Final Project

Daniel Mehta

Introduction and Problem Statement

Business Problem
Exchange Traded Funds (ETFs) are popular investment products, but creating them manually requires extensive research and portfolio balancing.
This project explores a stock clustering system that automatically groups S&P 500 companies into ETF-style categories based on investment-relevant characteristics such as volatility, return, momentum, and liquidity.

These clusters could help financial institutions or fintech platforms quickly generate investment ideas tailored to different risk profiles and objectives. For example, aggressive growth, stable income, or balanced diversification.

Note on Feature Selection
My initial concept included sector classification as a feature, but the available dataset does not contain sector information.
To maintain simplicity and avoid external data merging, my current implementation focuses solely on numerical, market derived features.

Goal
Develop a stock clustering pipeline that:

1. Processes historical price data for S&P 500 companies.
2. Creates numerical feature vectors for each stock.
3. Applies and compares **K-Means Clustering** (covered in course) and **Hierarchical Clustering** (new) to group the stocks.
4. Evaluates clusters using quantitative metrics and visualizations.
5. Recommends the optimal approach for deployment in an ETF grouping tool.

Imports

In [4]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score, davies_bouldin_score

import scipy.cluster.hierarchy as sch
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
```

Load Dataset / Cleaning

In [6]:
```
file_path = "all_stocks_5yr.csv"
df = pd.read_csv(file_path)
```

In [7]:
```
df.head(),df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 619040 entries, 0 to 619039
Data columns (total 7 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   date    619040 non-null  object 
 1   open    619029 non-null  float64
 2   high    619032 non-null  float64
 3   low     619032 non-null  float64
 4   close   619040 non-null  float64
 5   volume  619040 non-null  int64  
 6   Name    619040 non-null  object 
dtypes: float64(4), int64(1), object(2)
memory usage: 33.1+ MB
```

```
Out[7]: (         date   open   high    low  close    volume Name
         0  2013-02-08  15.07  15.12  14.63  14.75   8407500  AAL
         1  2013-02-11  14.89  15.01  14.26  14.46   8882000  AAL
         2  2013-02-12  14.45  14.51  14.10  14.27   8126000  AAL
         3  2013-02-13  14.30  14.94  14.25  14.66  10259500  AAL
         4  2013-02-14  14.94  14.96  13.16  13.99  31879900  AAL,
         None)
```

```
In [8]: # Converting date to datetime
        df['date'] =pd.to_datetime(df['date'])
```

```
In [9]: # check for missing values
        print("Missing values per column:\n", df.isnull().sum())
```

```
Missing values per column:
 date       0
open      11
high       8
low        8
close      0
volume     0
Name       0
dtype: int64
```

```
In [10]: # Drop rows with any missing open, high, low, close values
         df = df.dropna(subset=['open', 'high', 'low', 'close'])
```

```
In [11]: #Checking correct data types
         numeric_cols = ['open', 'high', 'low', 'close', 'volume']
         df[numeric_cols] = df[numeric_cols].astype(float)
```

```
In [12]: # Removeing any duplicates
         df = df.drop_duplicates(subset=['date', 'Name'])
```

```
In [13]: # Sort by Name then date
         df =df.sort_values(by=['Name','date']).reset_index(drop=True)
```

```
In [14]: print(f"Cleaned dataset shape: {df.shape}")
         df.head()
```

```
Cleaned dataset shape: (619029, 7)
```

Out[14]:

	date	open	high	low	close	volume	Name
0	2013-02-08	45.07	45.35	45.00	45.08	1824755.0	A
1	2013-02-11	45.17	45.18	44.45	44.60	2915405.0	A
2	2013-02-12	44.81	44.95	44.50	44.62	2373731.0	A
3	2013-02-13	44.81	45.24	44.68	44.75	2052338.0	A
4	2013-02-14	44.72	44.78	44.36	44.58	3826245.0	A

Feature Engineering

In [16]:
```
# daily returns
df['daily_return'] = (df['close']-df['open'])/df['open']
```

In [17]:
```
# Group by stock ticker
features_df = df.groupby('Name').apply(lambda x: pd.Series({
    'avg_daily_return': x['daily_return'].mean(),
    'volatility': x['daily_return'].std(),
    'momentum_30d': (x['close'].iloc[-1] - x['close'].iloc[-30])/x['close'].iloc[-30]
                    if len(x) >=30 else np.nan,
    'avg_volume': x['volume'].mean(),
    'max_drawdown': (x['close'].cummax() - x['close']).max()/x['close'].cummax().max()
})).reset_index()
```

```
/var/folders/xb/2tg9ddl94wl284px7ngj8hn40000gn/T/ipykernel_67731/3980291536.py:2: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and i
n a future version of pandas the grouping columns will be excluded from the operation. Either pass `include_groups=False` to exclude the groupings or explicitly select the grouping columns afte
r groupby to silence this warning.
  features_df = df.groupby('Name').apply(lambda x: pd.Series({
```

In [18]:
```
# Dropping rows with NaN
features_df =features_df.dropna()
```

In [19]:
```
print(f"Feature dataset shape: {features_df.shape}")
features_df.head()
```

```
Feature dataset shape: (505, 6)
```

Out[19]:

	Name	avg_daily_return	volatility	momentum_30d	avg_volume	max_drawdown
0	A	0.000421	0.011578	0.012045	2.338039e+06	0.368351
1	AAL	0.000312	0.019596	-0.027436	9.390321e+06	0.521464
2	AAP	0.000036	0.013973	0.078168	1.078043e+06	0.603853
3	AAPL	0.000139	0.011863	-0.064666	5.404790e+07	0.237978
4	ABBV	0.001016	0.014834	0.162353	7.870683e+06	0.186349

StandardScaler

In [21]:
```
# Scaling to prevent bias clusters toward features with larger ranges
# only numeric columns for scaling
X = features_df.drop(columns=['Name'])

