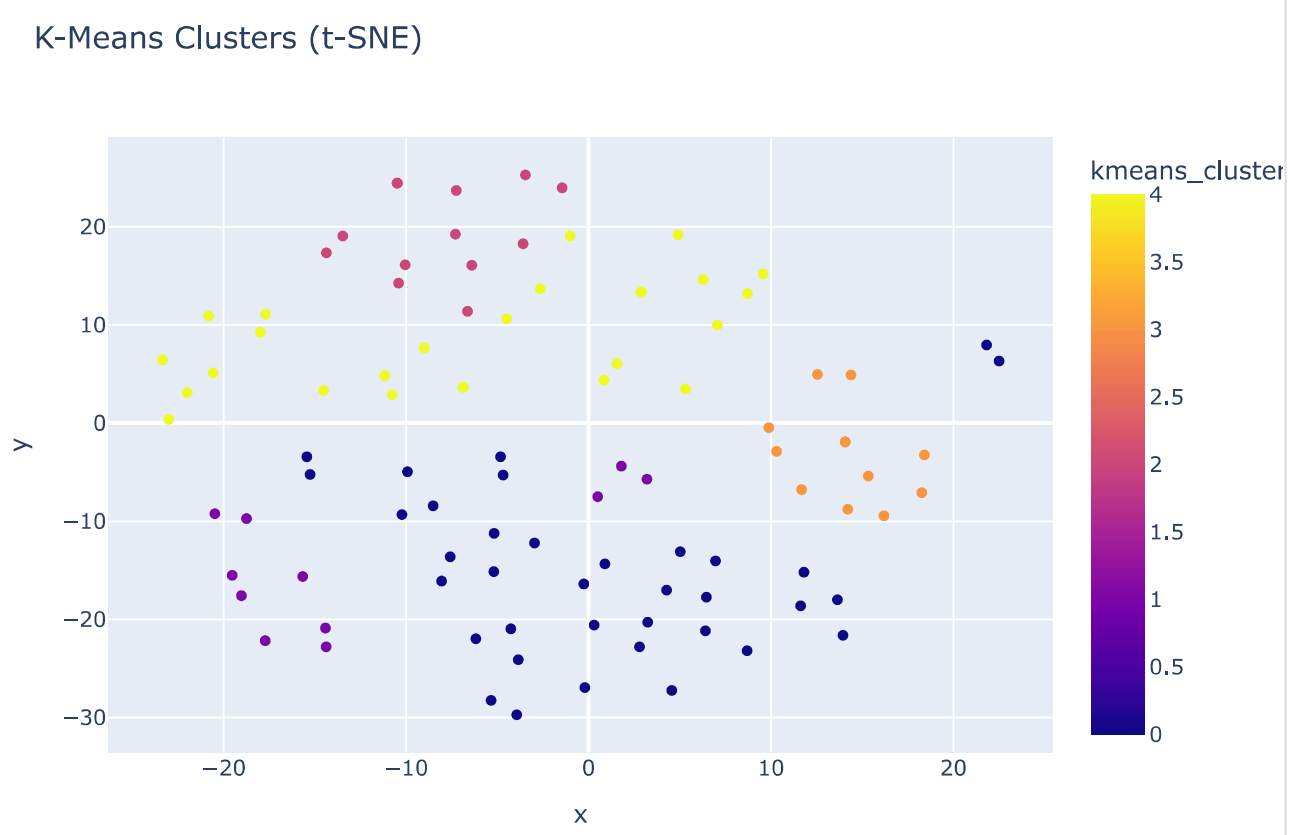
```
# --- Optimized K-Means Clustering ---

from sklearn.cluster import MiniBatchKMeans
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# 1️⃣ Optional: Reduce dimensions with PCA for faster clustering
pca = PCA(n_components=50, random_state=42)
embeddings_reduced = pca.fit_transform(embeddings)
print("Embeddings reduced from", embeddings.shape[1], "to", embeddings_reduced.shape[1], "dimen

# 2️⃣ Optional Elbow Method (only on reduced embeddings)
use_elbow = False  # Set True if you want to check optimal k
if use_elbow:
    inertia = []
    K = range(2, 10)
    for k in K:
        km = MiniBatchKMeans(n_clusters=k, batch_size=100, random_state=42)
        km.fit(embeddings_reduced)
        inertia.append(km.inertia_)
    plt.plot(K, inertia, 'o-')
    plt.title("Elbow Method (MiniBatchKMeans)")
    plt.xlabel("Number of clusters (k)")
    plt.ylabel("Inertia")
    plt.show()
    optimal_k = int(input("Enter optimal number of clusters from elbow: "))
else:
    optimal_k = 5  # Set manually if you know expected cluster count

# 3️⃣ Fit MiniBatchKMeans
kmeans = MiniBatchKMeans(n_clusters=optimal_k, batch_size=100, random_state=42)
df['kmeans_cluster'] = kmeans.fit_predict(embeddings_reduced)

print("✅ K-Means clustering complete")
df[['clean_text', 'kmeans_cluster']].head()
```

```
Embeddings reduced from 384 to 50 dimensions
✅ K-Means clustering complete
```

	clean_text	kmeans_cluster
0	love product	0
1	service terrible	0
2	movie amazing	4
3	im disappointed customer support	0
4	best meal life	3

```
# t-SNE Visualization for K-Means

from sklearn.manifold import TSNE
import plotly.express as px

tsne = TSNE(n_components=2, random_state=42, perplexity=30)
```

```
tsne_results = tsne.fit_transform(embeddings)
df['x'], df['y'] = tsne_results[:,0], tsne_results[:,1]

fig = px.scatter(
    df, x='x', y='y',
    color='kmeans_cluster',
    hover_data=['Text','Sentiment','sentiment_pred'],
    title="K-Means Clusters (t-SNE)"
)
fig.show()

# print("✅ Skipping t-SNE visualization for K-Means")
```



