

Problem Statement:

In this project, we focus on improving the healthcare system by analyzing a dataset from a USA-based health service provider, available on Kaggle. This dataset includes all transactions between patients and service providers. Our main goal is to enhance the efficiency of the healthcare system and prevent fraudulent activities. By detecting anomalies, we aim to improve security, streamline operations, and provide better patient service.

```
In [ ]: #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('Healthcare Providers.csv')
```

```
In [ ]: hdata.info()
```

```

-----
0  Credentials of the Provider      Milestone 2      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 [ ]: hdata.describe(include='object')
```

count	100000	100000	100000	100000	100000	100000	100000
unique	1539	2	2	Milestone 2	58	4	90
top	MD	M	I	NEW YORK	CA	US	Diagnostic Radiology
freq	73827	70895	95746	1061	7775	99994	12537

Feature Engineering

```
In [ ]: #generating new feature name Diff_submitted_allowed column in our dataset

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

# Create a binary column indicating whether the customer is from the UK or no
hdata['Is_US'] = hdata['Country Code of the Provider'].apply(lambda x: 1 if x

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

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

# Create a binary column indicating Medicare Participation Indicator
hdata['Medicare_Participation_Indicator'] = hdata['Medicare Participation Ind

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

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

DropCols = ['Country Code of the Provider', 'Gender of the Provider', 'Entity T
            'Place of Service', 'HCPCS Drug Indicator']
hdata = hdata.drop(DropCols, axis = 1)
```

```
In [ ]: hdata.head().T
```

State Code of the Provider	MO	NC	CT	
ProviderType	Internal Medicine	Obstetrics & Gynecology	Podiatry	In Men
HCPCS Description	Initial hospital inpatient care, typically 70 ...	Screening mammography, bilateral (2- view study...	Established patient home visit, typically 25 m...	Urin manu:
Number of Services	27.0	175.0	32.0	
Number of Medicare Beneficiaries	24	175	13	
Number of Distinct Medicare Beneficiary/Per Day Services	27	175	32	
Average Medicare Allowed Amount	200.587778	123.73	90.65	
Average Submitted Charge Amount	305.211111	548.8	155.0	
Average Medicare Payment Amount	157.262222	118.83	64.439688	
Average Medicare Standardized Amount	160.908889	135.315257	60.595937	
Name	UPADHYAYULA SATYASREE	JONES WENDY P	DUROCHER RICHARD W	FUL JA
Full Address	1402 S GRAND BLVD FDT 14TH FLOOR	2950 VILLAGE DR	20 WASHINGTON AVE STE 212	5' BROAD
Diff_submitted_allowed	104.623333	425.07	64.35	
Is_US	1	1	1	
Gender	0	0	1	
Entity	1	1	1	
Medicare_Participation_Indicator	1	1	1	
Place_of_Service	0	1	1	
HCPCS_Drug_Indicator	1	1	1	

- 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 hence 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.

Frequency Encoding

```
In [ ]: #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', 'S
                    'ProviderType', 'HCPCS Description']

hdata_encoded = frequency_encode_columns(hdata, columns_to_encode)

#Dropping the original columns for maintaining single entries
hdata_encoded = hdata_encoded.drop(columns_to_encode, axis = 1)

In [ ]: hdata_encoded.sample(5).T
```