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Store in df
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
X_scaled_df.head()
```

Out[21]:

	avg_daily_return	volatility	momentum_30d	avg_volume	max_drawdown
0	0.192156	-0.332729	0.295485	-0.280768	0.291482
1	-0.087933	1.844891	-0.233007	0.724016	1.140115
2	-0.795181	0.317619	1.180616	-0.460288	1.596756
3	-0.531922	-0.255375	-0.731363	7.086672	-0.431109
4	1.720948	0.551432	2.307523	0.507503	-0.717269

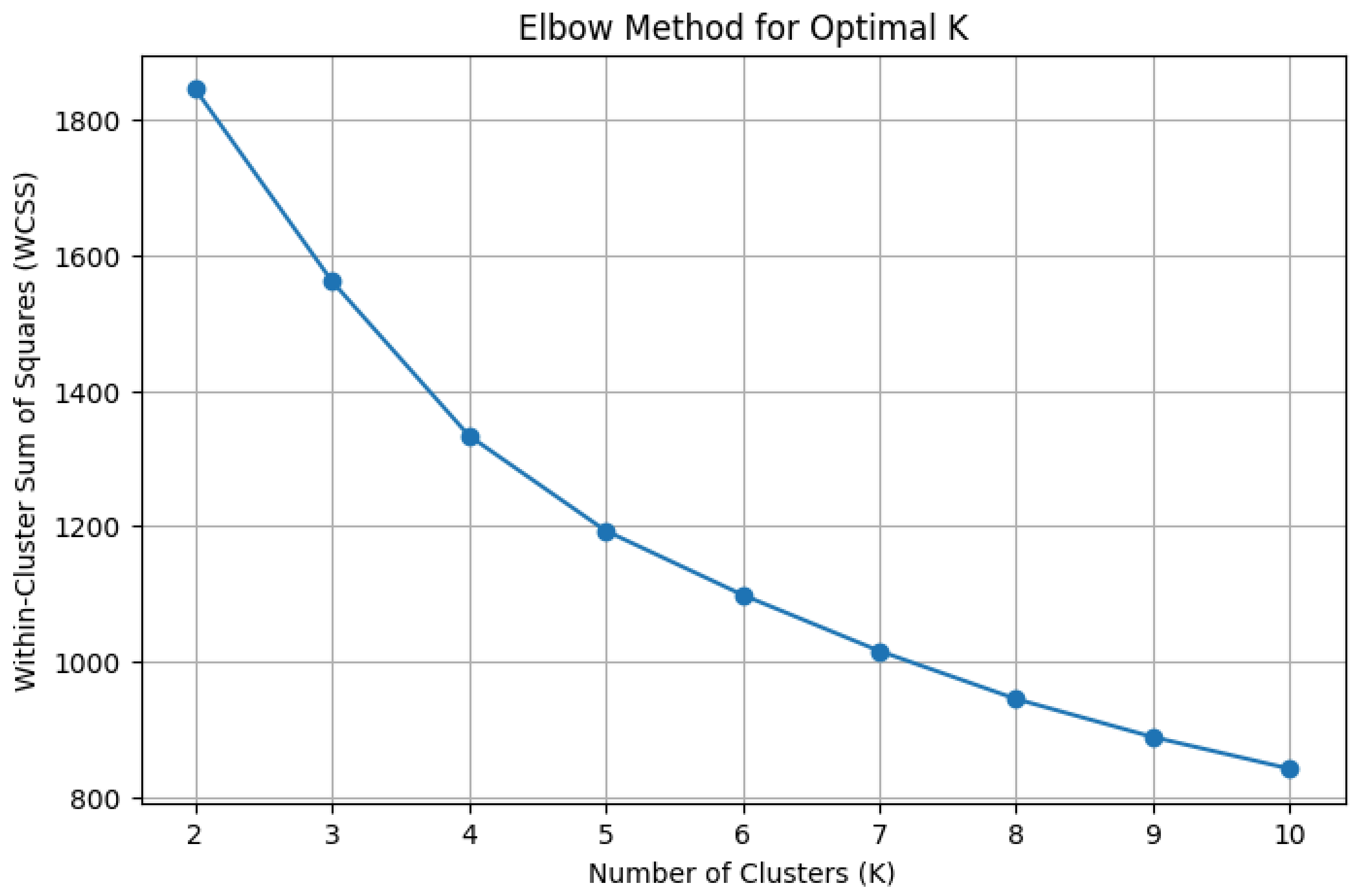
Elbow Method & K Selection

In [23]:
```
X_for_kmeans = X_scaled_df
```

In [24]:
```
# testing k val from 2 to 10
wcss = []
K_range = range(2,11)

for k in K_range:
    kmeans = KMeans(n_clusters=k, random_state=5503, n_init=10)
    kmeans.fit(X_for_kmeans)
    wcss.append(kmeans.inertia_)
```

In [25]:
```
plt.figure(figsize=(8, 5))
plt.plot(K_range, wcss, marker='o')
plt.xlabel('Number of Clusters (K)')
plt.ylabel('Within-Cluster Sum of Squares (WCSS)')
plt.title('Elbow Method for Optimal K')
plt.grid(True)
plt.show()
```



In [26]:
```
for k, score in zip(K_range, wcss):
    print(f"K={k}: WCSS={score:.2f}")
```

```
K=2: WCSS=1845.09
K=3: WCSS=1561.31
K=4: WCSS=1333.58
K=5: WCSS=1193.14
K=6: WCSS=1098.54
K=7: WCSS=1015.73
K=8: WCSS=944.82
K=9: WCSS=888.68
K=10: WCSS=841.93
```

In [27]:
```
# I chose K=4 as it marks the point where WCSS improvement significantly tapers off
```

K-Means Clustering

In [29]:
```
#KMeans Clustering
kmeans = KMeans(n_clusters=4, random_state=5503, n_init=10)
features_df['kmeans_cluster'] = kmeans.fit_predict(X_scaled_df)
```

In [30]:
```
print(features_df['kmeans_cluster'].value_counts())
```