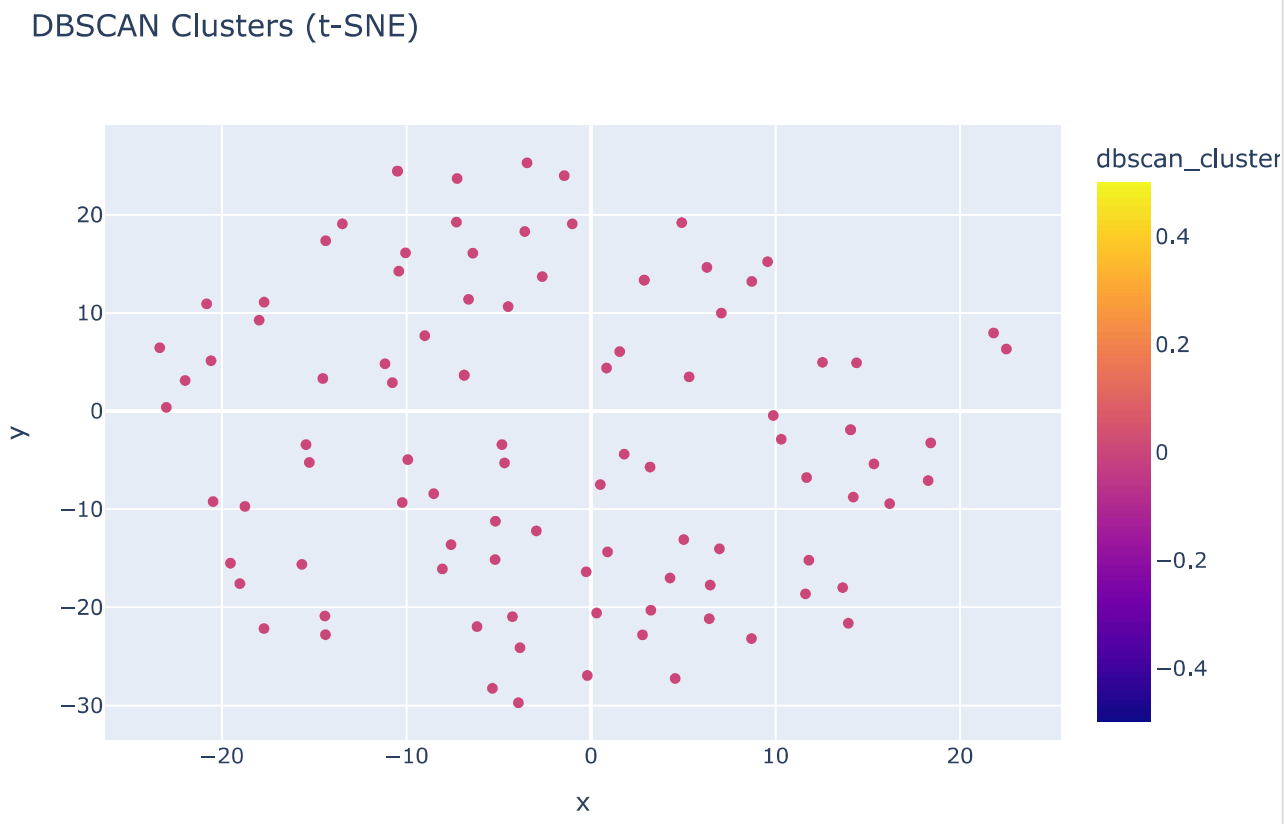

```
# DBSCAN Clustering

from sklearn.cluster import DBSCAN

# DBSCAN parameters
db = DBSCAN(eps=1.5, min_samples=5).fit(embeddings)
df['dbscan_cluster'] = db.labels_

# Check number of clusters and noise points
num_clusters = len(set(db.labels_)) - (1 if -1 in db.labels_ else 0)
num_noise = list(db.labels_).count(-1)

print(f"✅ DBSCAN clustering complete")
print(f"Number of clusters found: {num_clusters}")
print(f"Number of noise points: {num_noise}")
```

```
df[['clean_text', 'dbscan_cluster']].head()
```

```
✅ DBSCAN clustering complete
Number of clusters found: 1
Number of noise points: 0
```

	clean_text	dbscan_cluster
0	love product	0
1	service terrible	0
2	movie amazing	0
3	im disappointed customer support	0
4	best meal life	0

```
# t-SNE Visualization for DBSCAN

fig = px.scatter(
    df, x='x', y='y',
    color='dbscan_cluster',
    hover_data=['Text','Sentiment','sentiment_pred'],
    title="DBSCAN Clusters (t-SNE)"
)
fig.show()
# print("✅ Skipping t-SNE visualization for DBSCAN")
```