	Number of Distinct Medicare Beneficiary/Per Day Services	Milestone 2 160	121	216	21	
	Average Medicare Allowed Amount	88.41925	37.66	8.39	238.6	163.3
	Average Submitted Charge Amount	494.975	45.0	52.825112	597.0	
	Average Medicare Payment Amount	68.184875	28.208595	6.480583	183.950952	
	Average Medicare Standardized Amount	70.652125	24.031157	6.697892	192.440952	138.8
	Name	KIM JOSEPH S	NICKLES STEVEN L	KELLY GENE	FORESMAN WILLIAM	DEK J
	Full Address	13700 ST FRANCIS BLVD SUITE 103	581 N FRANKLIN TPKE	2901 N 4TH ST	192 GENESEE ST	2660 ST S
	Diff_submitted_allowed	406.55575	7.34	44.435112	358.4	399.6
	Is_US	1	1	1	1	
	Gender	1	1	1	1	
	Entity	1	1	1	1	
	Medicare_Participation_Indicator	1	1	1	1	
	Place_of_Service	1	1	0	1	
	HCPCS_Drug_Indicator	1	1	1	1	
	Credentials of the Provider_FrequencyEncoded	0.73827	0.06176	0.73827	0.73827	0.
	City of the Provider_FrequencyEncoded	0.00043	0.00008	0.00104	0.00082	0.
	State Code of the Provider_FrequencyEncoded	0.02792	0.03333	0.06861	0.06361	0.
	ProviderType_FrequencyEncoded	0.03098	0.0976	0.02377	0.01794	0.
	HCPCS Description_FrequencyEncoded	0.00128	0.00018	0.0063	0.00024	0.

Inferences from the Feature Encoding :

Feature Scaling

```
In [ ]: # 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
```

Number of Distinct Medicare Beneficiary/Per Day Services	-0.070183	0.020049	-0.067135	-0.074451	-0.067
Average Medicare Allowed Amount	0.385450	0.086673	-0.041922	-0.380709	-0.291
Average Submitted Charge Amount	-0.046433	0.182805	-0.187794	-0.328957	-0.296
Average Medicare Payment Amount	0.400082	0.207649	-0.064687	-0.370166	-0.289
Average Medicare Standardized Amount	0.414299	0.286359	-0.087154	-0.372921	-0.294
Diff_submitted_allowed	-0.166975	0.193356	-0.212261	-0.282934	-0.269
Is_US	0.007746	0.007746	0.007746	0.007746	0.007
Gender	-1.560716	-1.560716	0.640731	0.640731	0.640
Entity	0.210784	0.210784	0.210784	0.210784	0.210
Medicare_Participation_Indicator	0.017610	0.017610	0.017610	0.017610	0.017
Place_of_Service	-1.266985	0.789275	0.789275	0.789275	0.789
HCPCS_Drug_Indicator	0.257051	0.257051	0.257051	0.257051	0.257
Credentials of the Provider_FrequencyEncoded	0.594983	0.594983	-1.684316	0.594983	-1.549
City of the Provider_FrequencyEncoded	1.571686	0.189180	-0.756245	0.702275	-0.561
State Code of the Provider_FrequencyEncoded	-0.737342	-0.004973	-0.989093	-0.737342	1.494
ProviderType_FrequencyEncoded	1.336743	-0.940500	-0.720441	1.336743	1.336
HCPCS Description_FrequencyEncoded	0.389268	-0.450300	-0.608815	-0.277448	-0.060

Inferences from the Feature Scaling :

- Applied **StandardScaler** to scale the encoded features.
- Also Dropped the identifiers from the dataset such as '**Name**', '**Full Address**'.

Dimensionality Reduction

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

PCA is an excellent starting point because it effectively captures 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 while retaining a significant amount of information, making our clustering analysis potentially more accurate and interpretable. Additionally, PCA is computationally efficient, ensuring it won't significantly increase processing time.

However, we remain open to exploring other methods. If PCA's first few components do not capture a substantial amount of variance, indicating a loss of vital information, we will consider non-linear methods. These methods, such as KernelPCA, ICA, ISOMAP, TSNE, and UMAP, can provide a more nuanced approach to dimensionality reduction by capturing complex patterns that PCA might miss, though they come with 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 [ ]: # 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))),
                      y=explained_variance_ratio,
                      color='#fcc36d',
                      alpha=0.8)
```

```

optimal_k_line = plt.axvline(optimal_k - 1, color='red', linestyle='--')
Milestone 2