```
kmeans_cluster
3    228
2    178
0     85
1     14
Name: count, dtype: int64
```

In [31]:
```python
# cluster centers represent the average scaled feature values for each cluster
# shows how groups differ across return, volatility, momentum, volume, and drawdown.
centers = pd.DataFrame(kmeans.cluster_centers_, columns=X_scaled_df.columns)
print("\nK-Means Cluster Centers (scaled features):\n", centers)
```

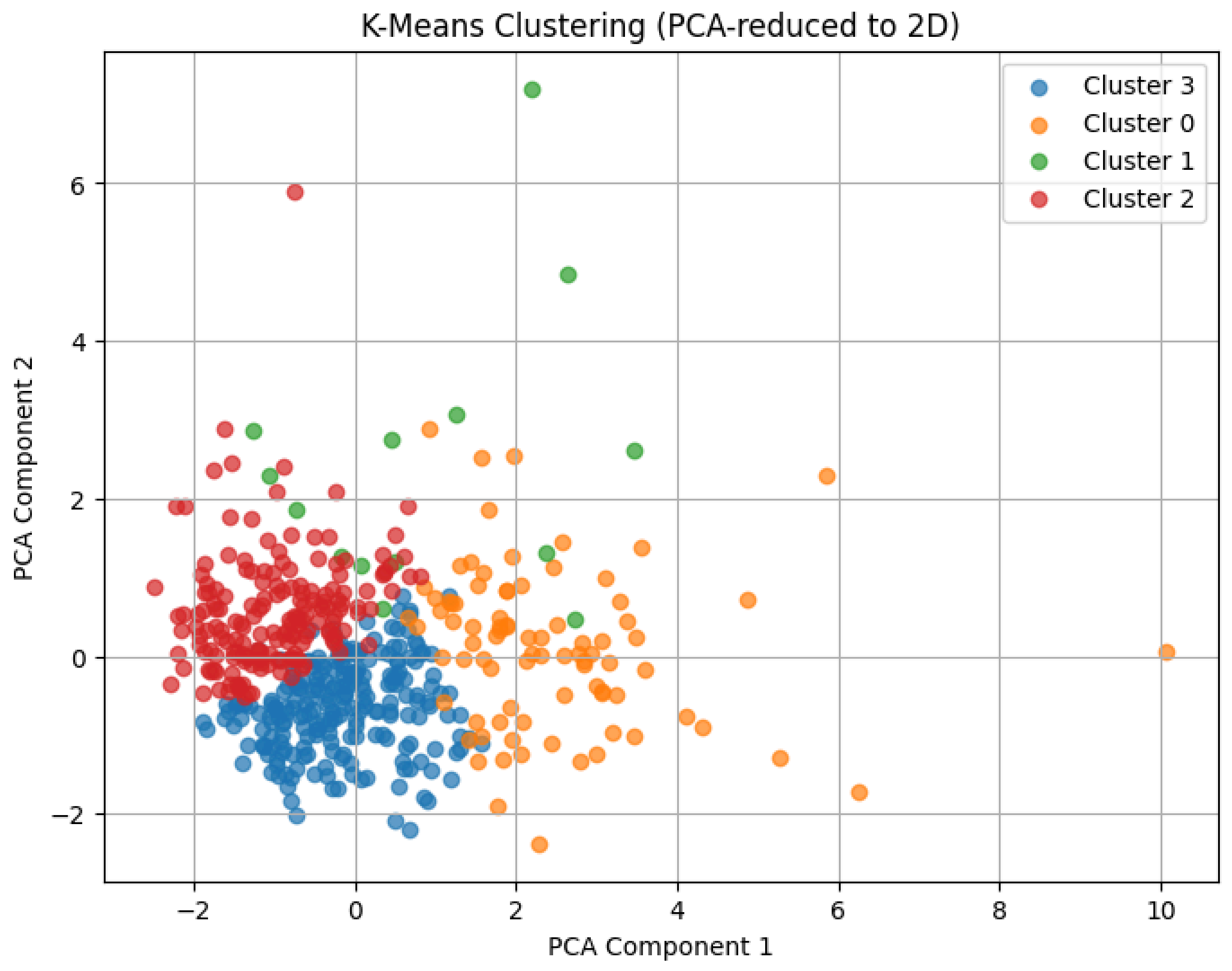
```
K-Means Cluster Centers (scaled features):
    avg_daily_return  volatility  momentum_30d  avg_volume  max_drawdown
0         -1.044587    1.496965     -0.337898   -0.009384      1.636725
1         -0.425539    0.103140     -0.040587    4.594347     -0.226703
2          0.641201   -0.219502      0.833602   -0.147130     -0.528005
3         -0.085028   -0.393047     -0.522331   -0.163746     -0.184048
```

In [32]:
```python
# Reduceing features to 2D for visualization
pca = PCA(n_components=2)
pca_result = pca.fit_transform(X_scaled_df)
```

In [33]:
```python
plot_df = pd.DataFrame({
    'PCA1': pca_result[:, 0],
    'PCA2': pca_result[:, 1],
    'Cluster': features_df['kmeans_cluster']
})
```

In [34]:
```python
plt.figure(figsize=(8, 6))
for cluster in plot_df['Cluster'].unique():
    cluster_data = plot_df[plot_df['Cluster'] == cluster]
    plt.scatter(cluster_data['PCA1'], cluster_data['PCA2'], label=f'Cluster {cluster}', alpha=0.7)