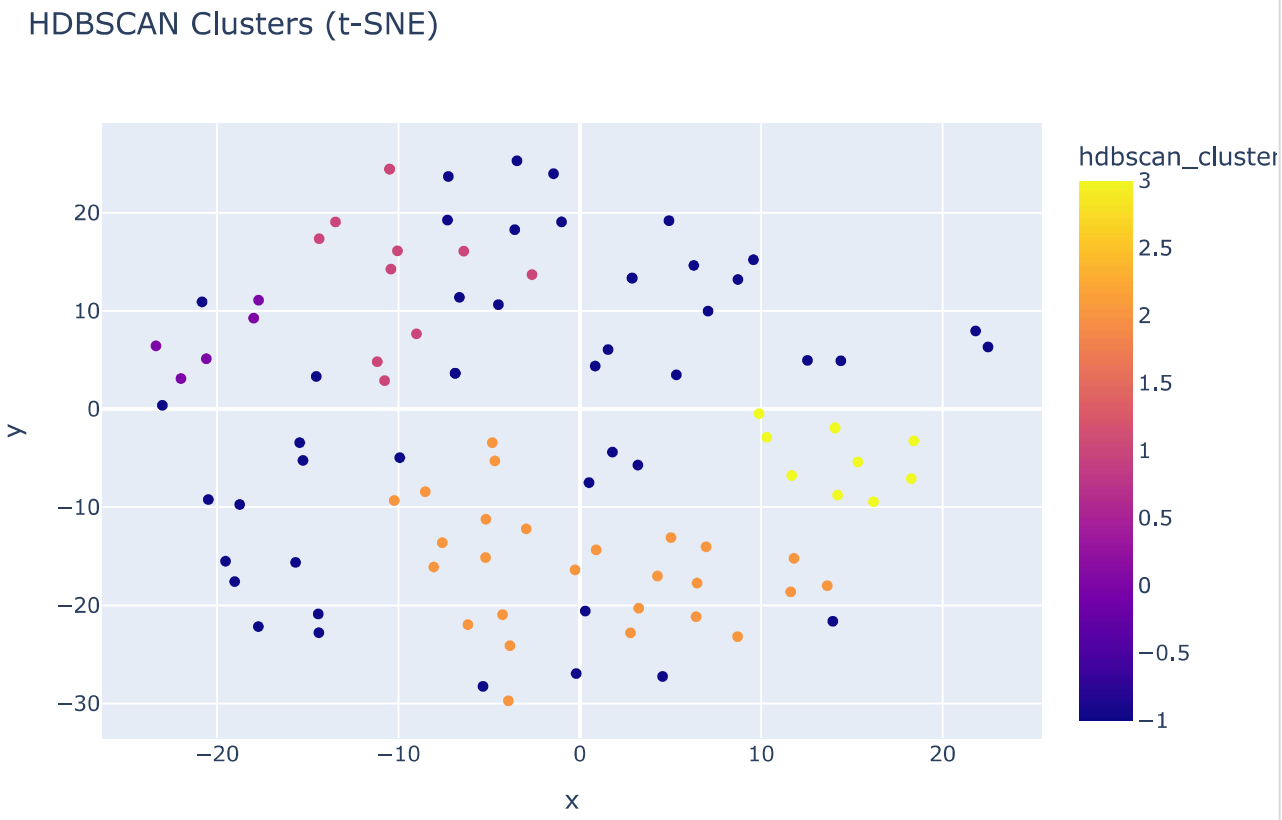


```
# HDBSCAN Clustering

import hdbscan

# HDBSCAN parameters
hdb = hdbscan.HDBSCAN(min_cluster_size=5)
df['hdbscan_cluster'] = hdb.fit_predict(embeddings)

# Check number of clusters and noise
num_clusters = len(set(hdb.labels_)) - (1 if -1 in hdb.labels_ else 0)
num_noise = list(hdb.labels_).count(-1)

print(f"✅ HDBSCAN clustering complete")
print(f"Number of clusters found: {num_clusters}")
print(f"Number of noise points: {num_noise}")

df[['clean_text', 'hdbscan_cluster']].head()
```

```
✅ HDBSCAN clustering complete
Number of clusters found: 4
Number of noise points: 46
/usr/local/lib/python3.12/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.

/usr/local/lib/python3.12/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning:

'force_all_finite' was renamed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.
```

	clean_text	hdbscan_cluster
0	love product	-1
1	service terrible	2
2	movie amazing	1
3	im disappointed customer support	2
4	best meal life	-1

```
# t-SNE Visualization for HDBSCAN

fig = px.scatter(
    df, x='x', y='y',
    color='hdbscan_cluster',
    hover_data=['Text','Sentiment','sentiment_pred'],
    title="HDBSCAN Clusters (t-SNE)"
)
fig.show()
# print("✅ Skipping t-SNE visualization for HDBSCAN")
```



