```python
#Importing all libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load datasets
customers = pd.read_csv(r'C:\Users\A.Rohith Venkatesh\Downloads\
Customers.csv')
transactions = pd.read_csv(r'C:\Users\A.Rohith Venkatesh\Downloads\
Transactions.csv')

# Data preprocessing
transactions_summary = transactions.groupby('CustomerID').agg({
    'TotalValue': 'sum',
    'Quantity': 'sum',
    'ProductID': lambda x: ' '.join(x.astype(str))
}).reset_index()

customer_profiles = pd.merge(customers, transactions_summary,
on='CustomerID', how='left')
customer_profiles.fillna(0, inplace=True)

from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import davies_bouldin_score
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
import seaborn as sns

# Ensure customer_profiles is defined
# Feature selection for clustering
clustering_data = customer_profiles.drop(columns=['CustomerID',
'CustomerName', 'ProductID'])

# Initialize and Standardize features
scaler = StandardScaler()  # Initialize the scaler
scaled_clustering_data = scaler.fit_transform(clustering_data)

# Determine optimal number of clusters
db_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    clusters = kmeans.fit_predict(scaled_clustering_data)
    db_score = davies_bouldin_score(scaled_clustering_data, clusters)
    db_scores.append(db_score)
```

```
# Visualize DB Index scores
plt.figure(figsize=(10, 6))
sns.lineplot(x=range(2, 11), y=db_scores, marker='o')
plt.title('Davies-Bouldin Index for Different Cluster Counts')
plt.xlabel('Number of Clusters')
plt.ylabel('DB Index')
plt.show()

# Final Clustering with Optimal K
optimal_k = db_scores.index(min(db_scores)) + 2
final_kmeans = KMeans(n_clusters=optimal_k, random_state=42)
customer_profiles['Cluster'] =
final_kmeans.fit_predict(scaled_clustering_data)

# Save results
customer_profiles.to_csv('Clustering_Results.csv', index=False)

# Visualize Clusters using PCA
pca = PCA(n_components=2)
reduced_data = pca.fit_transform(scaled_clustering_data)

plt.figure(figsize=(10, 6))
sns.scatterplot(x=reduced_data[:, 0], y=reduced_data[:, 1],
hue=customer_profiles['Cluster'], palette='viridis')
plt.title('Customer Clusters')
plt.show()

---------------------------------------------------------------------------
-----
ValueError                                Traceback (most recent call
last)
~\AppData\Local\Temp\ipykernel_9856\2095975723.py in ?()
     10 clustering_data =
customer_profiles.drop(columns=['CustomerID', 'CustomerName',
'ProductID'])
     11
     12 # Initialize and Standardize features
     13 scaler = StandardScaler()  # Initialize the scaler
---> 14 scaled_clustering_data = scaler.fit_transform(clustering_data)
     15
     16 # Determine optimal number of clusters
     17 db_scores = []

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\
utils\_set_output.py in ?(self, X, *args, **kwargs)
    317     @wraps(f)
    318     def wrapped(self, X, *args, **kwargs):
--> 319         data_to_wrap = f(self, X, *args, **kwargs)
    320         if isinstance(data_to_wrap, tuple):
    321             # only wrap the first output for cross
```

```
decomposition
    322             return_tuple = (

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\
base.py in ?(self, X, y, **fit_params)
    914                 )
    915
    916         if y is None:
    917             # fit method of arity 1 (unsupervised
transformation)
--> 918             return self.fit(X, **fit_params).transform(X)
    919         else:
    920             # fit method of arity 2 (supervised
transformation)
    921             return self.fit(X, y, **fit_params).transform(X)

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\
preprocessing\_data.py in ?(self, X, y, sample_weight)
    890             Fitted scaler.
    891         """
    892         # Reset internal state before fitting
    893         self._reset()
--> 894         return self.partial_fit(X, y, sample_weight)

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\
base.py in ?(estimator, *args, **kwargs)
   1385                 skip_parameter_validation=(
   1386                     prefer_skip_nested_validation or
global_skip_validation
   1387                 )
   1388             ):
-> 1389                 return fit_method(estimator, *args, **kwargs)

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\
preprocessing\_data.py in ?(self, X, y, sample_weight)
    926         self : object
    927             Fitted scaler.
    928         """
    929         first_call = not hasattr(self, "n_samples_seen_")
--> 930         X = validate_data(
    931             self,
    932             X,
    933             accept_sparse=("csr", "csc"),

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\
utils\validation.py in ?(_estimator, X, y, reset, validate_separately,
skip_check_array, **check_params)
   2940             out = y
   2941         else:
   2942             out = X, y
```

```
   2943     elif not no_val_X and no_val_y:
-> 2944         out = check_array(X, input_name="X", **check_params)
   2945     elif no_val_X and not no_val_y:
   2946         out = _check_y(y, **check_params)
   2947     else:

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\
utils\validation.py in ?(array, accept_sparse, accept_large_sparse,
dtype, order, copy, force_writeable, force_all_finite,
ensure_all_finite, ensure_non_negative, ensure_2d, allow_nd,
ensure_min_samples, ensure_min_features, estimator, input_name)
   1052                             )
   1053                         array = xp.astype(array, dtype,
copy=False)
   1054                 else:
   1055                     array = _asarray_with_order(array,
order=order, dtype=dtype, xp=xp)
-> 1056             except ComplexWarning as complex_warning:
   1057                 raise ValueError(
   1058                     "Complex data not supported\n{}\
n".format(array)
   1059                 ) from complex_warning

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\
utils\_array_api.py in ?(array, dtype, order, copy, xp, device)
    828         # Use NumPy API to support order
    829         if copy is True:
    830             array = numpy.array(array, order=order,
dtype=dtype)
    831         else:
--> 832             array = numpy.asarray(array, order=order,
dtype=dtype)
    833
    834         # At this point array is a NumPy ndarray. We convert
it to an array
    835         # container that is consistent with the input's
namespace.

~\AppData\Local\Programs\Python\Python313\Lib\site-packages\pandas\
core\generic.py in ?(self, dtype, copy)
   2149     def __array__(
   2150         self, dtype: npt.DTypeLike | None = None, copy: bool_t
| None = None
   2151     ) -> np.ndarray:
   2152         values = self._values
-> 2153         arr = np.asarray(values, dtype=dtype)
   2154         if (
   2155             astype_is_view(values.dtype, arr.dtype)
   2156             and using_copy_on_write()
```

```
ValueError: could not convert string to float: 'South America'
```