plt.title('K-Means Clustering (PCA-reduced to 2D)')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.legend()
plt.grid(True)
plt.show()
```



K Means Interpretation:

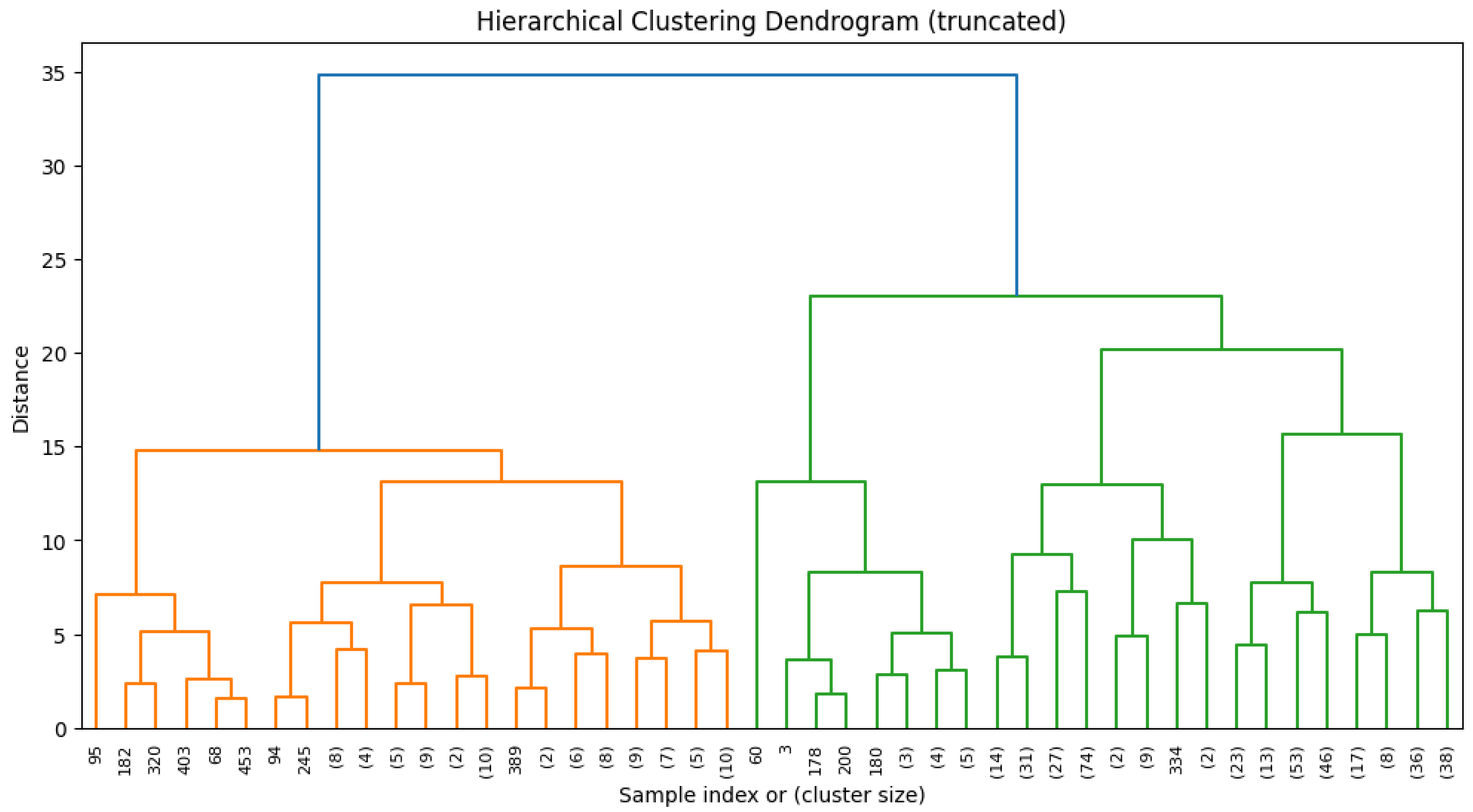
- The PCA plot reduces the 5 dimensional feature space to 2 dimensions for visualization.
- Each point represents a stock, colored by its assigned K-Means cluster (K=4)
- Clusters 2 (red) and 3 (blue) show dense, well-defined groupings, while Cluster 0 (orange) is more spread out, suggesting greater diversity in stock characteristics
- Cluster 1 (green) is very small (14 stocks), indicating a niche group with unique feature patterns
- Point separation in this 2D space suggests that the chosen features and K value produce meaningful differentiation between stock groups

---

Hierarchical Clustering

In [37]:
```
# Generating the linkage matrix
linkage_matrix = linkage(X_scaled_df, method='ward') # used ward to keep clusters compact and low variance
```

In [38]:
```
# Plotting dendrogram
plt.figure(figsize=(12, 6))
dendrogram(linkage_matrix, truncate_mode='level',p=5) # top 5 merge levels
plt.title('Hierarchical Clustering Dendrogram (truncated)')
plt.xlabel('Sample index or (cluster size)')
plt.ylabel('Distance')
plt.show()
```



In [39]:
```
# Cut the dendrogram at K=4
# Set t=4 to match the K=4 from KMeans for a fair comparison between methods
# criterion='maxclust' to make sure t=4 is interpreted as the desired number of clusters
hier_clusters = fcluster(linkage_matrix, t=4, criterion='maxclust')
```

In [40]:
```
#Add cluster assignments to df
features_df['hier_cluster'] =hier_clusters
```

In [41]:
```
print(features_df['hier_cluster'].value_counts())
```

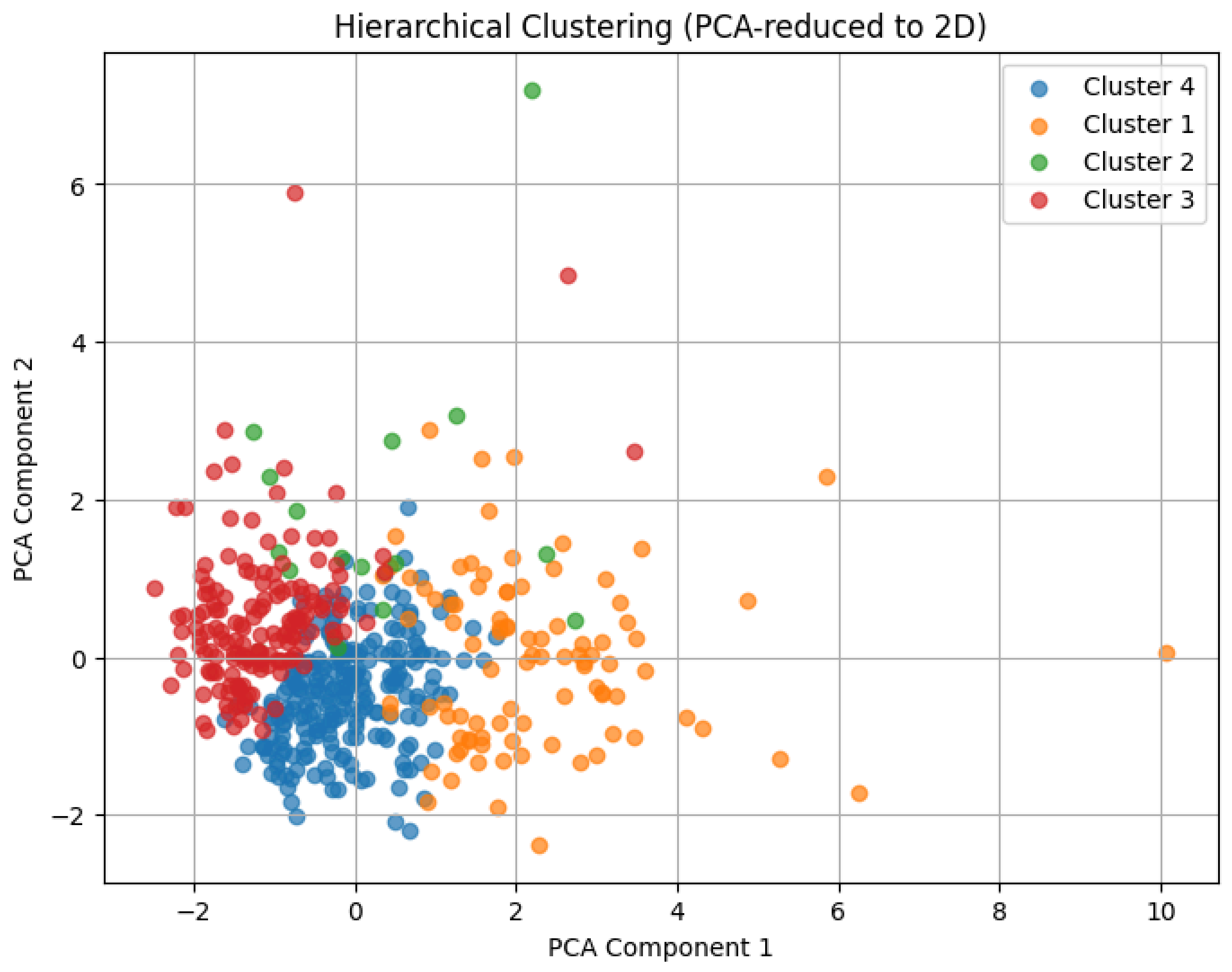
```
hier_cluster
4    234
3    160
1     94
2     17
Name: count, dtype: int64
```

In [42]:
```
# PCA scatterplot for Hierarchical Clustering
pca_result_hier= pca.transform(X_scaled_df)  # reusing the PCA model from earlier
```

In [43]:
```
# df for plotting
plot_df_hier = pd.DataFrame({
    'PCA1': pca_result_hier[:,0],
    'PCA2': pca_result_hier[:,1],
    'Cluster': features_df['hier_cluster']
})
```