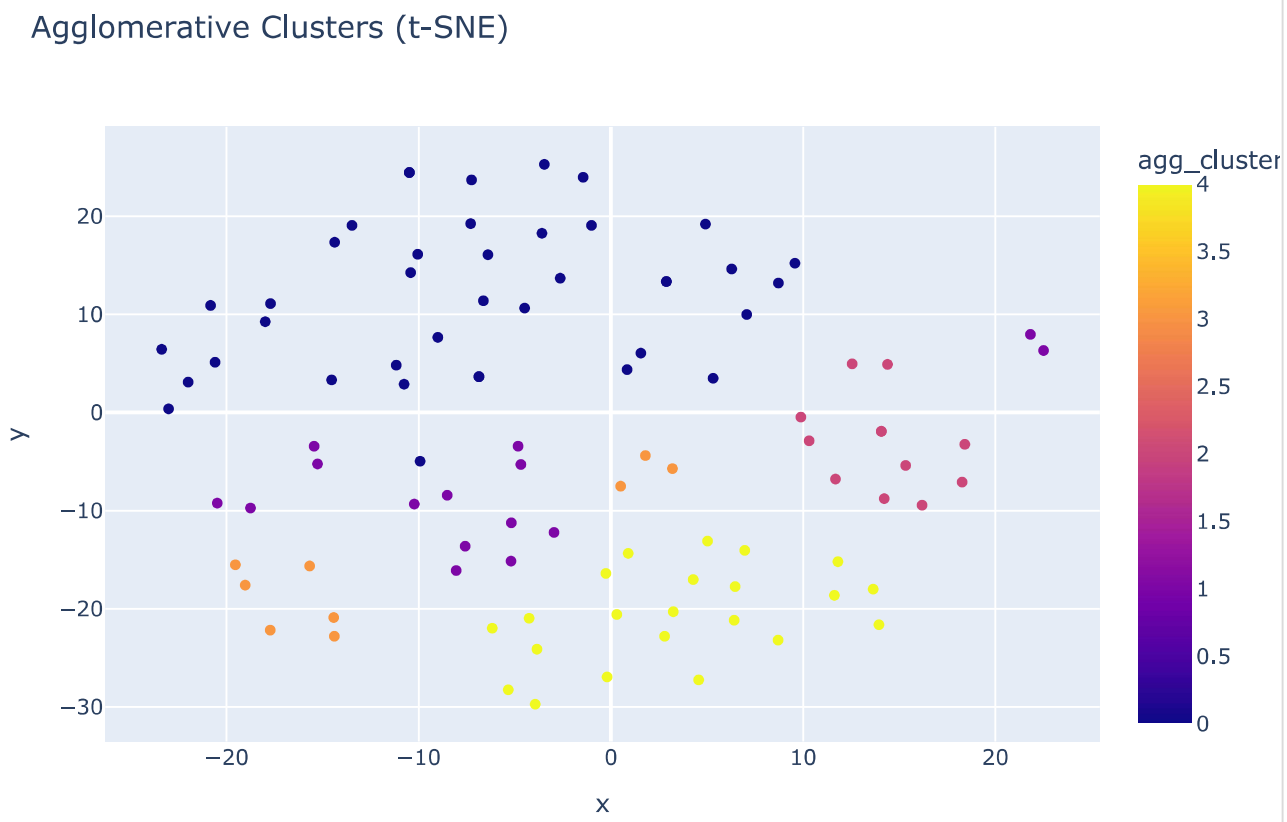

```
# Agglomerative Clustering
from sklearn.cluster import AgglomerativeClustering

agg = AgglomerativeClustering(n_clusters=optimal_k)
df['agg_cluster'] = agg.fit_predict(embeddings)

print("✅ Agglomerative Clustering complete")
print("Number of clusters:", df['agg_cluster'].nunique())
df['agg_cluster'].value_counts()
```

```
✅ Agglomerative Clustering complete
Number of clusters: 5
```

agg_cluster	count
0	40
4	22
1	15
2	12
3	9

dtype: int64

```
# Visualization (Agglomerative)
fig = px.scatter(
    df, x='x', y='y',
    color='agg_cluster',
    hover_data=['Text','Sentiment','sentiment_pred'],
    title="Agglomerative Clusters (t-SNE)"
)
fig.show()
# print("✅ Skipping t-SNE visualization for Agglomerative Clustering")
```



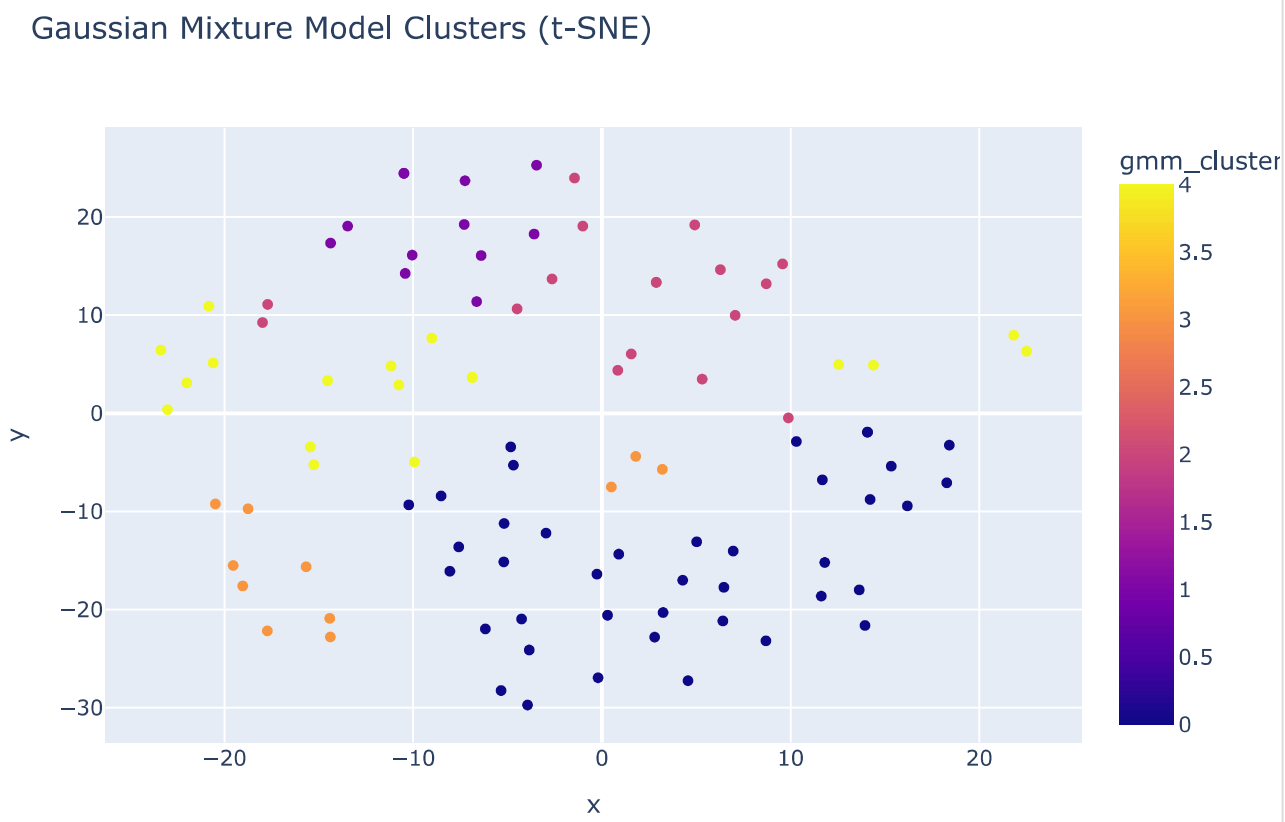

```
# Gaussian Mixture Model
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=optimal_k, random_state=42)
df['gmm_cluster'] = gmm.fit_predict(embeddings)

print("✅ GMM Clustering complete")
print("Number of clusters:", df['gmm_cluster'].nunique())
df['gmm_cluster'].value_counts()
```

```
✅ GMM Clustering complete
Number of clusters: 5
```

