INFOSYS SPRINGBOARD

Building Anomaly Detection System using Python

(unsupervised approach)

Problem Statement:

In this project, we delve deep into the thriving sector of **Security** by analyzing a **Anomaly detection on Healthcare Dataset** from a USA-based Health Service Providers, available at the kaggle. This dataset documents all transactions between patients and service providers. Our primary objective is to amplify the efficiency of Healthcare System and avoid fraudulent transactions in **Healthcare system**. We aim to transform the data into a -centric dataset that will facilitate the Base for Anomaly Detection system of patient providing better service , ultimately enhancing security ,efficiency and patient service.

In [116…

```
#importing necessary libraries
#Loading dataset

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import statistics
from matplotlib.colors import LinearSegmentedColormap
from sklearn.ensemble import IsolationForest
from sklearn.preprocessing import StandardScaler
import category_encoders as ce
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
from sklearn.cluster import DBSCAN

hdata = pd.read_csv(r'C:\Users\tmbha\Downloads\ifosys_springboard\df_processed.csv')
```

In [117…

```
hdata.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 20 columns):
 #   Column                                                Non-Null Count   Dtype
---  ------                                                --------------   -----
 0   Credentials of the Provider                           100000 non-null  object
 1   Gender of the Provider                                100000 non-null  object
 2   Entity Type of the Provider                           100000 non-null  object
 3   City of the Provider                                  100000 non-null  object
 4   State Code of the Provider                            100000 non-null  object
 5   Country Code of the Provider                          100000 non-null  object
 6   ProviderType                                          100000 non-null  object
 7   Medicare Participation Indicator                      100000 non-null  object
 8   Place of Service                                      100000 non-null  object
 9   HCPCS Description                                     100000 non-null  object
 10  HCPCS Drug Indicator                                  100000 non-null  object
 11  Number of Services                                    100000 non-null  float64
 12  Number of Medicare Beneficiaries                      100000 non-null  int64
 13  Number of Distinct Medicare Beneficiary/Per Day Services  100000 non-null  int64
 14  Average Medicare Allowed Amount                       100000 non-null  float64
 15  Average Submitted Charge Amount                       100000 non-null  float64
 16  Average Medicare Payment Amount                       100000 non-null  float64
 17  Average Medicare Standardized Amount                  100000 non-null  float64
 18  Name                                                  100000 non-null  object
 19  Full Address                                          100000 non-null  object
dtypes: float64(5), int64(2), object(13)
memory usage: 15.3+ MB
```

In [118…

```
hdata.describe(include='object')
```

Out[118…

	Credentials of the Provider	Gender of the Provider	Entity Type of the Provider	City of the Provider	State Code of the Provider	Country Code of the Provider	ProviderType	Medicare Participation Indicator	Place of Service	HCPCS Description	HCPCS Drug Indicator	Nam
count	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	100000	1000
unique	1539	2	2	5846	58	4	90	2	2	2455	2	874
top	MD	M	I	NEW YORK	CA	US	Diagnostic Radiology	Y	O	Established patient office or other outpatient...	N	WALGREE
freq	73827	70895	95746	1061	7775	99994	12537	99969	61616	4578	93802	3

Step 5 | Feature Engineering

In [119…

```
#generating new feature name Diff_submitted_allowed column in our dataset

hdata['Diff_submitted_allowed'] = hdata['Average Submitted Charge Amount'] - hdata['Average Medicare Allowed Amount']

# Create a binary column indicating whether the customer is from the UK or not
hdata['Is_US'] = hdata['Country Code of the Provider'].apply(lambda x: 1 if x == 'US' else 0)

# Create a binary column indicating Gender of the Provider
hdata['Gender'] = hdata['Gender of the Provider'].apply(lambda x: 1 if x == 'M' else 0)

# Create a binary column indicating Entity Type of the Provider
hdata['Entity'] = hdata['Entity Type of the Provider'].apply(lambda x: 1 if x == 'I' else 0)

# Create a binary column indicating Medicare Participation Indicator
hdata['Medicare_Participation_Indicator'] = hdata['Medicare Participation Indicator'].apply(lambda x: 1 if x == 'Y' else 0)

# Create a binary column indicating Place of Service
hdata['Place_of_Service'] = hdata['Place of Service'].apply(lambda x: 1 if x == 'O' else 0)

# Create a binary column indicating Place of Service
hdata['HCPCS_Drug_Indicator'] = hdata['HCPCS Drug Indicator'].apply(lambda x: 1 if x == 'N' else 0)

DropCols = ['Country Code of the Provider','Gender of the Provider','Entity Type of the Provider','Medicare Participation Indic
           'Place of Service','HCPCS Drug Indicator']
hdata = hdata.drop(DropCols, axis = 1)
```

In [120…

```
hdata.head().T
```

Out[120…

	0	1	2	3	4
Credentials of the Provider	MD	MD	DPM	MD	DO
City of the Provider	SAINT LOUIS	FAYETTEVILLE	NORTH HAVEN	KANSAS CITY	JUPITER
State Code of the Provider	MO	NC	CT	MO	FL
ProviderType	Internal Medicine	Obstetrics & Gynecology	Podiatry	Internal Medicine	Internal Medicine
HCPCS Description	Initial hospital inpatient care, typically 70 ...	Screening mammography, bilateral (2-view study...	Established patient home visit, typically 25 m...	Urinalysis, manual test	Injection beneath the skin or into muscle for ...
Number of Services	27.0	175.0	32.0	20.0	33.0
Number of Medicare Beneficiaries	24	175	13	18	24
Number of Distinct Medicare Beneficiary/Per Day Services	27	175	32	20	31
Average Medicare Allowed Amount	200.587778	123.73	90.65	3.5	26.52
Average Submitted Charge Amount	305.211111	548.8	155.0	5.0	40.0
Average Medicare Payment Amount	157.262222	118.83	64.439688	3.43	19.539394
Average Medicare Standardized Amount	160.908889	135.315257	60.595937	3.43	19.057576
Name	UPADHYAYULA SATYASREE	JONES WENDY P	DUROCHER RICHARD W	FULLARD JASPER	PERROTTI ANTHONY E
Full Address	1402 S GRAND BLVD FDT 14TH FLOOR	2950 VILLAGE DR	20 WASHINGTON AVE STE 212	5746 N BROADWAY ST	875 MILITARY TRL SUITE 200
Diff_submitted_allowed	104.623333	425.07	64.35	1.5	13.48
Is_US	1	1	1	1	1
Gender	0	0	1	1	1
Entity	1	1	1	1	1
Medicare_Participation_Indicator	1	1	1	1	1
Place_of_Service	0	1	1	1	1
HCPCS_Drug_Indicator	1	1	1	1	1

Inferences from the Feature Engineering :

- Created a new column name **Diff_submitted_allowed** which shows the difference between average allowed medicare amount and average submitted charge amount .
- There is existing column named Country Code of the Provider which had 4 unique values but the entire data is dominated by US covering 100% proportion of entire column henced created a new feature column **Is_US** and converted it in numeric data type(Boolean) and dropped existing column
- Converted categorical columns which have only two unique values to **binary** columns such as **Gender, Entity type, Medicare_Participation_Indicator,Place_of_Service, HCPCS_Drug_Indicator** to 0 and 1s.

Step 6 | Frequency Encoding

In [121…

```
#encoding categorical variable with frequency encoding
def frequency_encode_columns(df, columns):
    for col in columns:
        frequency_map = df[col].value_counts(normalize=True)
        df[col + '_FrequencyEncoded'] = df[col].map(frequency_map)
    return df

#creating list of categorical columns to encode
columns_to_encode = ['Credentials of the Provider','City of the Provider','State Code of the Provider',
                     'ProviderType','HCPCS Description']

hdata_encoded = frequency_encode_columns(hdata, columns_to_encode)

#Droping the orignal columns for maintaining sigle entries
hdata_encoded = hdata_encoded.drop(columns_to_encode, axis = 1)
```

In [122…

```
hdata_encoded.sample(5).T
```

Out[122...