```
In [44]: plt.figure(figsize=(8, 6))
         for cluster in plot_df_hier['Cluster'].unique():
             cluster_data = plot_df_hier[plot_df_hier['Cluster']==cluster]
             plt.scatter(cluster_data['PCA1'], cluster_data['PCA2'], label=f'Cluster {cluster}',alpha=0.7)

         plt.title('Hierarchical Clustering (PCA-reduced to 2D)')
         plt.xlabel('PCA Component 1')
         plt.ylabel('PCA Component 2')
         plt.legend()
         plt.grid(True)
         plt.show()
```



Hierarchical Clustering Interpretation:

- The PCA plot shows the 5 dimensional feature space reduced to 2 dimensions, with points colored by their Hierarchical Clustering group (K=4)
- Cluster 4 (blue) and Cluster 3 (red) form the two largest, denser groups, representing the bulk of typical stock profiles
- Cluster 1 (orange) is moderately sized but shows more spread, indicating higher internal variability
- Cluster 2 (green) is very small (17 stocks), suggesting a niche set of companies with unique return, volatility, or momentum patterns
- Compared to K-Means, the clusters here appear less compact, reflecting how hierarchical clustering does not optimize for spherical
- separation but instead builds nested groupings.

Evaluation (Silhouette, Davies-Bouldin)

In [47]:
```
# K means metrics
silhouette_kmeans = silhouette_score(X_scaled_df,features_df['kmeans_cluster'])
db_kmeans = davies_bouldin_score(X_scaled_df,features_df['kmeans_cluster'])
```

In [48]:
```
# Hierarchical metrics
silhouette_hier = silhouette_score(X_scaled_df,features_df['hier_cluster'])
db_hier = davies_bouldin_score(X_scaled_df,features_df['hier_cluster'])
```

In [49]:
```
metrics_df = pd.DataFrame({
    'Algorithm': ['K-Means', 'Hierarchical'],
    'Silhouette Score (higher=better)': [silhouette_kmeans, silhouette_hier],
    'Davies-Bouldin Index (lower=better)': [db_kmeans, db_hier]
})

