In [ ]:
```
import pandas as pd

# Load the dataset
data = pd.read_csv('Healthcare Providers.csv')
data.head()
```

Out[ ]:

	index	National Provider Identifier	Last Name/Organization Name of the Provider	First Name of the Provider	Middle Initial of the Provider	Credentials of the Provider	Gende of th Provide
0	8774979	1891106191	UPADHYAYULA	SATYASREE	NaN	M.D.	
1	3354385	1346202256	JONES	WENDY	P	M.D.	
2	3001884	1306820956	DUROCHER	RICHARD	W	DPM	N
3	7594822	1770523540	FULLARD	JASPER	NaN	MD	N
4	746159	1073627758	PERROTTI	ANTHONY	E	DO	N

5 rows × 27 columns

Dropping Unnecessary Columns

In [ ]:
```
# Dropping unnecessary columns
data = data.drop(columns=['Middle Initial of the Provider', 'Street Address 1 of
data.head()
```

Out[ ]:

	index	National Provider Identifier	Last Name/Organization Name of the Provider	First Name of the Provider	Credentials of the Provider	Gender of the Provider	Entity Type of the Provide
0	8774979	1891106191	UPADHYAYULA	SATYASREE	M.D.	F	
1	3354385	1346202256	JONES	WENDY	M.D.	F	
2	3001884	1306820956	DUROCHER	RICHARD	DPM	M	
3	7594822	1770523540	FULLARD	JASPER	MD	M	
4	746159	1073627758	PERROTTI	ANTHONY	DO	M	

5 rows × 24 columns

Encoding Categorical Columns Categorical columns will be encoded using One-Hot Encoding if they have low cardinality or Frequency Encoding if they have high cardinality.

In [ ]:

```
# Identifying categorical columns
categorical_columns = data.select_dtypes(include=['object']).columns
encoded_data = data.copy()

# Applying One-Hot Encoding for low cardinality categorical columns
low_cardinality_cols = [col for col in categorical_columns if data[col].nunique(
encoded_data = pd.get_dummies(encoded_data, columns=low_cardinality_cols, drop_f

# Applying Frequency Encoding for high cardinality categorical columns
high_cardinality_cols = [col for col in categorical_columns if data[col].nunique
for col in high_cardinality_cols:
    freq_encoding = encoded_data[col].value_counts().to_dict()
    encoded_data[col] = encoded_data[col].map(freq_encoding)

encoded_data.head()
```

Out[ ]:

	index	National Provider Identifier	Last Name/Organization Name of the Provider	First Name of the Provider	Credentials of the Provider	Street Address 2 of the Provider	City of the Provider
0	8774979	1891106191	1	1.0	32757.0	2.0	500
1	3354385	1346202256	251	75.0	32757.0	NaN	209
2	3001884	1306820956	2	1030.0	1330.0	7.0	10
3	7594822	1770523540	1	2.0	32874.0	NaN	317
4	746159	1073627758	1	356.0	2478.0	1624.0	51

5 rows × 26 columns

Scaling We'll use StandardScaler to scale both numerical columns and the transformed categorical columns.

In [ ]:

```
from sklearn.preprocessing import StandardScaler

# Scaling the data
scaler = StandardScaler()
scaled_data = pd.DataFrame(scaler.fit_transform(encoded_data), columns=encoded_d
scaled_data.head()
```

Out[ ]:

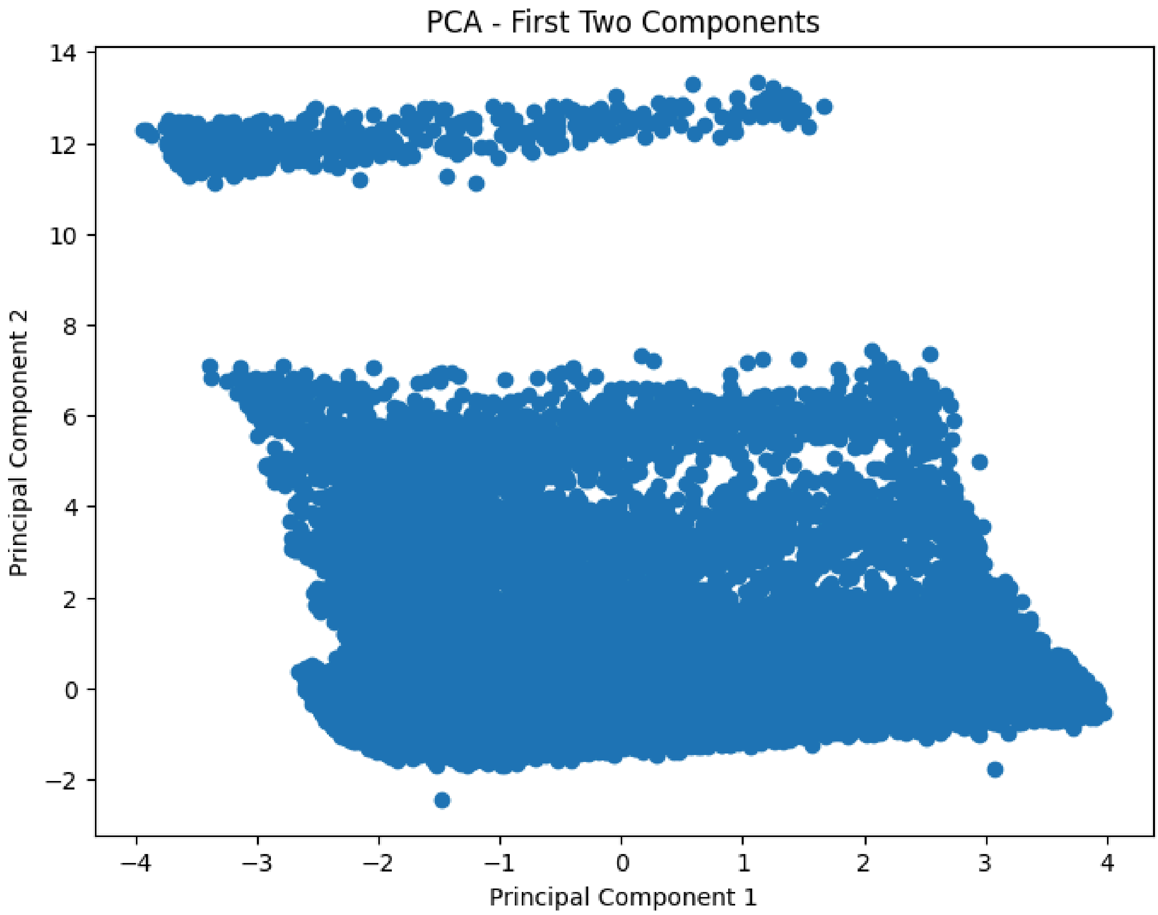
	index	National Provider Identifier	Last Name/Organization Name of the Provider	First Name of the Provider	Credentials of the Provider	Street Address 2 of the Provider	City o the Provide
0	1.361920	1.366960	-0.383401	-0.660442	0.638605	-0.548038	1.57168
1	-0.546996	-0.528945	2.854727	-0.549784	0.638605	NaN	0.18918
2	-0.671133	-0.665966	-0.370449	0.878292	-1.541230	-0.536514	-0.75624
3	0.946316	0.947412	-0.383401	-0.658946	0.646720	NaN	0.70227
4	-1.465509	-1.477323	-0.383401	-0.129586	-1.461602	3.190434	-0.56145

5 rows × 26 columns

In [ ]:

```
import pandas as pd
import numpy as np
from sklearn.impute import SimpleImputer
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt

# Assuming scaled_data is your DataFrame with scaled features

# Impute NaN values with the mean of each column
imputer = SimpleImputer(strategy='mean')
scaled_data_imputed = imputer.fit_transform(scaled_data)
```