	4617	39920	96318	70736	88686
Number of Services	160.0	121.0	223.0	21.0	11.0
Number of Medicare Beneficiaries	103	121	207	20	11
Number of Distinct Medicare Beneficiary/Per Day Services	160	121	216	21	11
Average Medicare Allowed Amount	88.41925	37.66	8.39	238.6	163.337273
Average Submitted Charge Amount	494.975	45.0	52.825112	597.0	563.0
Average Medicare Payment Amount	68.184875	28.208595	6.480583	183.950952	128.06
Average Medicare Standardized Amount	70.652125	24.031157	6.697892	192.440952	138.838182
Name	KIM JOSEPH S	NICKLES STEVEN L	KELLY GENE	FORESMAN WILLIAM	DEKKINGA JACK A
Full Address	13700 ST FRANCIS BLVD SUITE 103	581 N FRANKLIN TPKE	2901 N 4TH ST	192 GENESEE ST	2660 44TH ST SW STE 100
Diff_submitted_allowed	406.55575	7.34	44.435112	358.4	399.662727
Is_US	1	1	1	1	1
Gender	1	1	1	1	1
Entity	1	1	1	1	1
Medicare_Participation_Indicator	1	1	1	1	1
Place_of_Service	1	1	0	1	1
HCPCS_Drug_Indicator	1	1	1	1	1
Credentials of the Provider_FrequencyEncoded	0.73827	0.06176	0.73827	0.73827	0.73827
City of the Provider_FrequencyEncoded	0.00043	0.00008	0.00104	0.00082	0.00027
State Code of the Provider_FrequencyEncoded	0.02792	0.03333	0.06861	0.06361	0.03516
ProviderType_FrequencyEncoded	0.03098	0.0976	0.02377	0.01794	0.02022
HCPCS Description_FrequencyEncoded	0.00128	0.00018	0.0063	0.00024	0.00014

Inferences from the Feature Encoding :

- Apllied frequency encoding technique to encode high cardinal categorical columns such as **'Credentials of the Provider','City of the Provider','State Code of the Provider','ProviderType','HCPCS Description'** which has more than 50 unique values and some has 1000s of uniquevalues .

Step 7 | Feature Scaling

In [123...

```
# Initialize the StandardScaler
scaler = StandardScaler()

# List of columns that don't need to be scaled
columns_to_exclude = ['Name','Full Address']

# List of columns that need to be scaled
columns_to_scale = hdata_encoded.columns.difference(columns_to_exclude)

# Copy the cleaned dataset
hdata_scaled = hdata_encoded.copy()

# Applying the scaler to the necessary columns in the dataset
hdata_scaled[columns_to_scale] = scaler.fit_transform(hdata_scaled[columns_to_scale])

# Dropping identifiers
DropCols = ['Name','Full Address']
hdata_scaled = hdata_scaled.drop(DropCols, axis = 1)

# Display the first few rows of the scaled data
hdata_scaled.head().T
```

Out[123...

	0	1	2	3	4
Number of Services	-0.085301	-0.025939	-0.083296	-0.088109	-0.082895
Number of Medicare Beneficiaries	-0.059308	0.076775	-0.069222	-0.064716	-0.059308
Number of Distinct Medicare Beneficiary/Per Day Services	-0.070183	0.020049	-0.067135	-0.074451	-0.067744
Average Medicare Allowed Amount	0.385450	0.086673	-0.041922	-0.380709	-0.291221
Average Submitted Charge Amount	-0.046433	0.182805	-0.187794	-0.328957	-0.296019
Average Medicare Payment Amount	0.400082	0.207649	-0.064687	-0.370166	-0.289505
Average Medicare Standardized Amount	0.414299	0.286359	-0.087154	-0.372921	-0.294800
Diff_submitted_allowed	-0.166975	0.193356	-0.212261	-0.282934	-0.269463
Is_US	0.007746	0.007746	0.007746	0.007746	0.007746
Gender	-1.560716	-1.560716	0.640731	0.640731	0.640731
Entity	0.210784	0.210784	0.210784	0.210784	0.210784
Medicare_Participation_Indicator	0.017610	0.017610	0.017610	0.017610	0.017610
Place_of_Service	-1.266985	0.789275	0.789275	0.789275	0.789275
HCPCS_Drug_Indicator	0.257051	0.257051	0.257051	0.257051	0.257051
Credentials of the Provider_FrequencyEncoded	0.594983	0.594983	-1.684316	0.594983	-1.549260
City of the Provider_FrequencyEncoded	1.571686	0.189180	-0.756245	0.702275	-0.561459
State Code of the Provider_FrequencyEncoded	-0.737342	-0.004973	-0.989093	-0.737342	1.494517
ProviderType_FrequencyEncoded	1.336743	-0.940500	-0.720441	1.336743	1.336743
HCPCS Description_FrequencyEncoded	0.389268	-0.450300	-0.608815	-0.277448	-0.060785

Inferences from the Feature Scaling :

- Apllied **StandardScalar** to scale the encoded features.
- Also Dropped the identifiers from thedataset such as **'Name','Full Address'**.

Step 8 | Dimensionality Reduction

Dimensionality Reduction Method PCA

In this step, we are considering the application of dimensionality reduction techniques to simplify our data while retaining the essential information. Among various methods such as KernelPCA, ICA, ISOMAP, TSNE, and UMAP, I am starting with **PCA (Principal Component Analysis)**. Here's why:

PCA is an excellent starting point because it works well in capturing linear relationships in the data, which is particularly relevant given the multicollinearity we identified in our dataset. It allows us to reduce the number of features in our dataset while still retaining a significant amount of the information, thus making our clustering analysis potentially more accurate and interpretable. Moreover, it is computationally efficient, which means it won't significantly increase the processing time.

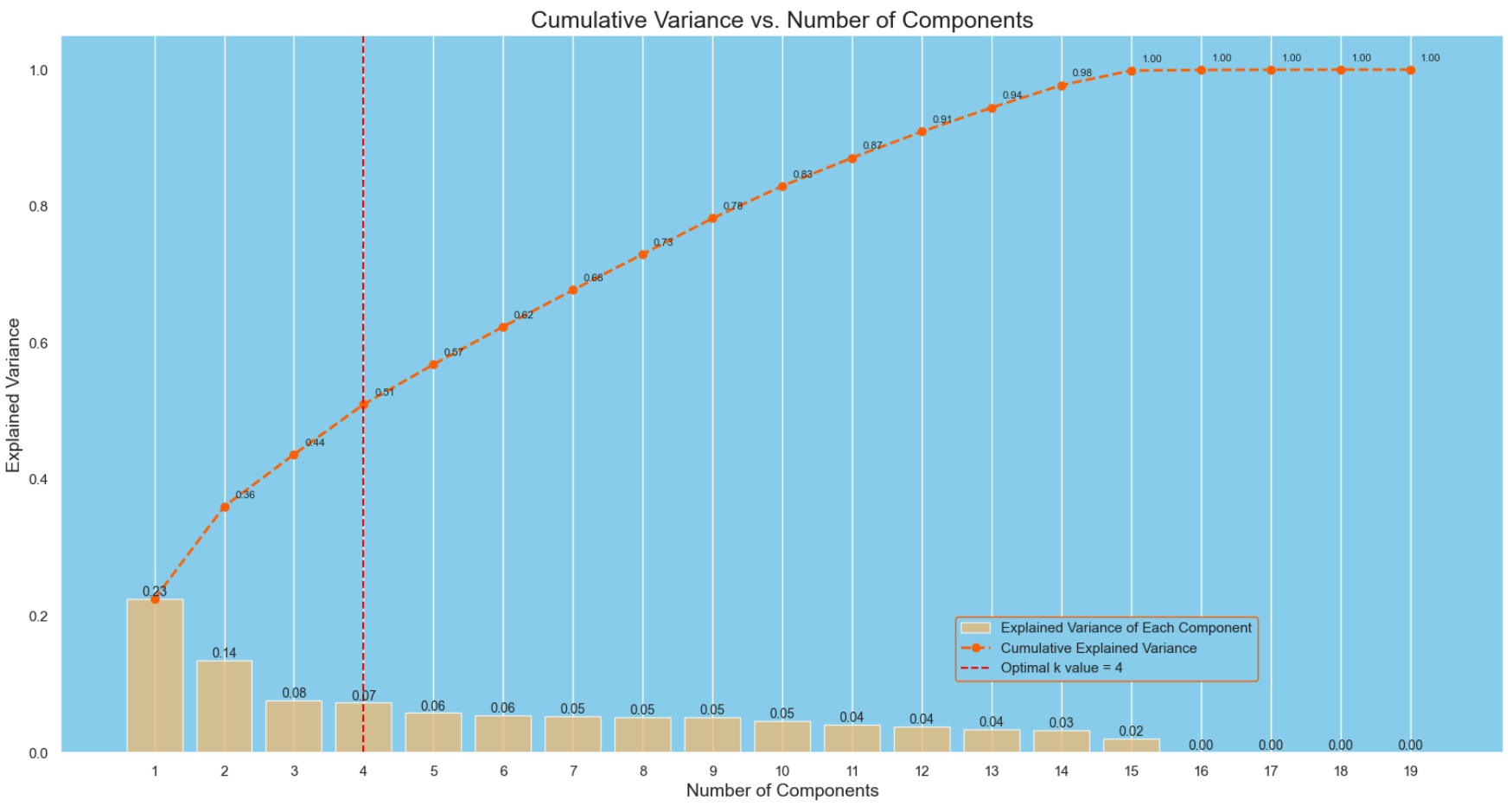
However, it's essential to note that we are keeping our options open. After applying PCA, if we find that the first few components do not capture a significant amount of variance, indicating a loss of vital information, we might consider exploring other non-linear methods. These methods can potentially provide a more nuanced approach to dimensionality reduction, capturing complex patterns that PCA might miss, albeit at the cost of increased computational time and complexity.