metrics_df
```

Out[49]:

	Algorithm	Silhouette Score (higher=better)	Davies-Bouldin Index (lower=better)
0	K-Means	0.216312	1.303582
1	Hierarchical	0.181917	1.430785

Evaluation Summary

Silhouette Score (higher = better)

- K-Means = 0.216 is higher than Hierarchical = 0.182 -> K-Means clusters are slightly better separated and more cohesive
- Both scores are low (max is 1), which is common in financial datasets where boundaries between groups are fuzzy

Davies-Bouldin Index (lower = better)

- K-Means = 1.30 is lower than Hierarchical = 1.43 -> K-Means clusters are more compact and distinct on average.

Conclusion

- K-Means outperforms Hierarchical Clustering on both metrics for this dataset, suggesting it produces cleaner, more well-defined clusters for the ETF grouping task.

Note:

- In a real world finance setting, the best evaluation would be backtesting cluster based ETF portfolios against the S&P 500, but for this project evaluation is limited to clustering quality metrics and visual separation

---

Dashboard Visualizations

Cluster summary table in original units

In [53]:
```
# a table of mean feature values per cluster (original units, not scaled)
orig_feats = features_df[['Name','kmeans_cluster','avg_daily_return','volatility','momentum_30d','avg_volume','max_drawdown']]

cluster_summary = (
    orig_feats
    .groupby('kmeans_cluster')[['avg_daily_return','volatility','momentum_30d','avg_volume','max_drawdown']]
    .mean()
    .sort_index()
    .round(4)
)

print("Cluster Summary (original units)")
display(cluster_summary)
```

```
Cluster Summary (original units)
```

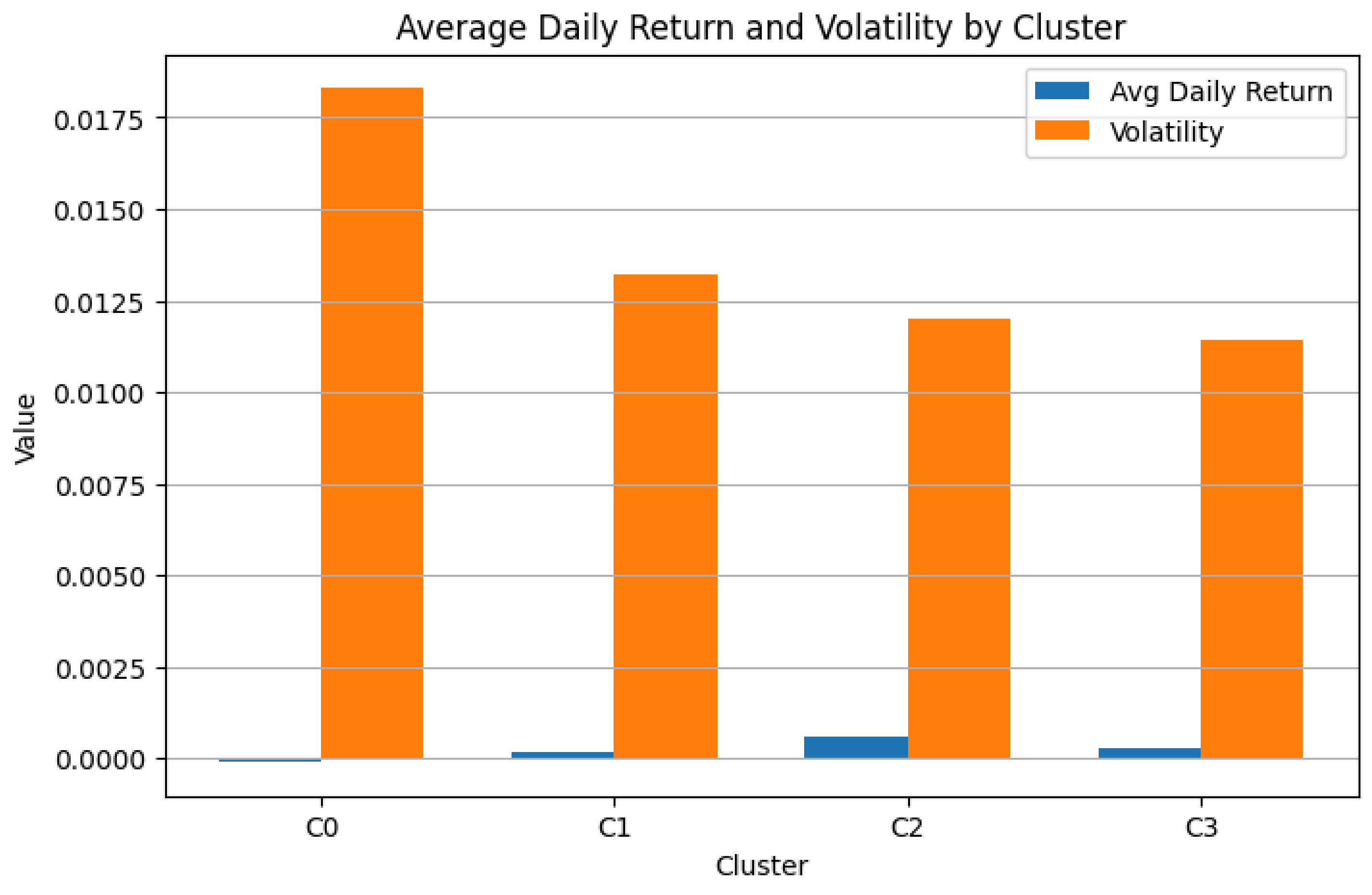
kmeans_cluster	avg_daily_return	volatility	momentum_30d	avg_volume	max_drawdown
0	-0.0001	0.0183	-0.0353	4.242808e+06	0.6111
1	0.0002	0.0132	-0.0131	3.655502e+07	0.2749
2	0.0006	0.0120	0.0522	3.276005e+06	0.2205
3	0.0003	0.0114	-0.0490	3.159385e+06	0.2826

Bar chart: average return vs volatility by cluster

In [55]:
```
bar_data = cluster_summary[['avg_daily_return','volatility']]