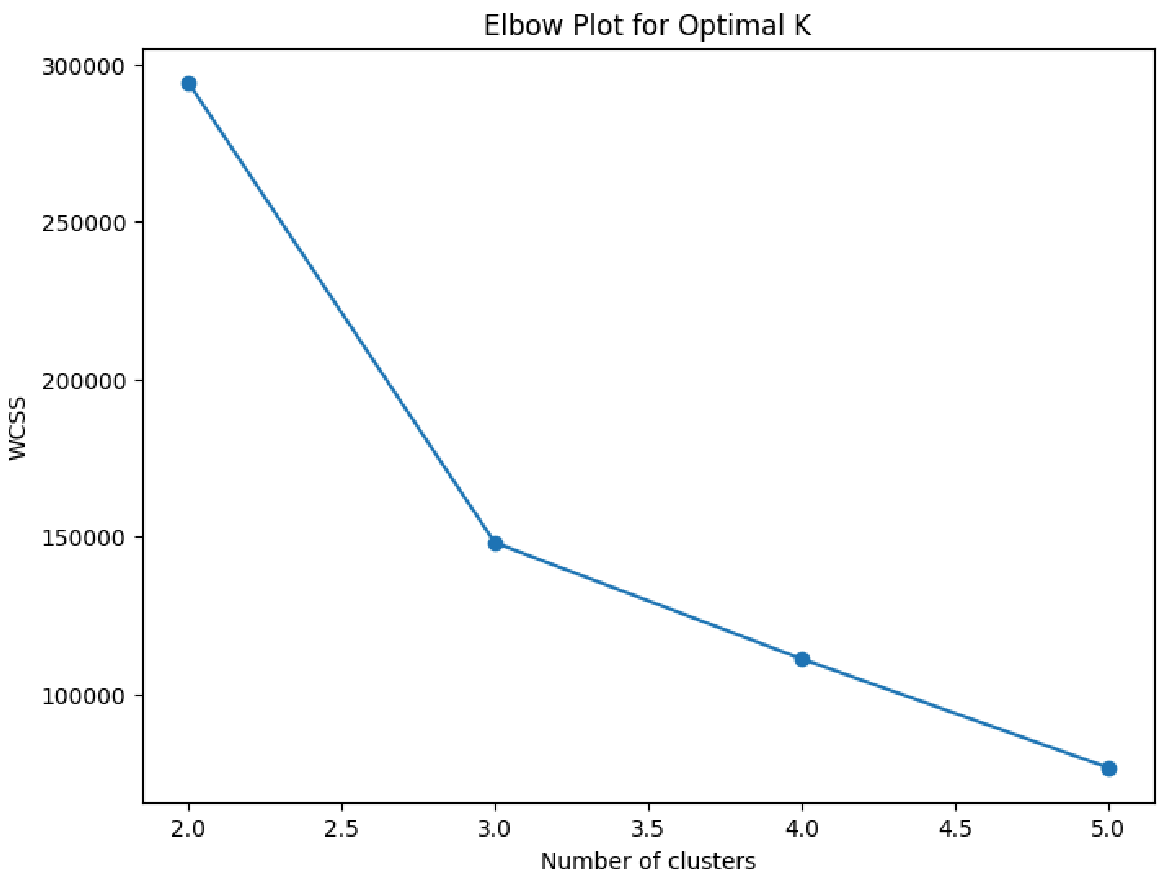
```
# Applying PCA
pca = PCA(n_components=2)
pca_components = pca.fit_transform(scaled_data_imputed)

# Creating a DataFrame for PCA components
pca_df = pd.DataFrame(data=pca_components, columns=['PC1', 'PC2'])

# Scatter plot of the first two PCA components
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'])
plt.title('PCA - First Two Components')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.show()
```