Methodology

We will apply PCA on all the available components and plot the cumulative variance explained by them. This process will allow me to visualize how much variance each additional principal component can explain, thereby helping me to pinpoint the optimal number of components to retain for the analysis:

In [124…

```
# Apply PCA
pca = PCA().fit(hdata_scaled)

# Calculate the Cumulative Sum of the Explained Variance
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_explained_variance = np.cumsum(explained_variance_ratio)

# Set the optimal k value (based on our analysis, we can choose 6)
optimal_k = 4

# Set seaborn plot style
sns.set(rc={'axes.facecolor': 'skyblue'}, style='darkgrid')

# Plot the cumulative explained variance against the number of components
plt.figure(figsize=(20, 10))

# Bar chart for the explained variance of each component
barplot = sns.barplot(x=list(range(1, len(cumulative_explained_variance) + 1)),
                      y=explained_variance_ratio,
                      color='#fcc36d',
                      alpha=0.8)

# Line plot for the cumulative explained variance
lineplot, = plt.plot(range(0, len(cumulative_explained_variance)), cumulative_explained_variance,
                     marker='o', linestyle='--', color='#ff6200', linewidth=2)

# Plot optimal k value line
optimal_k_line = plt.axvline(optimal_k - 1, color='red', linestyle='--', label=f'Optimal k value = {optimal_k}')

# Set labels and title
plt.xlabel('Number of Components', fontsize=14)
plt.ylabel('Explained Variance', fontsize=14)
plt.title('Cumulative Variance vs. Number of Components', fontsize=18)

# Customize ticks and legend
plt.xticks(range(0, len(cumulative_explained_variance)))
plt.legend(handles=[barplot.patches[0], lineplot, optimal_k_line],
           labels=['Explained Variance of Each Component', 'Cumulative Explained Variance', f'Optimal k value = {optimal
           loc=(0.62, 0.1),
           frameon=True,
           framealpha=1.0,
           edgecolor='#ff6200')

# Display the variance values for both graphs on the plots
x_offset = 0.3
y_offset = 0.01
for i, (ev_ratio, cum_ev_ratio) in enumerate(zip(explained_variance_ratio, cumulative_explained_variance)):
    plt.text(i, ev_ratio, f"{ev_ratio:.2f}", ha="center", va="bottom", fontsize=10)
    if i > 0:
        plt.text(i + x_offset, cum_ev_ratio + y_offset, f"{cum_ev_ratio:.2f}", ha="center", va="bottom", fontsize=8)

plt.grid(axis='both')
plt.show()
```



Conclusion

The plot and the cumulative explained variance values indicate how much of the total variance in the dataset is captured by each principal component, as well as the cumulative variance explained by the first n components.

Here, we can observe that:

- The first component explains approximately 23% of the variance.
- The first two components together explain about 36% of the variance.
- The first three components explain approximately 44% of the variance, and so on.

To choose the optimal number of components, we generally look for a point where adding another component doesn't significantly increase the cumulative explained variance, often referred to as the "**elbow point**" in the curve.

From the plot, we can see that the increase in cumulative variance starts to slow down after the **4th component** (which **captures about 51% of the total variance**).

Considering the context of Anomaly Detection, we want to retain a sufficient amount of information to identify distinct patient groups effectively. Therefore, retaining **the first 2 components** might be a balanced choice, as they together explain a substantial portion of the total variance while reducing the dimensionality of the dataset.

In [125…

```
# Creating a PCA object with 2 components
pca = PCA(n_components=2)

# Fitting and transforming the original data to the new PCA dataframe
data_pca = pca.fit_transform(hdata_scaled)

# Creating a new dataframe from the PCA dataframe, with columns labeled PC1, PC2, etc.
data_pca = pd.DataFrame(data_pca, columns=['PC'+str(i+1) for i in range(pca.n_components_)])

# Displaying the resulting dataframe based on the PCs
data_pca.head(10)
```

Out[125…

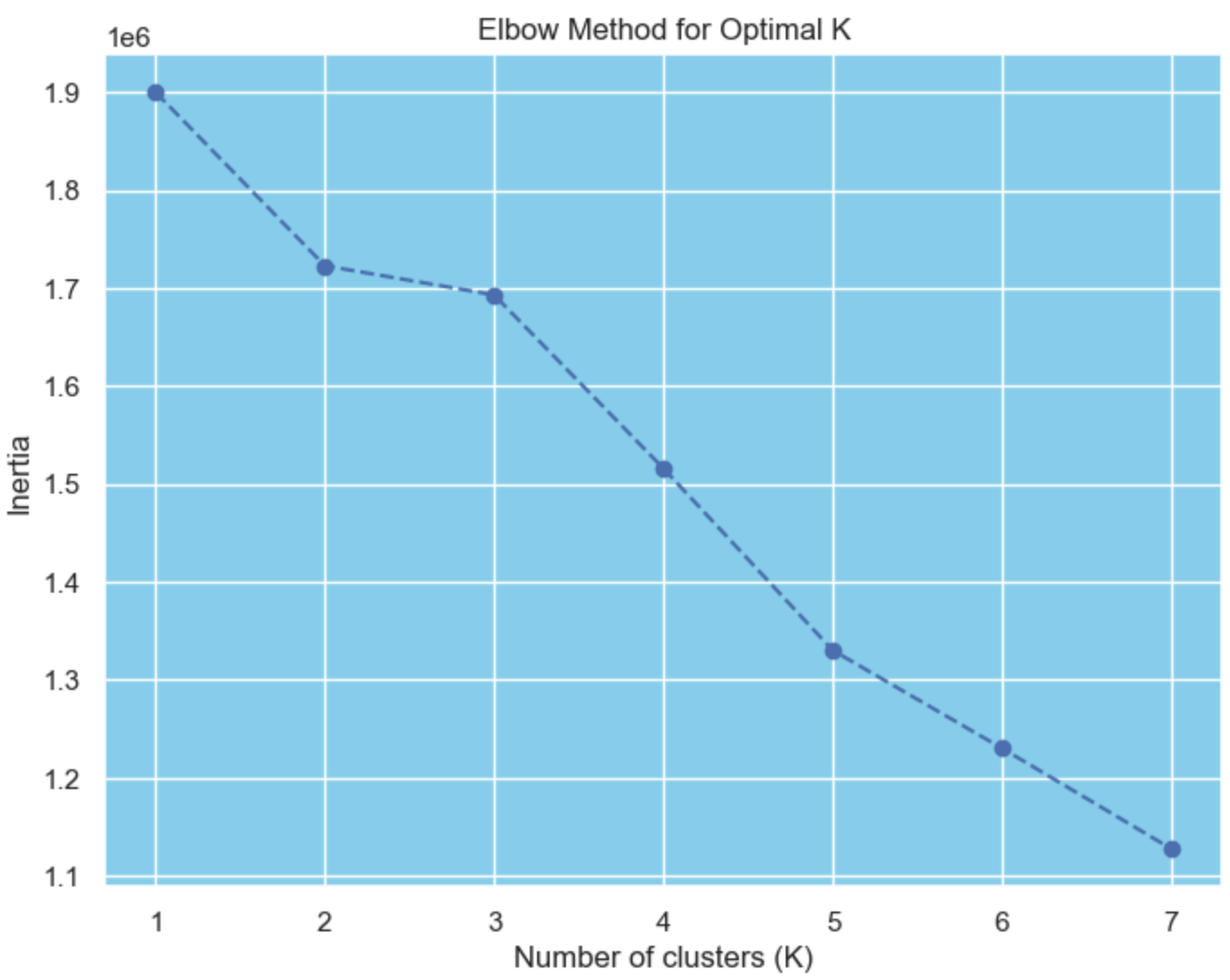
	PC1	PC2
0	0.435022	-0.288328
1	0.392231	0.052477
2	-0.323010	-0.134462
3	-0.868838	-0.183356
4	-0.824890	-0.188083
5	-0.734071	-0.040215
6	-0.470852	-0.299845
7	-0.688050	-0.095425
8	-0.736538	-0.132574
9	-0.369886	0.406647