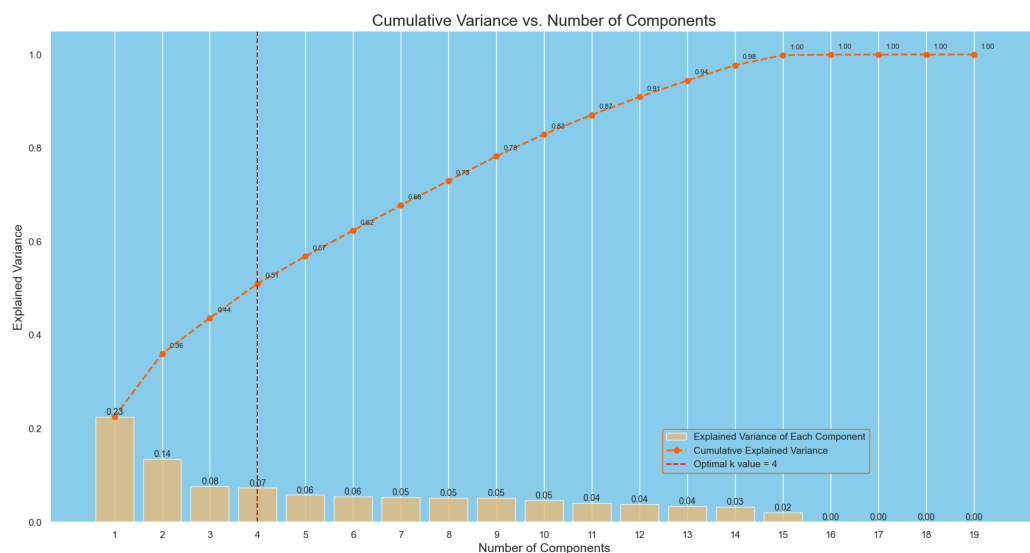
# Set labels and title
plt.xlabel('Number of Components', fontsize=14)
plt.ylabel('Explained Variance', fontsize=14)
plt.title('Cumulative Variance vs. Number of Components', fontsize=18)

# 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',
                  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",
             if i > 0:
                 plt.text(i + x_offset, cum_ev_ratio + y_offset, f"{cum_ev_ratio:.2f}")

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:

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 [ ]: # 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
data_pca = pd.DataFrame(data_pca, columns=['PC'+str(i+1) for i in range(2)])

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

```
Out[ ]:
```

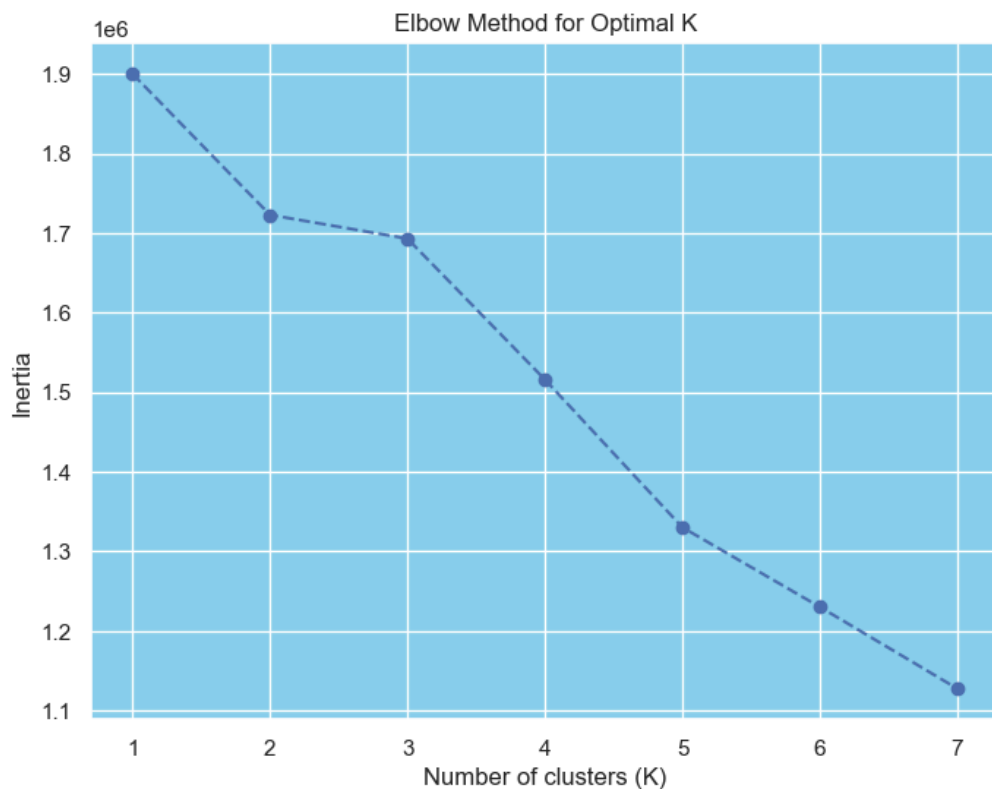
	PC1	PC2
0	0.435022	-0.288328
1	0.392231	0.052477
2	-0.323010	-0.134462
3	-0.868838	-0.183356
4	-0.824890	-0.188083
5	-0.734071	-0.040215
6	-0.470852	-0.299845
7	-0.688050	-0.095425
8	-0.736538	-0.132574
9	-0.369886	0.406647