gmm_cluster	count
0	40
4	18
2	17
1	12
3	11

dtype: int64

```
# Visualization (GMM)
fig = px.scatter(
    df, x='x', y='y',
    color='gmm_cluster',
    hover_data=['Text','Sentiment','sentiment_pred'],
    title="Gaussian Mixture Model Clusters (t-SNE)"
)
fig.show()
# print("✅ Skipping t-SNE visualization for Gaussian Mixture Model")
```



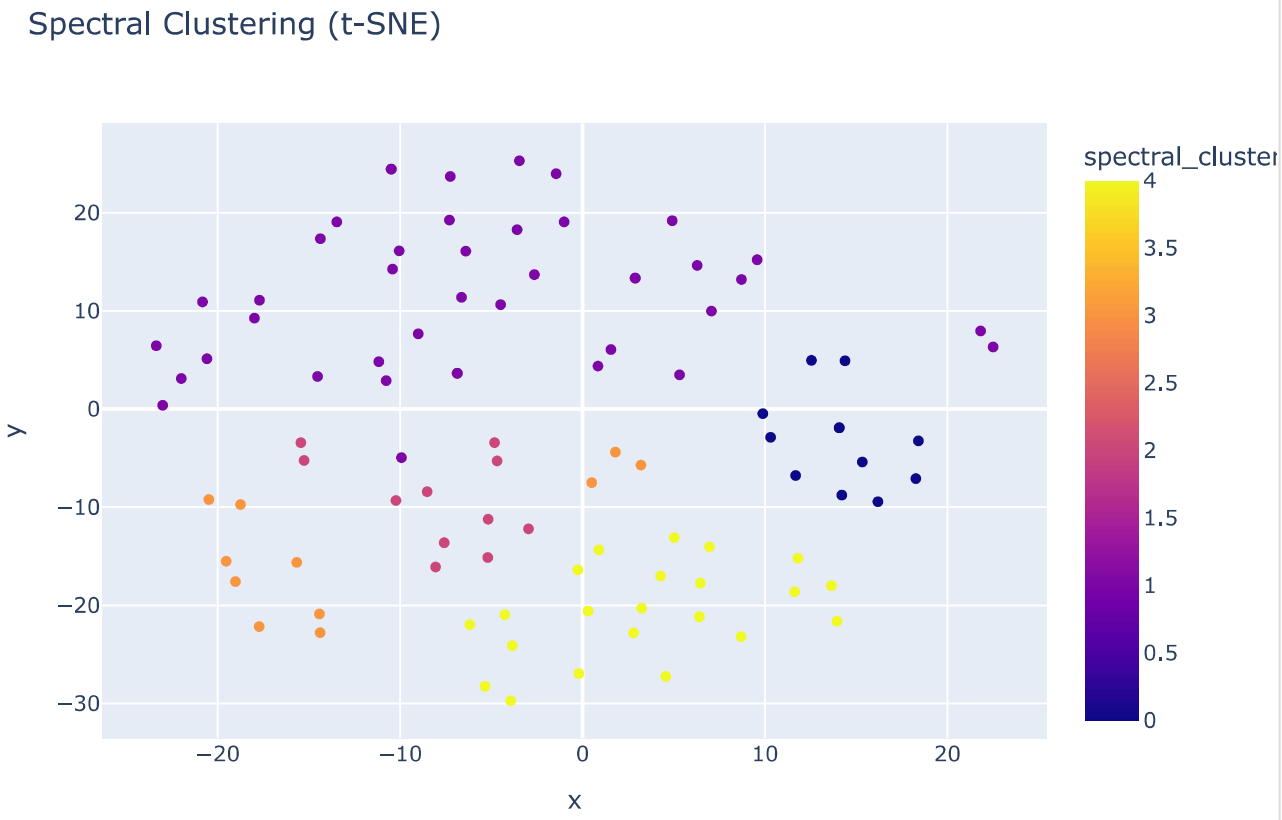
```
# Spectral Clustering
from sklearn.cluster import SpectralClustering

spec = SpectralClustering(n_clusters=optimal_k, assign_labels='kmeans', random_state=42)
df['spectral_cluster'] = spec.fit_predict(embeddings)

print("✅ Spectral Clustering complete")
print("Number of clusters:", df['spectral_cluster'].nunique())
df['spectral_cluster'].value_counts()
```

```
✅ Spectral Clustering complete
Number of clusters: 5
```

spectral_cluster	count
1	42
4	22
0	12
2	11
3	11

dtype: int64

```
# Visualization (Spectral)
fig = px.scatter(
    df, x='x', y='y',
    color='spectral_cluster',
    hover_data=['Text','Sentiment','sentiment_pred'],
    title="Spectral Clustering (t-SNE)"
)
fig.show()
# print("✅ Skipping t-SNE visualization for Spectral Clustering")
```



