We filter out funds with incomplete time periods as this is a small number and makes handling of flattening slightly easier. We drop features with more than 20% of data missing.

```
data_cleaned = data_cleaned.groupby('Security ID').filter(lambda x: x['Date'].nunique() == 3)

data_cleaned = data.dropna(thresh=(len(data) * missing_threshold), axis=1)
```

Feature Engineering: An important part of how to identify or categorise financial funds is their performance and volatility as well as their exposure to different financial instruments. The last of these is already provided in the data, so we provide some features to help the model delimit on performance and volatility. We fill NaN values (March has no previous month for performance reference) with a constant so the model doesn't fit any variance on this feature. We use a small value just to avoid any strong associations with 0.

```
data_cleaned['NAV_pct_change'] = data_cleaned.groupby('Security ID')['NAV (Daily - USD)'].pct_change()
data_cleaned['NAV_pct_change'].fillna(1e-8, inplace=True)
```

Impute missing values for numeric data on a per-fund basis using the median. This isn't an perfect strategy but PCA dimensionality reduction can't be done with NaN values. The number of NaNs is small so we hope this doesn't affect the clustering too much.

```
numeric_cols = data_cleaned.select_dtypes(include=[np.number]).columns
    data_cleaned[numeric_cols] = data_cleaned.groupby('Security ID')[numeric_cols].transform(
        lambda x: x.fillna(x.median() if not x.median() else 1e-8)
    )
```

Impute missing values for categorical data on a per-fund basis using the mode as these values should hold stable.

Flatten the data to create a single row per fund with separate columns for each time point by adding a time suffix. This way we retain the temporal nature of the data but cluster funds as just one data point rather than 3.

```
pivoted_data = data_cleaned.pivot(index='Security ID', columns='Date', values=numeric_cols)

pivoted_data.columns = [f'{col[0]}_T{col[1].month}' for col in pivoted_data.columns]
```

Clustering using Agglomerative Clustering

```
agg_cluster = AgglomerativeClustering(n_clusters=6,  metric='euclidean', linkage='ward')
cluster_labels = agg_cluster.fit_predict(data_reduced)
```

Evaluate Clustering

```
sil_score = silhouette_score(data_reduced, cluster_labels)
print(f'Silhouette score : {sil_score}')
```

We see a silhouette score of around 0.56 with this method which is considered a strong clustering score (> 0.5), but leaves room for improvement.

Potential limitations include the actual clustering algorithm used (agglomerative) which seemed to produce the highest silhouette scores but can be sensitive to outliers. We could experiment further with other methods. The imputation of missing data per fund (especially numerical data) using the median can be reductive and mislead clustering algorithms. Potential improvements could be more advanced imputation techniques such as K-NN or iterative imputation.

Adding more advanced features will also help the clustering of the fund.