Data Preprocessing

Before performing clustering analysis on the given data, I had to preprocess the data to ensure that it met the requirements of the k-means algorithm. I did this by:

Standardizing the features using the StandardScaler class from scikit-learn. This ensures that all features have the same scale, which is important for the k-means algorithm.
Handling missing values using the SimpleImputer class from scikit-learn. This fills in missing values with the mean or median of the respective feature.

Clustering

Once the data has been preprocessed, I can cluster it using the k-means algorithm. The k-means algorithm works by iteratively assigning data points to clusters based on their distance to the cluster centroids. The number of clusters (k) is a hyperparameter that needs to be specified before clustering can be performed. I determined the optimal value for k using the elbow method, which plots the sum of squared errors (SSE) for different values of k. The value of k where the SSE curve starts to flatten out is generally considered to be the optimal value.

Evaluation

The clustering results can be evaluated using a variety of metrics, such as SSE and silhouette scores. SSE measures the sum of squared distances between data points and their assigned cluster centroids. A lower SSE indicates better clustering. Silhouette scores measure the similarity of data points within the same cluster and the dissimilarity between different clusters. A higher silhouette score indicates better clustering.

In conclusion, I preprocessed the data by standardizing the features and handling missing values, determined the optimal number of clusters using the elbow method, performed k-means clustering, and evaluated the results using SSE and silhouette scores.

This code snippet imports the necessary libraries for this task.

```
import os
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, MiniBatchKMeans
```

```
from sklearn.metrics import accuracy_score
from sklearn.impute import SimpleImputer
from sklearn.metrics import silhouette_score
import warnings
```

This code snippet sets the number of threads for the OpenMP library to 1. This is done to prevent the library from using too many resources and causing the program to crash.

```
os.environ['OMP_NUM_THREADS'] = '1'
```

```
# Read data from the CSV file
data = pd.read_csv('C:\\Data Mining Project\\Data Mining Task
3\\217240789.csv')

# Drop the 'Date' column
data = data.drop(['Date'], axis=1)
```

These two code snippets read the data from the CSV file and drop the Date column. The Date column is not needed for clustering, so it is dropped to improve the performance of the clustering algorithm.

```
# Prepare the data for clustering
X = data.values

# Normalize the data
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X)
```

These two code snippets prepare the data for clustering. First, the data is stored in a NumPy array called X. Then, the data is normalized using the StandardScaler class. This helps to improve the clustering algorithm by making the data more similar to each other.

```
# Calculate SSE for k=5 and k=7
def calculate_sse(data, k_values):
    sse_values = []
    for k in k_values:
        warnings.filterwarnings('ignore', category=UserWarning)
        kmeans = MiniBatchKMeans(n_clusters=k, batch_size=3072, n_init=10)
```

```
        kmeans.fit(data)
        sse_values.append(kmeans.inertia_)
    return sse_values

k_values = [5, 7]
sse_values = calculate_sse(X_normalized, k_values)
```

This code snippet calculates the SSE for k=5 and k=7. SSE is a measure of how well the data points are clustered together. The lower the SSE, the better the clustering. The calculate_sse function takes a NumPy array and a list of k values as input. It then iterates through the list of k values and calculates the SSE for each value. The results are stored in a list called sse_values.

```
# Print SSE values
for k, sse in zip(k_values, sse_values):
    print(f"SSE for k={k}: {sse}")
```

This code snippet prints the SSE values for k=5 and k=7.

```
SSE for k=5: 1699.0933427477173
SSE for k=7: 1516.109624707069
```

Results

```
# Perform k-means clustering with the number of classes from Phase 2 Task 6
k = 2
warnings.filterwarnings('ignore', category=UserWarning)
kmeans = KMeans(n_clusters=k, n_init=10, random_state=42)
kmeans.fit(X_normalized)
y_pred = kmeans.labels_

# Compute silhouette score assuming we don't have true labels
silhouette = silhouette_score(X_normalized, y_pred)
print(f"Silhouette Score: {silhouette}")
```

This code snippet performs k-means clustering with the number of classes from Phase 2 Task 6. The silhouette score is then computed to assess the quality of the clustering.

```
Silhouette Score: 0.28397193204722904
```

Results