x = np.arange(len(bar_data))# cluster indices
width = 0.35


plt.figure(figsize=(8,5))
plt.bar(x - width/2, bar_data['avg_daily_return'], width, label='Avg Daily Return')
plt.bar(x + width/2, bar_data['volatility'], width, label='Volatility')
plt.xticks(x, [f'C{k}' for k in bar_data.index])
plt.xlabel('Cluster')
plt.ylabel('Value')
plt.title('Average Daily Return and Volatility by Cluster')
plt.legend()
plt.grid(True, axis='y')
plt.show()
```

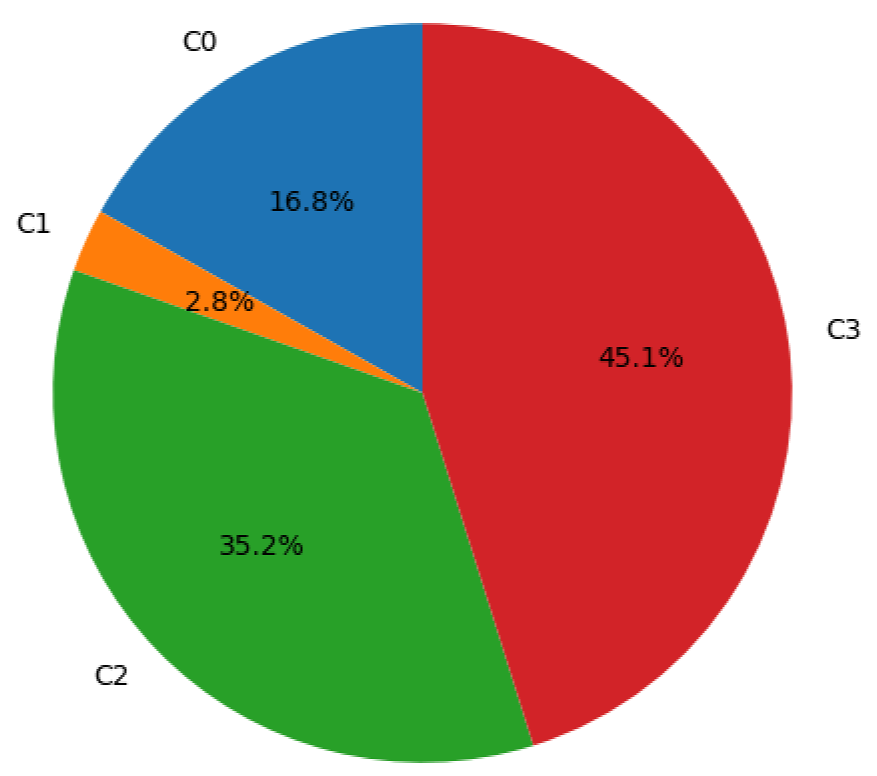


Pie chart: cluster size distribution

In [57]:
```
sizes = features_df['kmeans_cluster'].value_counts().sort_index()
labels = [f'C{k}' for k in sizes.index]

plt.figure(figsize=(6,6))
plt.pie(sizes.values, labels=labels, autopct='%1.1f%%', startangle=90)
plt.title('Cluster Size Distribution')
plt.show()
```

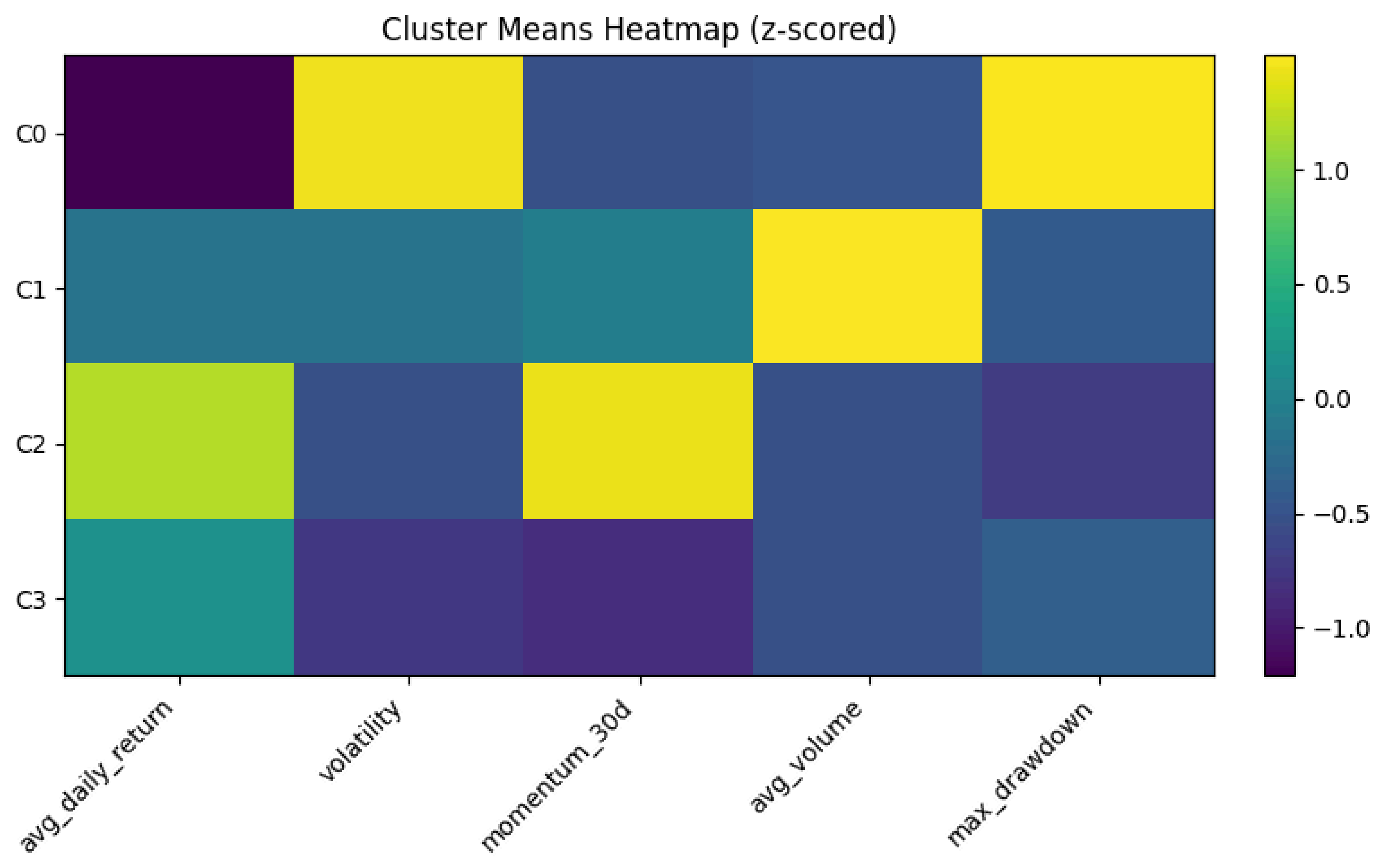
Cluster Size Distribution



Heatmap of feature means by cluster

In [59]:
```
# Z score the cluster means per feature for visualization
means = cluster_summary.copy()
means_z = (means - means.mean(axis=0))/means.std(axis=0)