Cluster Visualization

```
In [ ]: # Initialize a list to store the values of inertia (sum of squared di
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()
```



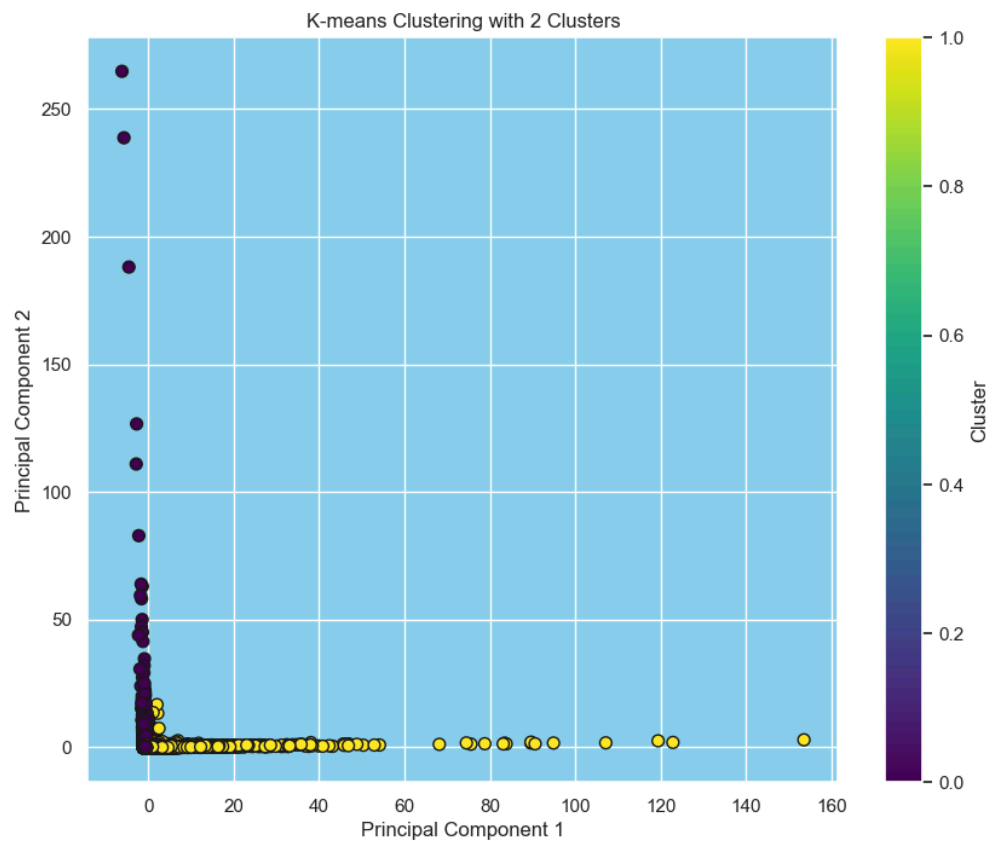
Inferences from the Plot :