Step 9 | Cluster Visualization

K-means Clustering

In [126…

```
# Initialize a list to store the values of inertia (sum of squared distances)
inertia = []

# Iterate over a range of possible number of clusters
for k in range(1, 8):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(hdata_scaled)
    inertia.append(kmeans.inertia_)

# Plotting the Elbow curve
plt.figure(figsize=(8, 6))
plt.plot(range(1, 8), inertia, marker='o', linestyle='--')
plt.xlabel('Number of clusters (K)')
plt.ylabel('Inertia')
plt.title('Elbow Method for Optimal K')
plt.show()
```

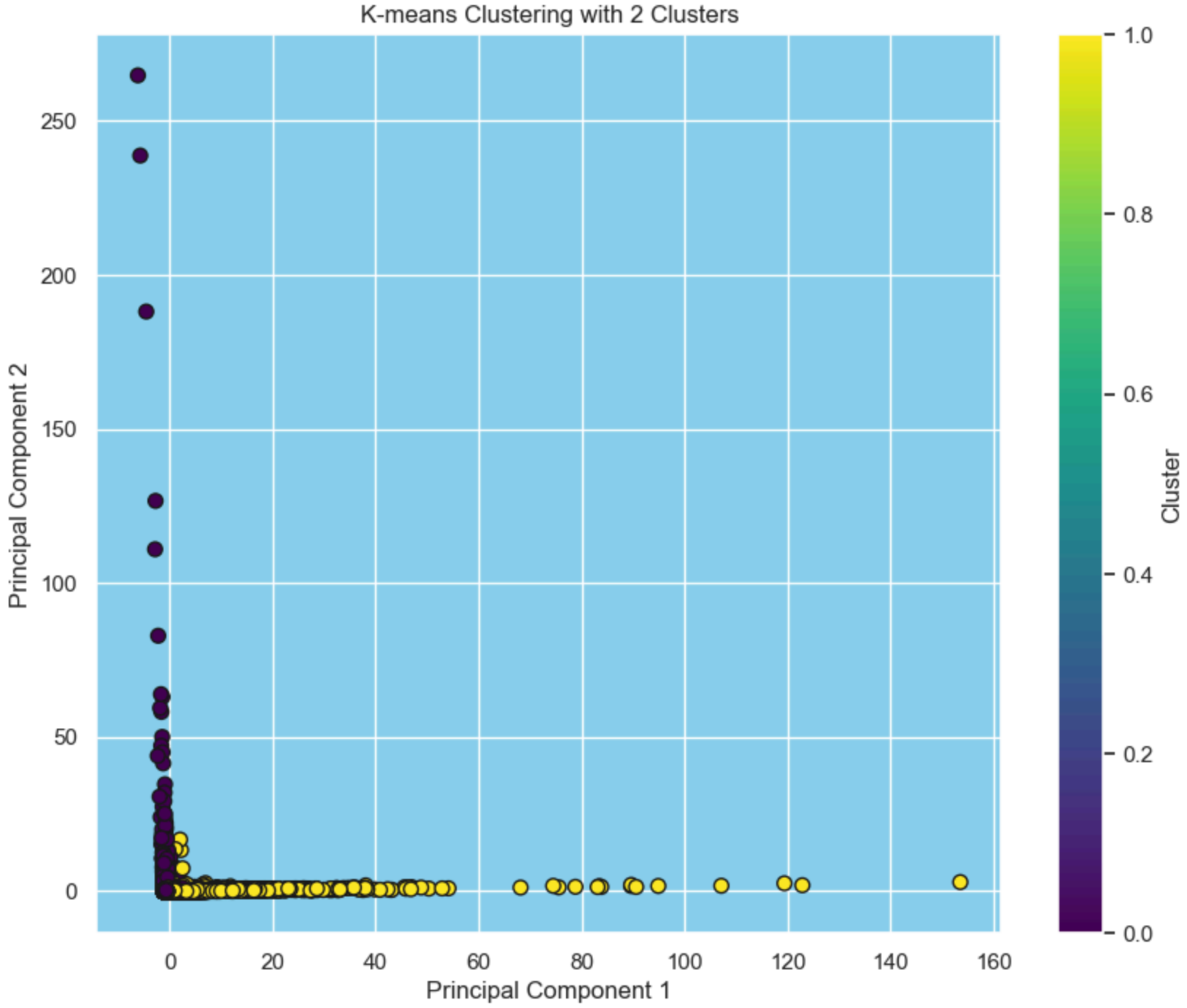


Inferences from the Plot :

- The Elbow method plot shows the no of clusters that can be formed with data .
- by observing above graph 2 should be the optimal no of cluster

In [127…
```python
# Initialize K-means with 2 clusters
kmeans = KMeans(n_clusters=2, random_state=0)
kmeans.fit(hdata_scaled)

# Get cluster labels
cluster_labels = kmeans.labels_

# Assign clusters back to the original DataFrame
hdata_scaled['Cluster'] = cluster_labels
```