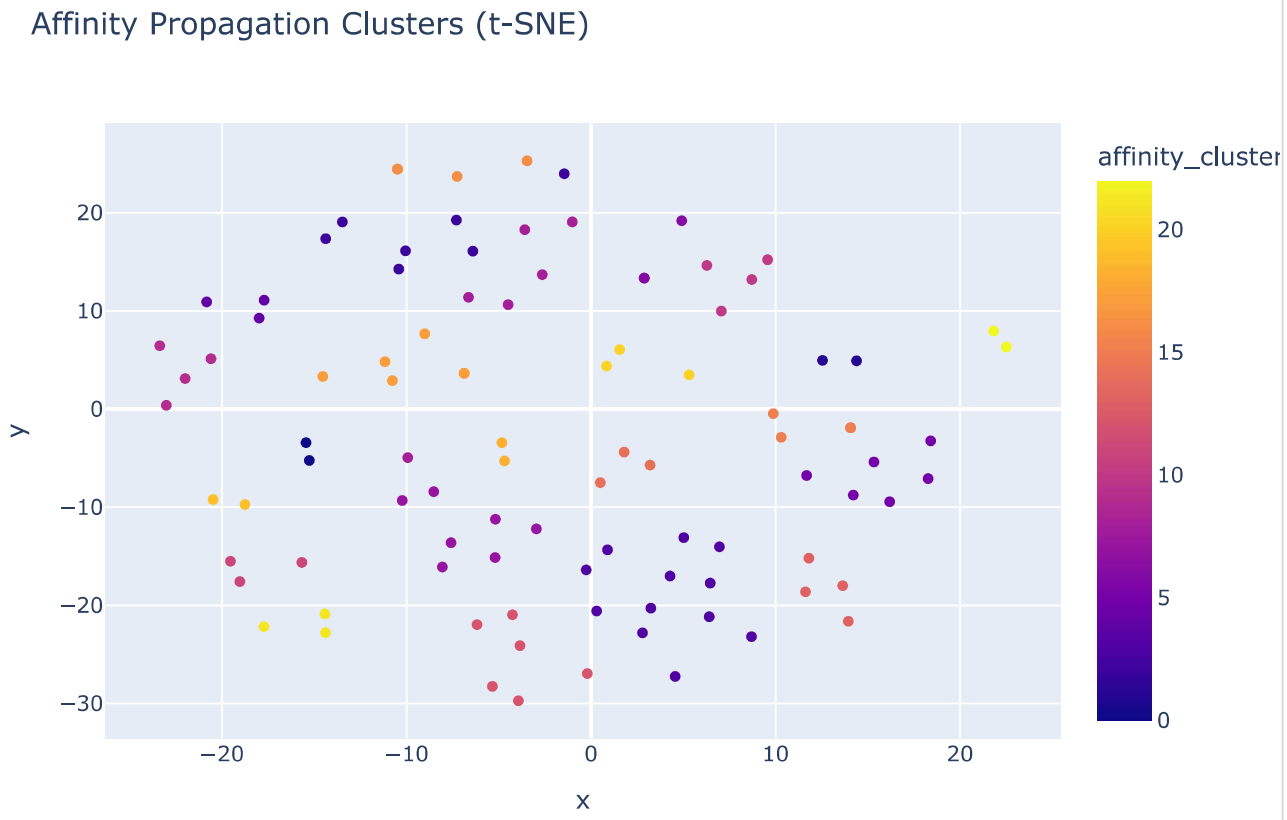

```
# Affinity Propagation
from sklearn.cluster import AffinityPropagation

aff = AffinityPropagation(random_state=42)
df['affinity_cluster'] = aff.fit_predict(embeddings)

print("✅ Affinity Propagation complete")
print("Number of clusters:", df['affinity_cluster'].nunique())
df['affinity_cluster'].value_counts()
```

```
✅ Affinity Propagation complete
Number of clusters: 23
```

affinity_cluster	count
3	12
2	7
7	7
12	6
8	6
5	6
17	6
10	4
15	4
13	4
16	4
9	4
4	3
20	3
6	3
14	3
11	3
21	3
0	2
18	2
1	2
19	2
22	2

dtype: int64

```
# Visualization (Affinity Propagation)
fig = px.scatter(
    df, x='x', y='y',
    color='affinity_cluster',
    hover_data=['Text','Sentiment','sentiment_pred'],
    title="Affinity Propagation Clusters (t-SNE)"
)
fig.show()
# print("✅ Skipping t-SNE visualization for Affinity Propagation")
```