plt.figure(figsize=(8,5))
im = plt.imshow(means_z.values, aspect='auto')
plt.colorbar(im, fraction=0.046, pad=0.04)
plt.xticks(range(means_z.shape[1]), means_z.columns, rotation=45, ha='right')
plt.yticks(range(means_z.shape[0]), [f'C{k}' for k in means_z.index])
plt.title('Cluster Means Heatmap (z-scored)')
plt.tight_layout()
plt.show()
```



Interpretation

- **Bar Chart (Average Daily Return vs. Volatility)**
  Cluster 0 shows the highest volatility but near zero average daily return, suggesting high risk, low reward stocks.
  Cluster 2 has the highest average daily return with lower volatility, indicating potentially more attractive risk adjusted performance.
  Clusters 1 and 3 show moderate volatility with mixed returns.

- **Pie Chart (Cluster Size Distribution)**
  Cluster 3 contains the largest share of stocks (45.1%), representing the most common profile in the S&P 500 dataset.
  Cluster 1 is very small (2.8%), suggesting a niche set of companies with unique characteristics.

- **Heatmap (Z-scored Cluster Means)**
  Cluster 0 stands out for extremely high volatility and high maximum drawdown, combined with weak returns.
  Cluster 1 is notable for extremely high trading volume and lower drawdowns.
  Cluster 2 leads in both average return and momentum while maintaining relatively low volatility.
  Cluster 3 has moderate values across most metrics but slightly negative momentum.

- **Cluster Summary Table (Original Units)**
  Confirms the patterns observed in the visualizations:
  Cluster 2 has the strongest return/momentum profile, Cluster 0 has the highest volatility/drawdown, Cluster 1 dominates in trading volume, and Cluster 3 is balanced but negative in momentum.

Recommendation Summary

Preprocessing Pipeline

- Clean OHLCV stock data by removing missing values and duplicates, then sort by ticker and date.
- Engineer features: average daily return, volatility, 30-day momentum, average volume, and maximum drawdown.
- Standardize features using `StandardScaler` to ensure equal weight in clustering.

Recommended AI Algorithm

- **K-Means Clustering (K=4)** is recommended.
- Outperformed Hierarchical Clustering in both Silhouette Score (0.216 vs. 0.182) and Davies-Bouldin Index (1.30 vs. 1.43).
- PCA plots show more compact and separable clusters, improving interpretability for ETF grouping.

Dashboard Visualizations

- PCA scatterplot for cluster separation.
- Bar chart of average return vs. volatility to compare cluster risk/return profiles.
- Pie chart of cluster sizes to show distribution across groups.
- Heatmap of z-scored feature means for quick identification of distinctive cluster traits.
- These visuals allow end users (e.g. portfolio managers) to quickly understand the characteristics and relative appeal of each cluster.

Limitations and Ethics

Limitations

- **No Performance Guarantee:** Clusters are based solely on historical patterns in returns, volatility, momentum, volume, and drawdown. There is no guarantee that these groupings will outperform or behave the same way in the future.
- **Missing Data Dimensions:** Model does not incorporate stock correlations or fundamental indicators (e.g., P/E ratio, earnings growth), which could improve cluster quality.
- **No Backtesting:** Evaluation is limited to clustering metrics (Silhouette, Davies-Bouldin) rather than portfolio performance against benchmarks such as the S&P 500.
- **Static Snapshot:** Features are calculated on the full dataset as a whole; dynamic, rolling-window clustering could better adapt to changing market conditions.

Ethical Considerations

- **Not Financial Advice:** The stock groupings in this project are for academic purposes only and should be viewed as an example of data analysis, not as actual investment recommendations.
- **Risk Awareness:** Users should be informed that investing based on historical patterns carries risk, especially if the market regime changes.
- **Transparency:** Clearly document feature definitions, clustering methodology, and limitations so users understand how the groupings were formed.
- **Bias Mitigation:** Ensure the model does not systematically favor or exclude certain sectors or company sizes without valid financial justification.

```
In [ ]:
```