In [128…
```python
# Create a DataFrame with the principal components and cluster labels
pca_df = pd.DataFrame(data=data_pca, columns=['PC1', 'PC2'])
pca_df['Cluster'] = cluster_labels

# Plot clusters
plt.figure(figsize=(10, 8))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['Cluster'], cmap='viridis', edgecolor='k', s=50)
plt.title('K-means Clustering with 2 Clusters')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

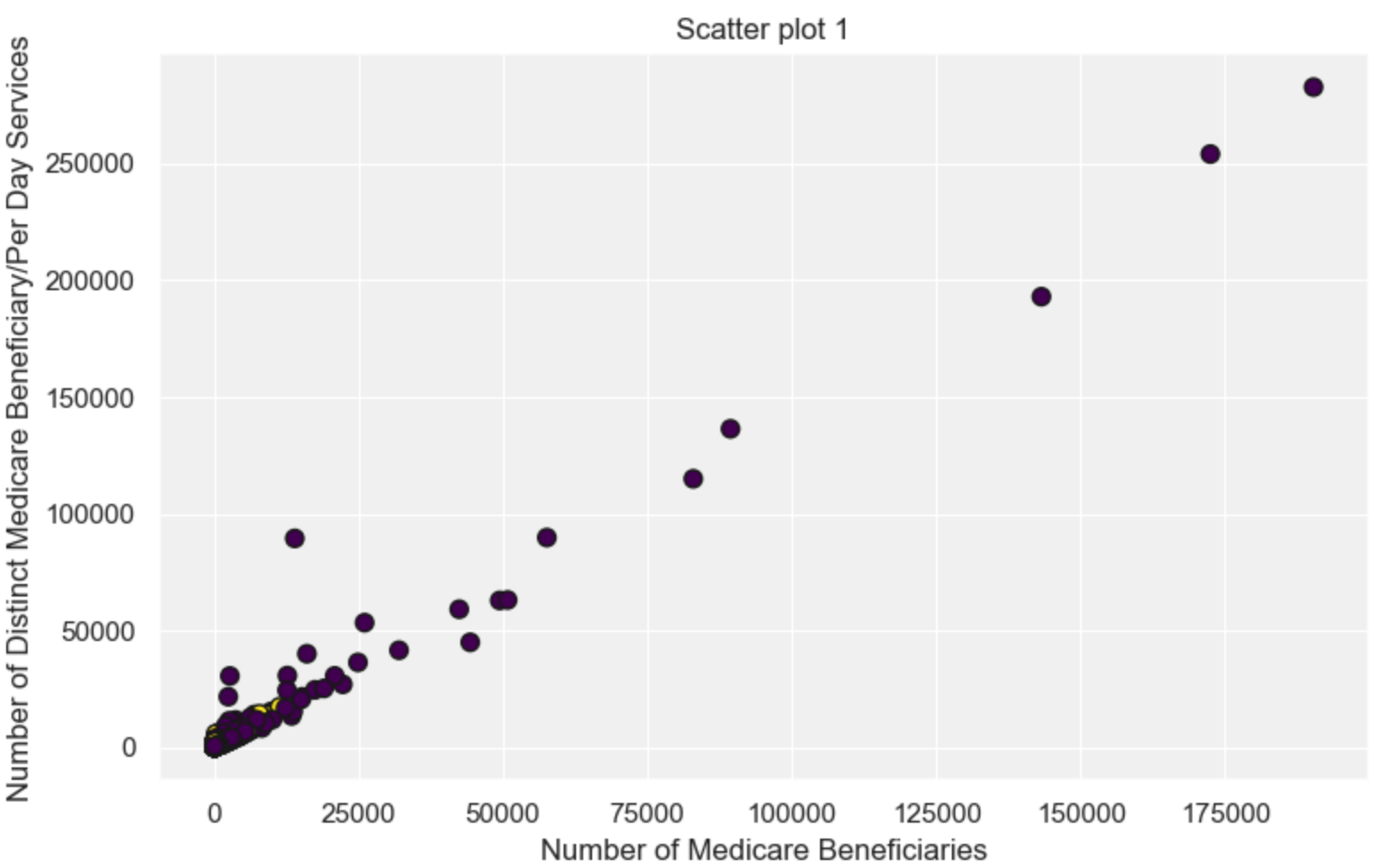


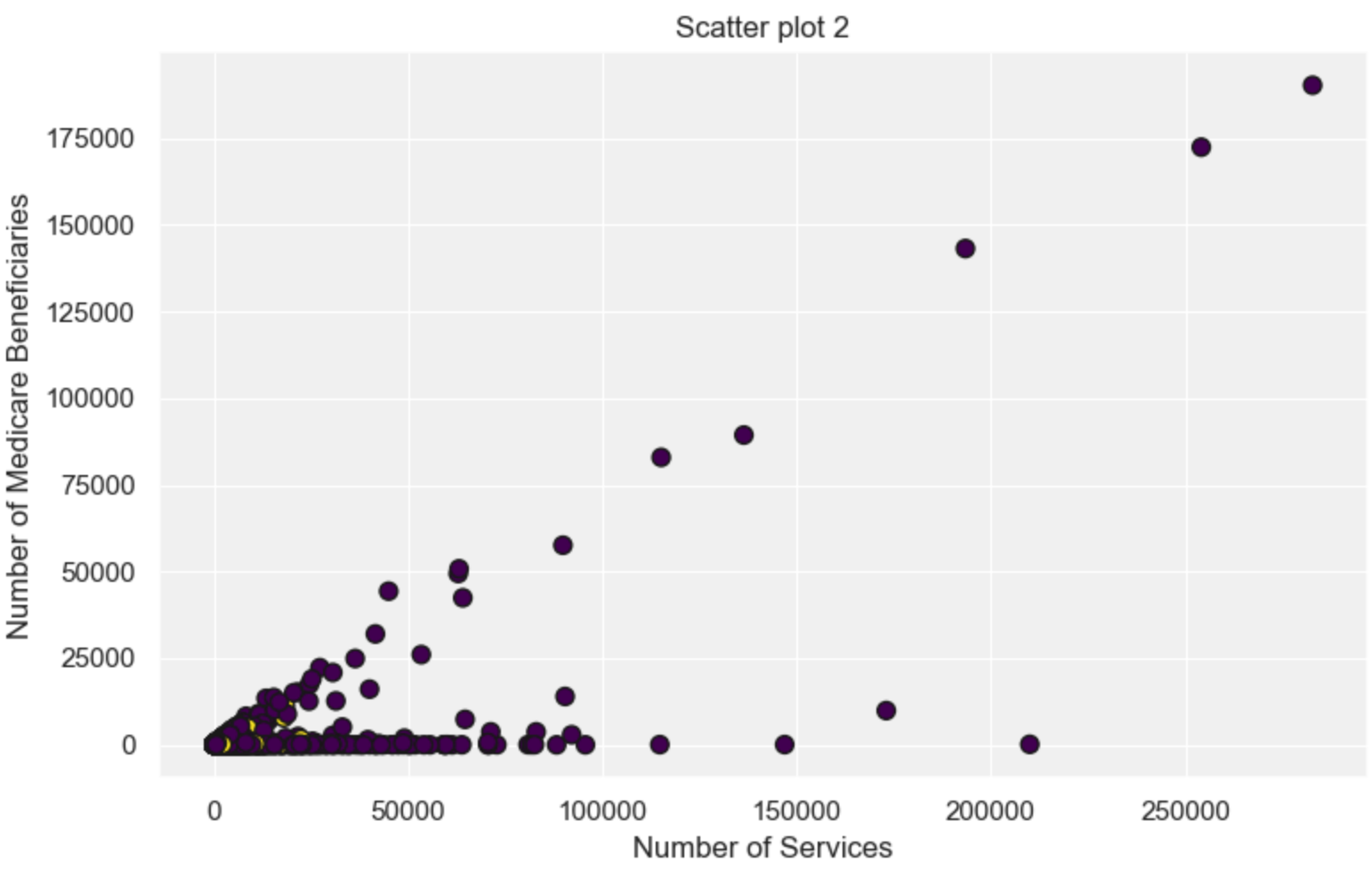
Inferences from the Scatter Plot :

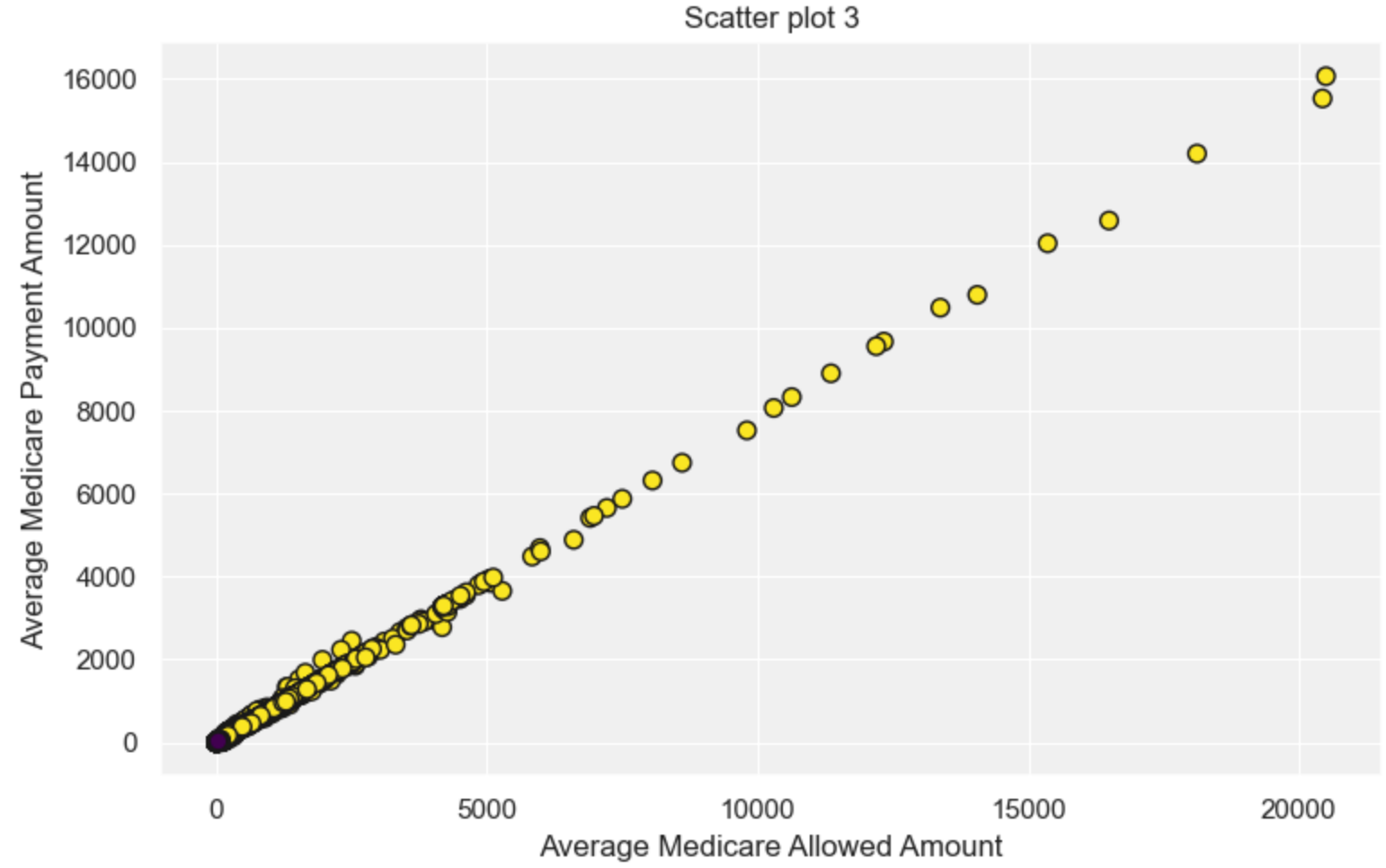
- By using k means clustering clusters are assigned there are total 4 numbers of clusters are present.
- by analysing plot we can observe that some clusters are close while some has wide spread and far from other cluster

In [129…