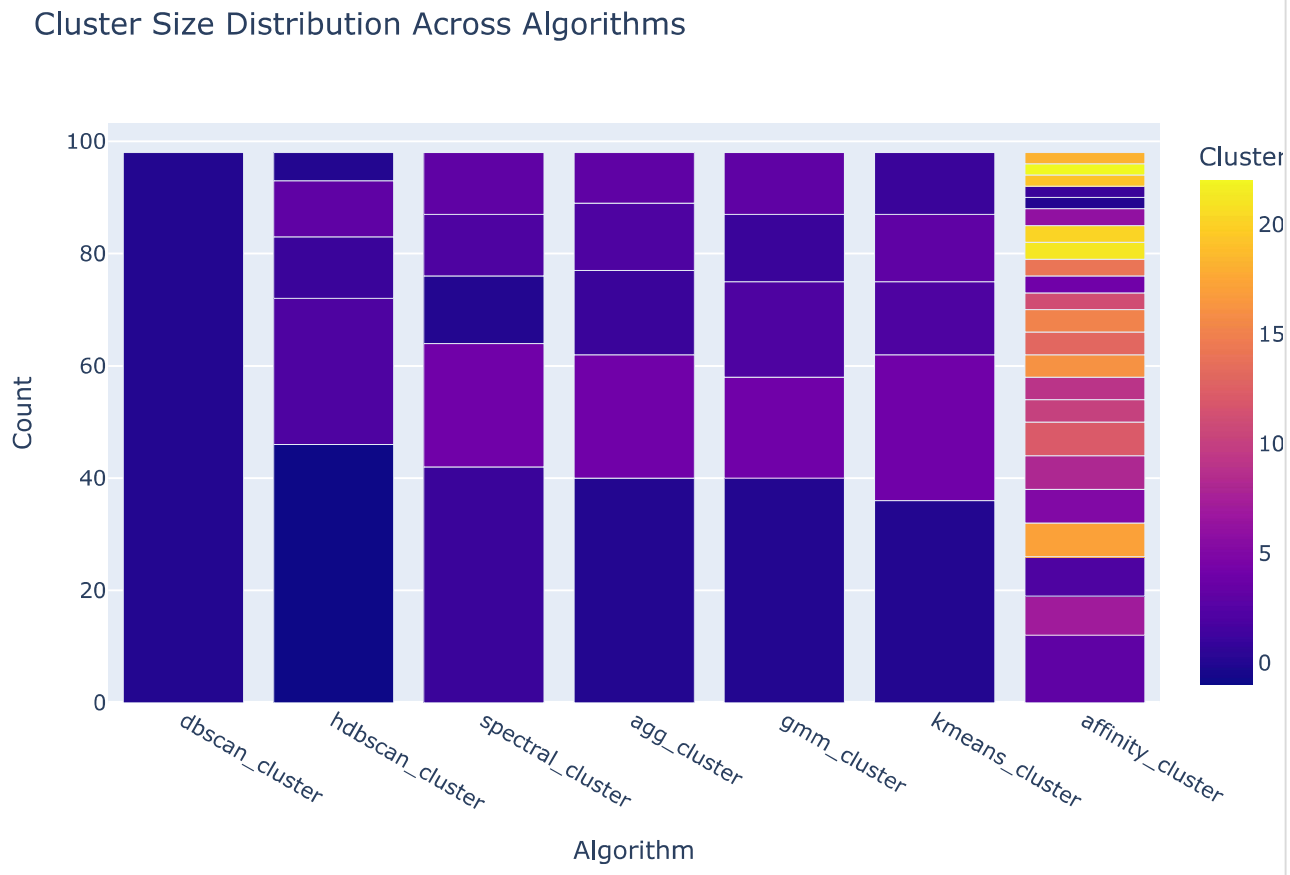



```
# Basic Cluster Overview

cluster_cols = ['kmeans_cluster','dbscan_cluster','hdbscan_cluster','agg_cluster','gmm_cluster'

summary = pd.DataFrame({
    'Algorithm': cluster_cols,
    'Num_Clusters': [df[c].nunique() for c in cluster_cols],
    'Num_NoisePoints': [(df[c] == -1).sum() if df[c].dtype != 'object' else 0 for c in cluster_
})

summary
```

	Algorithm	Num_Clusters	Num_NoisePoints
0	kmeans_cluster	5	0
1	dbscan_cluster	1	0
2	hdbscan_cluster	5	46
3	agg_cluster	5	0
4	gmm_cluster	5	0
5	spectral_cluster	5	0
6	affinity_cluster	23	0