PCA

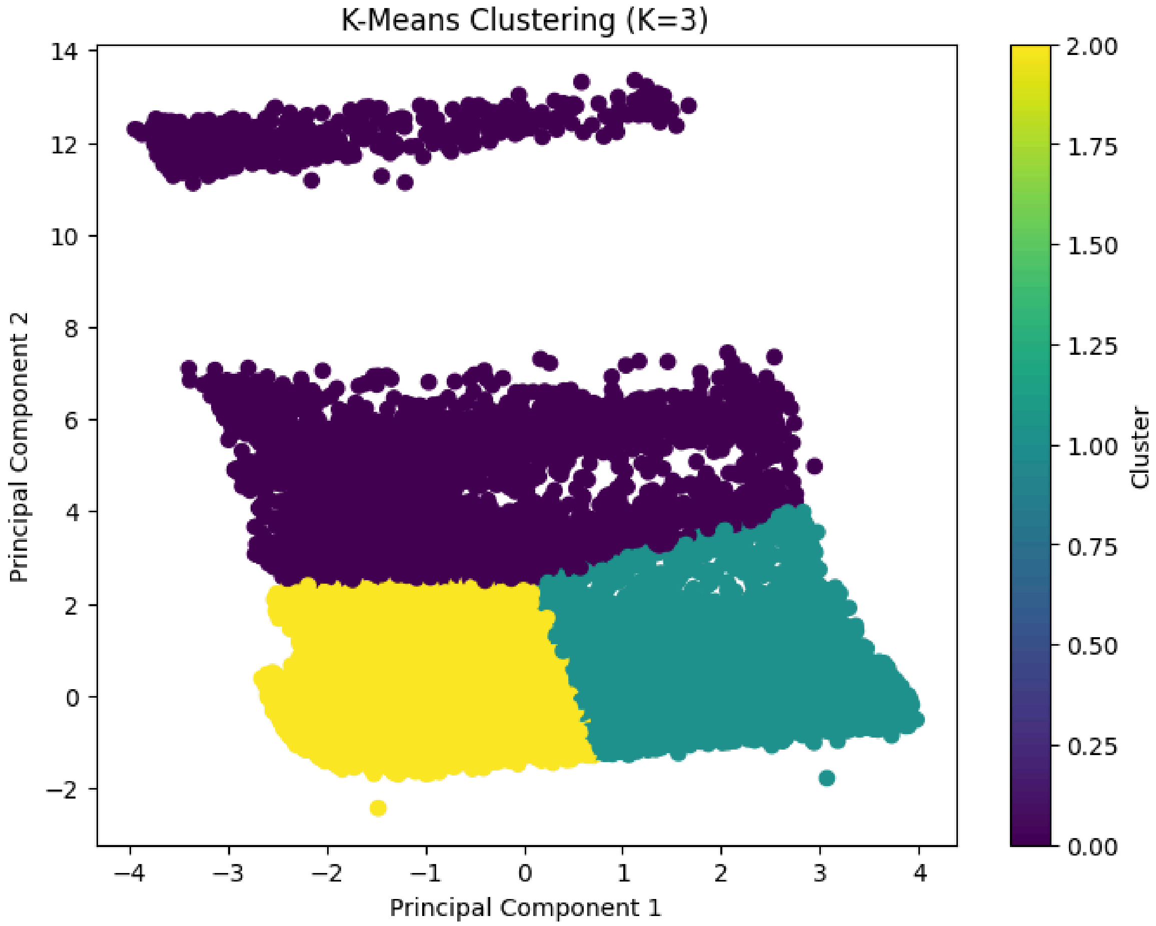
In [ ]:
```
from sklearn.cluster import KMeans

# Using the elbow method to find the optimal number of clusters
wcss = []
for i in range(2, 6):
    kmeans = KMeans(n_clusters=i, random_state=42)
    kmeans.fit(pca_df)
    wcss.append(kmeans.inertia_)

# Plotting the elbow plot
plt.figure(figsize=(8, 6))
plt.plot(range(2, 6), wcss, marker='o')
plt.title('Elbow Plot for Optimal K')
plt.xlabel('Number of clusters')
```

```
plt.ylabel('WCSS')
plt.show()
```