```
sns.set(rc={'axes.facecolor': '#F0F0F0'}, style='darkgrid')
#visualization on Scatter plots
fig, axs = plt.subplots(3, figsize=(8,15),tight_layout=True)

#plotting Scatter Plots
axs[0].scatter(hdata['Number of Medicare Beneficiaries'], hdata['Number of Distinct Medicare Beneficiary/Per Day Ser
               cmap='viridis',edgecolor='k', s=50)
axs[0].set_title('Scatter plot 1')
axs[0].set_xlabel('Number of Medicare Beneficiaries')
axs[0].set_ylabel('Number of Distinct Medicare Beneficiary/Per Day Services')


axs[1].scatter(hdata['Number of Services'], hdata['Number of Medicare Beneficiaries'], c=pca_df['Cluster'],
               cmap='viridis',edgecolor='k', s=50)
axs[1].set_title('Scatter plot 2')
axs[1].set_xlabel('Number of Services')
axs[1].set_ylabel('Number of Medicare Beneficiaries')

axs[2].scatter(hdata['Average Medicare Allowed Amount'], hdata['Average Medicare Payment Amount'], c=pca_df['Cluster
               cmap='viridis',edgecolor='k', s=50)
axs[2].set_title('Scatter plot 3')
axs[2].set_xlabel('Average Medicare Allowed Amount')
axs[2].set_ylabel('Average Medicare Payment Amount')

plt.show()
```







Inferences from the Scatter Plot :

- scatter plot1 shows that around between 0 to 25000 on x axis cluster 0 is present and clusster 1 has wide spread .

- in scatter plot 3 the Average Medicare Allowed Amount against Average Medicare Payment Amount where cluster point are positively related showing linear characters

DBSCAN Clustering

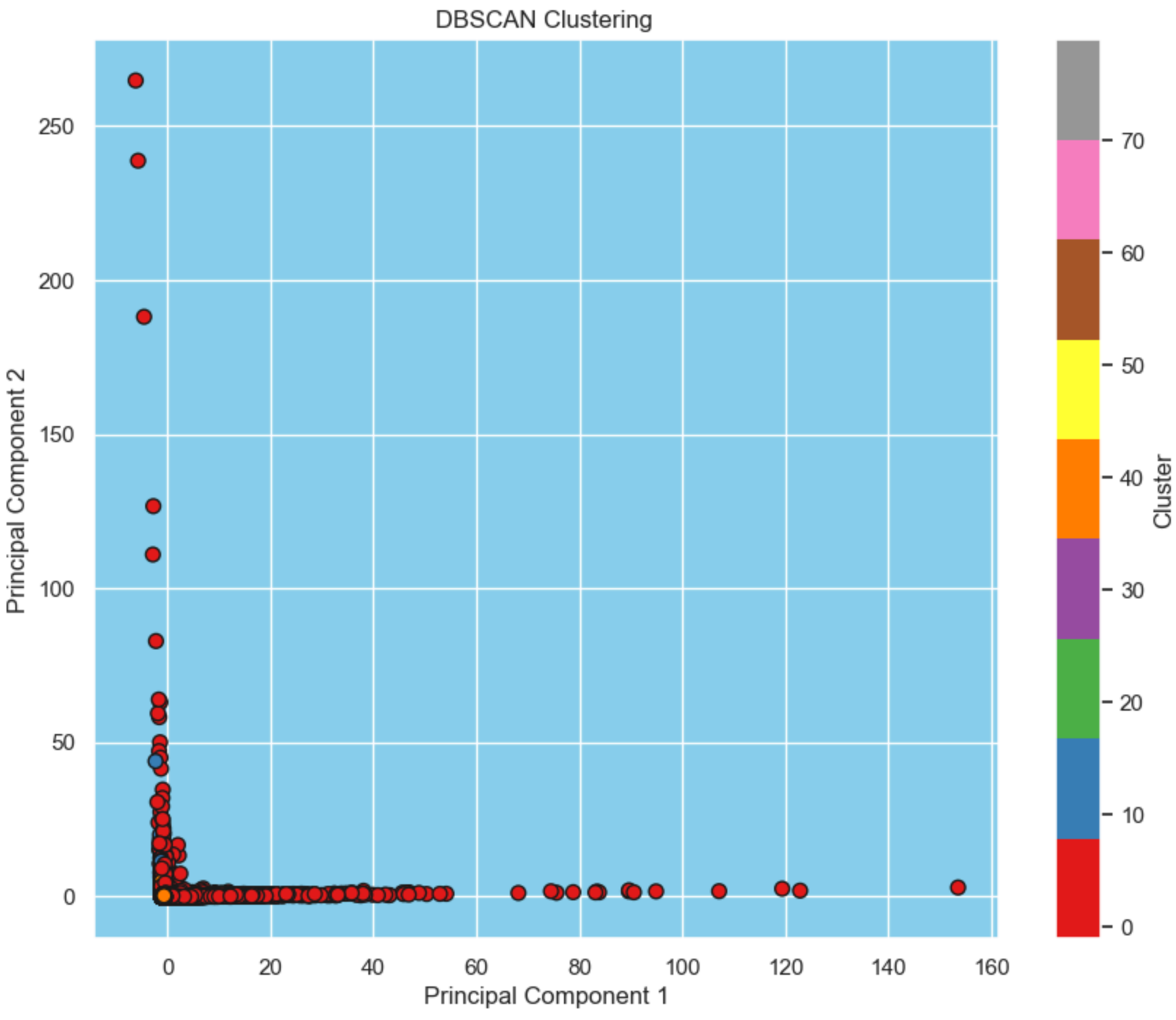
In [130…
```
DropCols = ['Cluster']
hdata_scaled = hdata_scaled.drop(DropCols, axis = 1)
```

In [131…
```
# Initialize DBSCAN
dbscan = DBSCAN(eps=0.5,min_samples=50)
hdata_scaled['Cluster'] = dbscan.fit_predict(hdata_scaled.iloc[:, 1:].to_numpy())