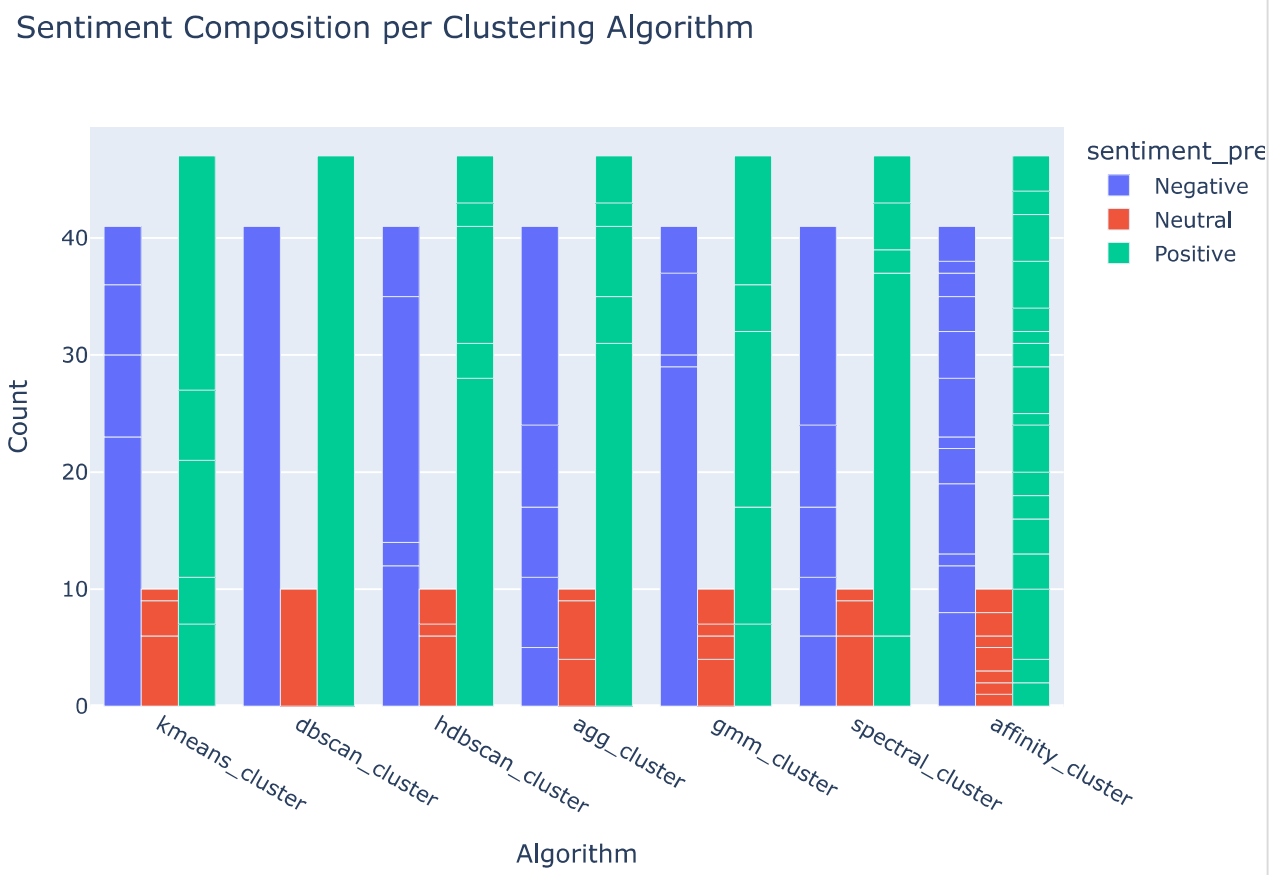
Next steps: Generate code with summary | New interactive sheet

```
# Cluster Size Comparison

cluster_counts = (
    df[cluster_cols]
    .melt(var_name='Algorithm', value_name='Cluster')
    .value_counts()
    .reset_index(name='Count')
)

fig = px.bar(
    cluster_counts,
    x='Algorithm',
    y='Count',
    color='Cluster',
    title="Cluster Size Distribution Across Algorithms"
)
fig.show()
```




```
# Sentiment Distribution per Algorithm

sentiment_compare = []

for algo in cluster_cols:
    tmp = df.groupby([algo, 'sentiment_pred']).size().reset_index(name='Count')
```

```
    tmp['Algorithm'] = algo
    sentiment_compare.append(tmp)

sentiment_compare = pd.concat(sentiment_compare)

fig = px.bar(
    sentiment_compare,
    x='Algorithm',
    y='Count',
    color='sentiment_pred',
    barmode='group',
    title="Sentiment Composition per Clustering Algorithm"
)
fig.show()
```




```
# Cross-Model Agreement

from sklearn.metrics import adjusted_rand_score

pairs = []
for i in range(len(cluster_cols)):
    for j in range(i+1, len(cluster_cols)):
        ari = adjusted_rand_score(df[cluster_cols[i]], df[cluster_cols[j]])
        pairs.append([cluster_cols[i], cluster_cols[j], ari])

ari_df = pd.DataFrame(pairs, columns=['Model_1','Model_2','ARI_Score']).sort_values('ARI_Score'
```

```
ari_df.head(10)
```

	Model_1	Model_2	ARI_Score
16	agg_cluster	spectral_cluster	0.920485
2	kmeans_cluster	agg_cluster	0.583276
3	kmeans_cluster	gmm_cluster	0.570818
4	kmeans_cluster	spectral_cluster	0.559314
18	gmm_cluster	spectral_cluster	0.350545
15	agg_cluster	gmm_cluster	0.327069
1	kmeans_cluster	hdbscan_cluster	0.316297
13	hdbscan_cluster	spectral_cluster	0.288519
11	hdbscan_cluster	agg_cluster	0.251524
12	hdbscan_cluster	gmm_cluster	0.249448

Next steps: Generate code with ari_df | New interactive sheet

Block 14 – Cluster Theme Extraction

After clustering, you get numeric cluster labels like 0, 1, 2, 3... — but those numbers alone don't tell you what each group means. This block helps you interpret each cluster by:

1. Extracting the most frequent and distinctive keywords in each cluster.
2. Showing a sample feedback text from each cluster for context.
3. Optionally, using these insights to name clusters with semantic tags (e.g., "Delivery Issue," "Product Praise").

```
from sklearn.feature_extraction.text import CountVectorizer

def extract_cluster_themes(df, cluster_label, text_col='clean_text', top_n=8):
    print(f"\n🔹 Cluster Theme Extraction for: {cluster_label}\n")
    clusters = df[cluster_label].unique()
    for cluster in clusters:
        if cluster == -1:  # Skip noise
            continue
        subset = df[df[cluster_label] == cluster]
        texts = subset[text_col].tolist()

        # Vectorize text
        vectorizer = CountVectorizer(stop_words='english', max_features=1000)
        X = vectorizer.fit_transform(texts)
        word_freq = np.asarray(X.sum(axis=0)).ravel()
        words = np.array(vectorizer.get_feature_names_out())

        # Get top words
        top_indices = word_freq.argsort()[::-1][:top_n]
        top_words = words[top_indices]
```

```
        print(f"🟩 Cluster {cluster}: ({len(subset)} feedbacks)")
        print("Top Keywords:", ", ".join(top_words))
        print("Sample Feedback:", subset[text_col].iloc[0][:200], "...")
        print("-" * 80)

# Example: Run for KMeans and HDBSCAN
extract_cluster_themes(df, 'kmeans_cluster')
extract_cluster_themes(df, 'hdbscan_cluster')
```

```
🔹 Cluster Theme Extraction for: kmeans_cluster

🟩 Cluster 0: (36 feedbacks)
Top Keywords: customer, service, product, terrible, support, disappointed, quality, im
```