- 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 [ ]: # Initialize K-means with 2 clusters
kmeans = KMeans(n_clusters=2, random_state=0)
```

```
In [ ]: # 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=
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

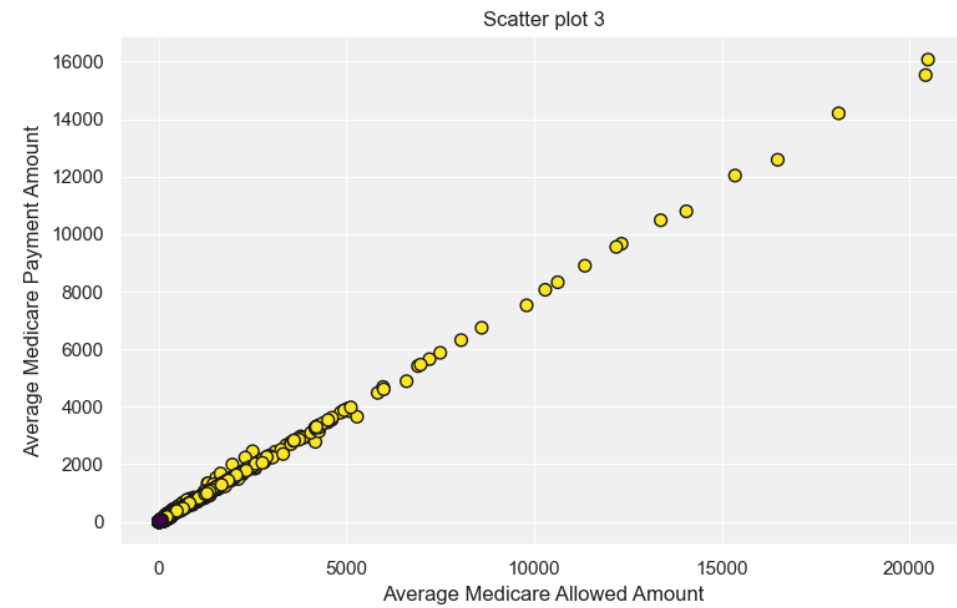
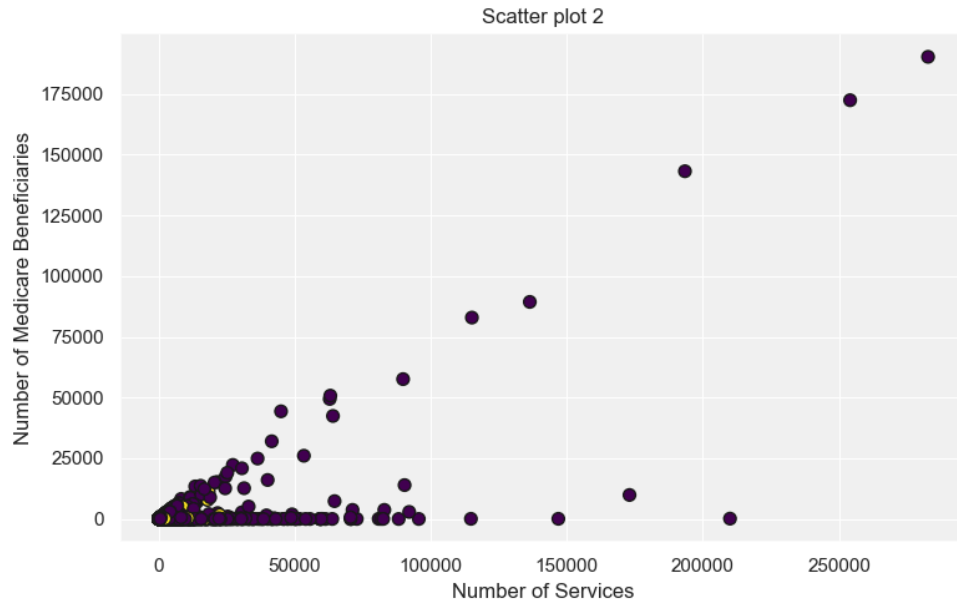
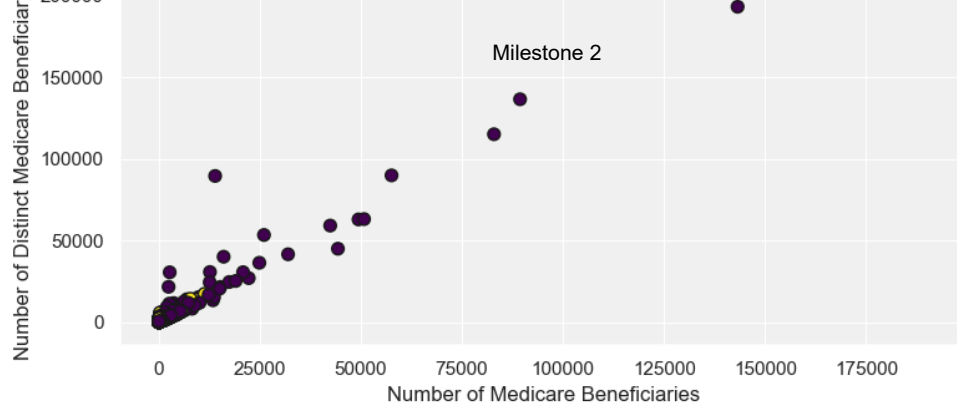
```
In [ ]: sns.set(rc={'axes.facecolor': '#F0F0F0'}, style='darkgrid')
#visualization on Scatter plots
```

```