hdata_scaled.sample(5)
```

Out[131…

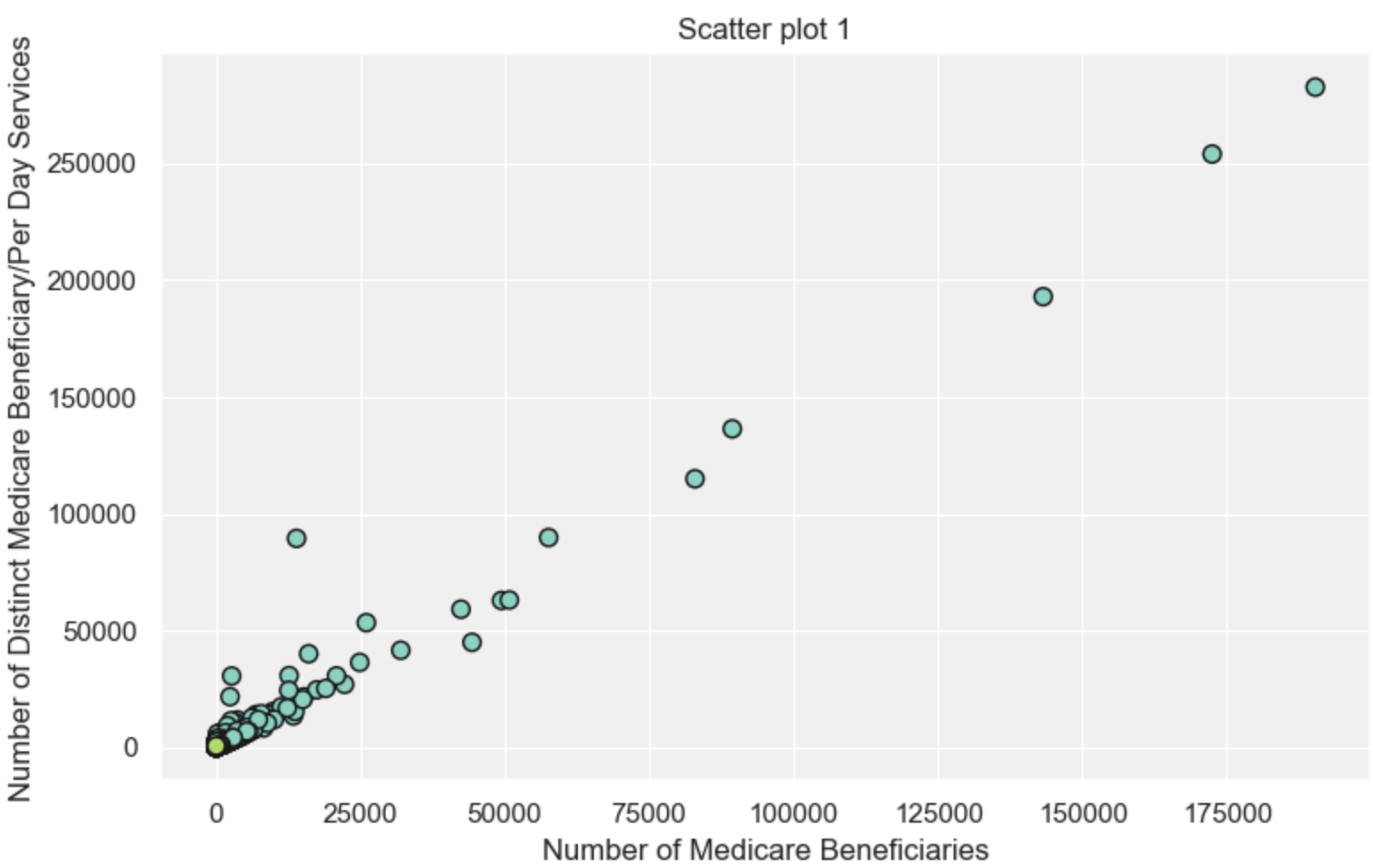
	Number of Services	Number of Medicare Beneficiaries	Number of Distinct Medicare Beneficiary/Per Day Services	Average Medicare Allowed Amount	Average Submitted Charge Amount	Average Medicare Payment Amount	Average Medicare Standardized Amount	Diff_submitted_allowed	Is_
59657	-0.070461	-0.041284	-0.047625	0.021300	-0.081437	0.018279	0.027544	-0.103467	0.007
41495	-0.091318	-0.070123	-0.079328	0.350802	-0.153280	0.553207	0.548945	-0.284621	0.007
27830	0.122466	0.308388	0.245628	-0.382342	-0.289368	-0.372219	-0.374970	-0.235159	0.007
1392	-0.090516	-0.068321	-0.078109	-0.349435	-0.300993	-0.342030	-0.345037	-0.258567	0.007
12398	-0.089312	-0.068321	-0.076280	-0.223425	-0.292292	-0.258980	-0.258552	-0.284621	0.007

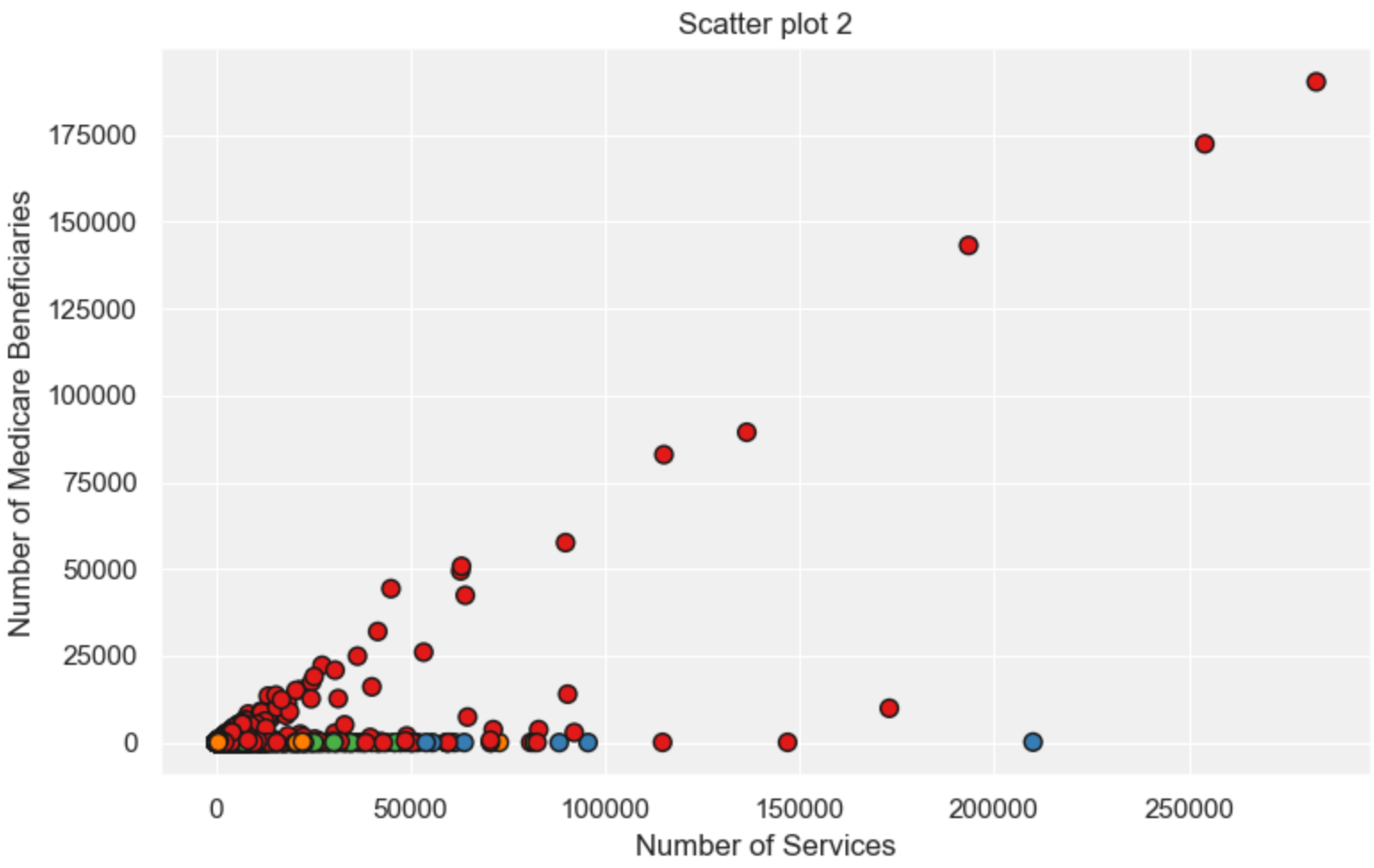
In [132…
```
sns.set(rc={'axes.facecolor': 'skyblue'}, style='darkgrid')
# Create a DataFrame with the principal components and cluster labels
pca_df = pd.DataFrame(data=data_pca, columns=['PC1', 'PC2'])
pca_df['Cluster'] = hdata_scaled['Cluster']