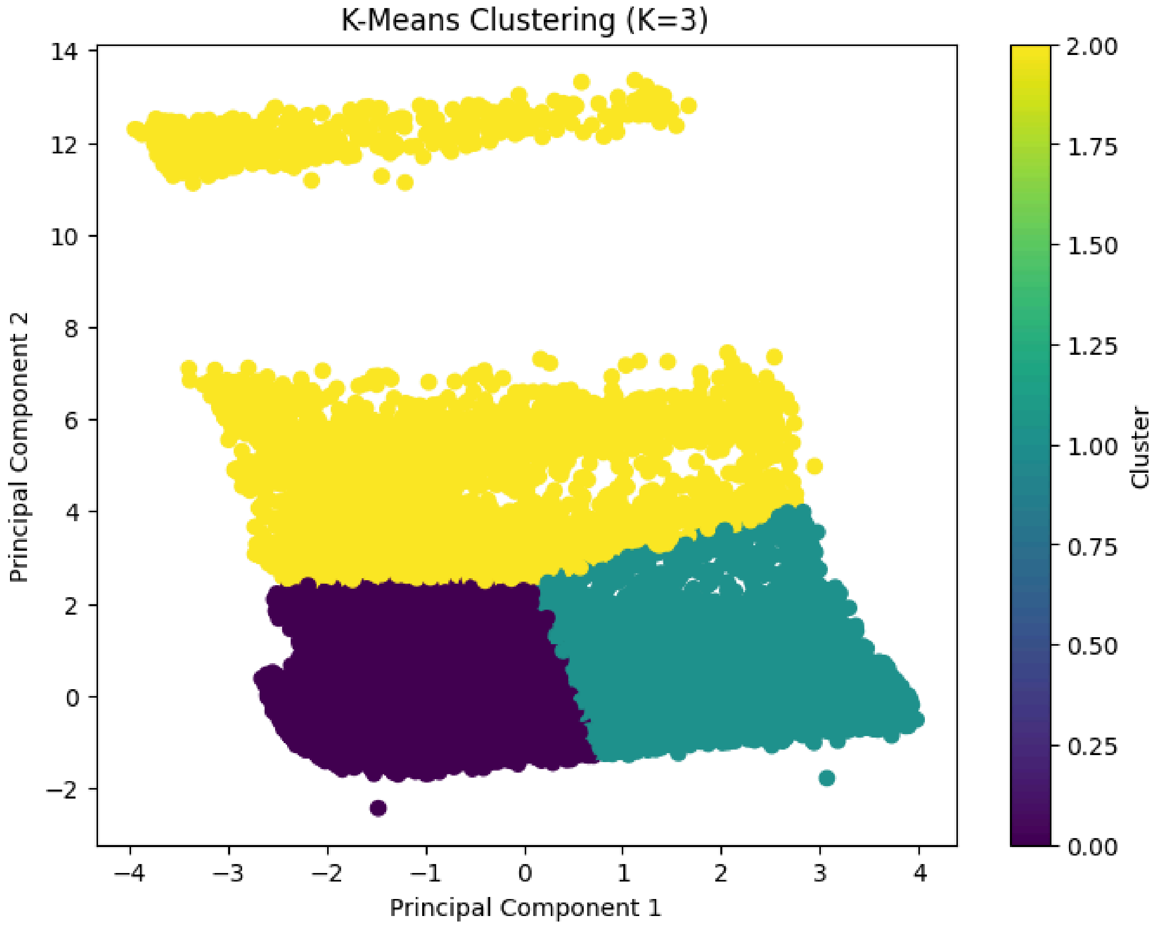
Clustering K-Means Clustering We'll apply K-Means Clustering on the PCA-transformed data and determine the optimal number of clusters using the elbow method.