# Plot clusters
plt.figure(figsize=(10, 8))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=pca_df['Cluster'], cmap='Set1', edgecolor='k', s=50)
plt.title('DBSCAN Clustering ')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

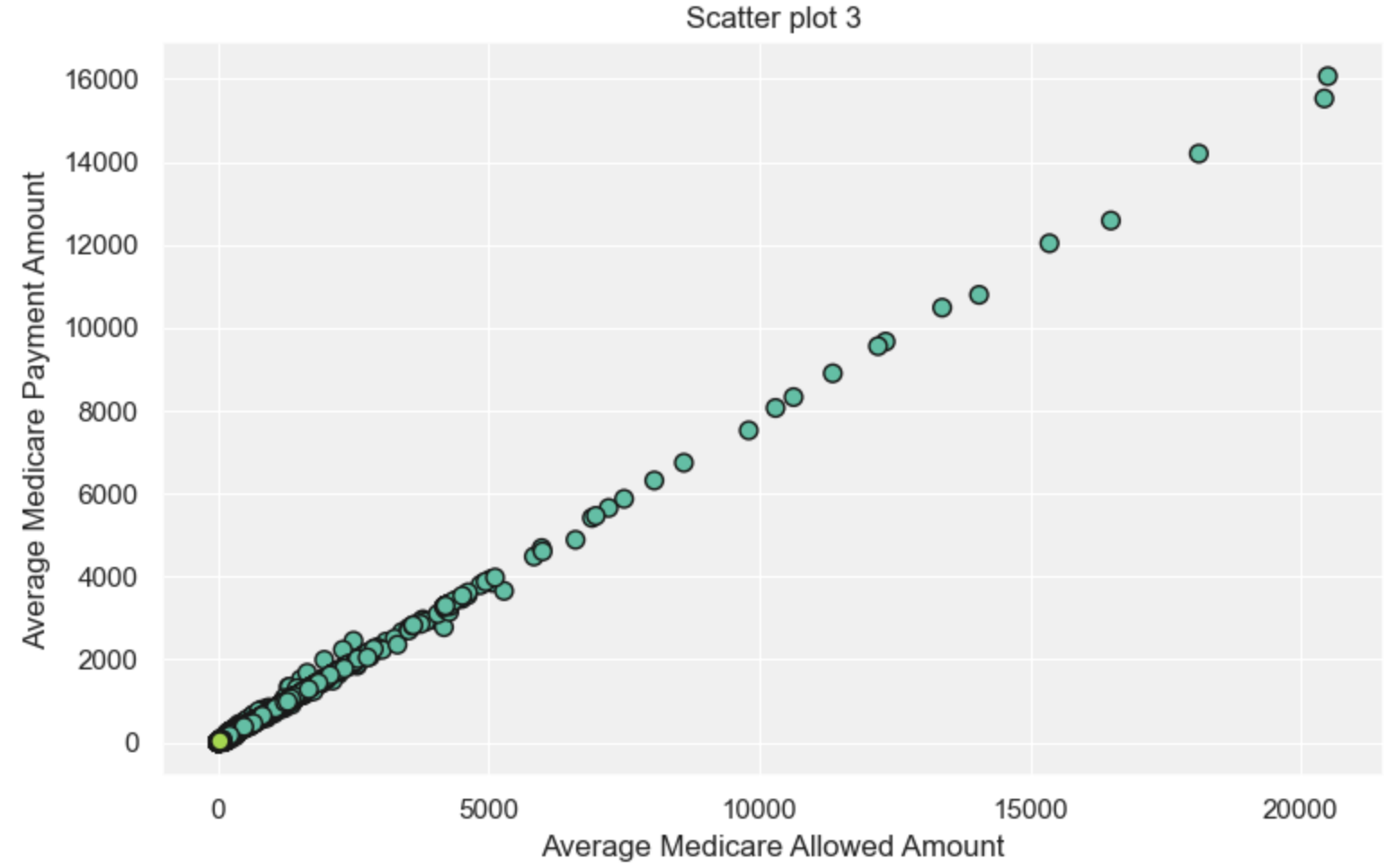


In [133…

```
sns.set(rc={'axes.facecolor': '#F0F0F0'}, style='darkgrid')
#visualization on Scatter plots
fig, axs = plt.subplots(3, figsize=(8,15),tight_layout=True)

#plotting Scatter Plots
axs[0].scatter(hdata['Number of Medicare Beneficiaries'], hdata['Number of Distinct Medicare Beneficiary/Per Day Se
               cmap='Set3',edgecolor='k', s=50)
axs[0].set_title('Scatter plot 1')
axs[0].set_xlabel('Number of Medicare Beneficiaries')
axs[0].set_ylabel('Number of Distinct Medicare Beneficiary/Per Day Services')


axs[1].scatter(hdata['Number of Services'], hdata['Number of Medicare Beneficiaries'], c=pca_df['Cluster'],
               cmap='Set1',edgecolor='k', s=50)
axs[1].set_title('Scatter plot 2')
axs[1].set_xlabel('Number of Services')
axs[1].set_ylabel('Number of Medicare Beneficiaries')

axs[2].scatter(hdata['Average Medicare Allowed Amount'], hdata['Average Medicare Payment Amount'], c=pca_df['Cluste
               cmap='Set2',edgecolor='k', s=50)
axs[2].set_title('Scatter plot 3')
axs[2].set_xlabel('Average Medicare Allowed Amount')
axs[2].set_ylabel('Average Medicare Payment Amount')

plt.show()
```







Inferences from the Scatter Plot :

- scatter plot 2 shows that multiple clusters are on lower boundaries .

- in scatter plot 3 the Average Medicare Allowed Amount against Average Medicare Payment Amount where cluster point are positively related showing linear characters