In [ ]:
```
# Applying K-Means with the optimal number of clusters
optimal_k = 3  # Assuming 3 from elbow plot
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
pca_df['Cluster'] = kmeans.fit_predict(pca_df)

# Scatter plot of clusters
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['Cluster'], cmap='viridis')
plt.title(f'K-Means Clustering (K={optimal_k})')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```



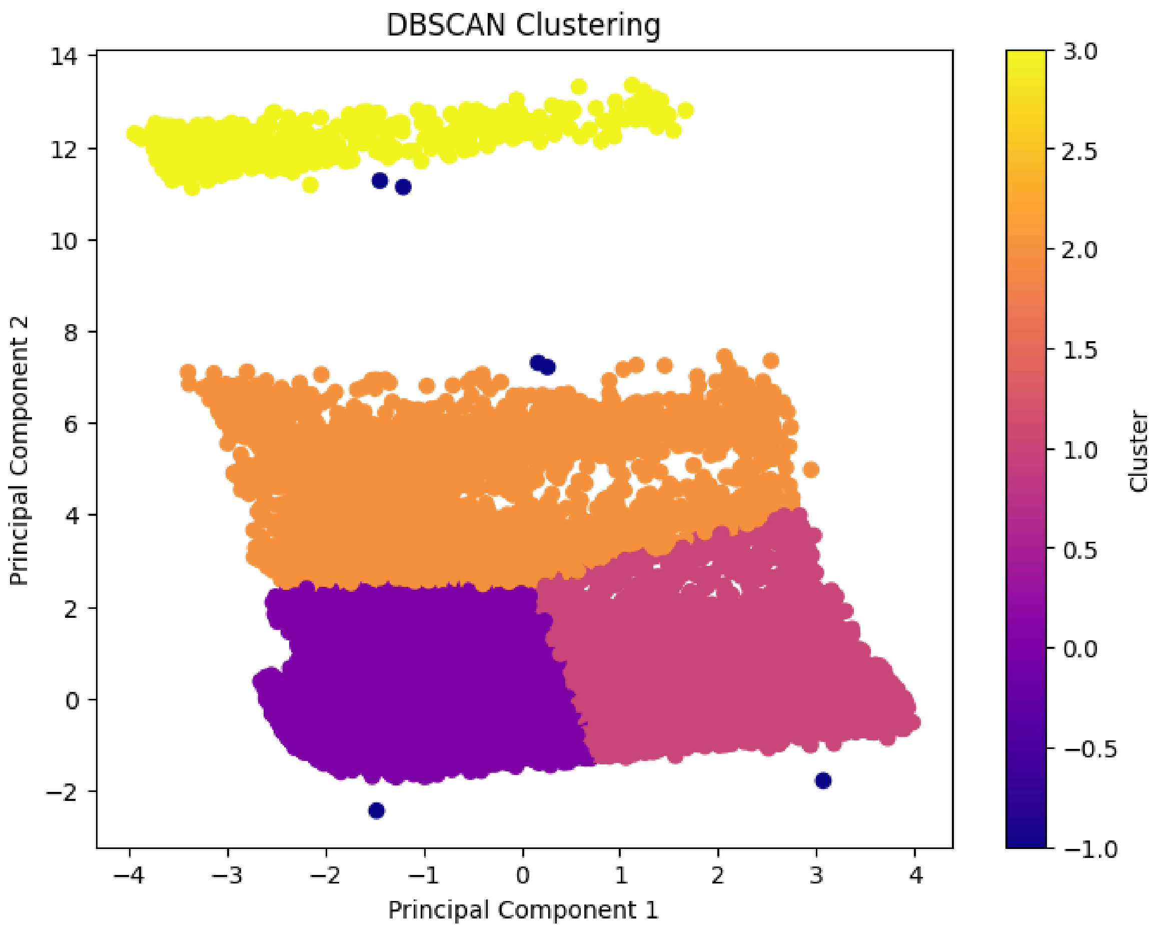
Based on the elbow plot, we'll choose the optimal K value and visualize the clusters.

In [ ]:
```
# Applying K-Means with the optimal number of clusters
optimal_k = 3  # Assuming 3 from elbow plot
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
pca_df['Cluster'] = kmeans.fit_predict(pca_df)

# Scatter plot of clusters
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['Cluster'], cmap='viridis')
plt.title(f'K-Means Clustering (K={optimal_k})')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```



DBSCAN Clustering We'll apply DBSCAN Clustering and visualize the clusters.

In [ ]:
```
from sklearn.cluster import DBSCAN

# Applying DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
pca_df['DBSCAN_Cluster'] = dbscan.fit_predict(pca_df)

# Scatter plot of DBSCAN clusters
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['DBSCAN_Cluster'], cmap='plas
plt.title('DBSCAN Clustering')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```


DBSCAN Clustering
Principal Component 2
Principal Component 1
Cluster
14
12
10
8
6
4
2
0
−2
−4
−3
−2
−1
0
1
2
3
4
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0