axs[0].set_xlabel('Number of Medicare Beneficiaries')
axs[0].set_ylabel('Number of Distinct Medicare Beneficiary/Per Day')

axs[1].scatter(hdata['Number of Services'], hdata['Number of Medicare Beneficiaries'],
               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'],
               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 :

DBSCAN Clustering

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

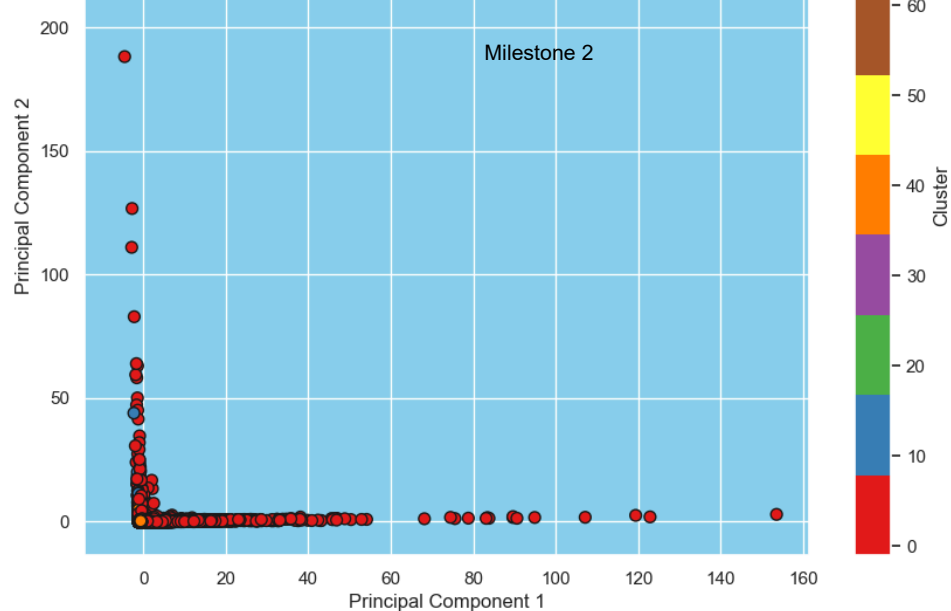
```
In [ ]: # Initialize DBSCAN
dbscan = DBSCAN(eps=0.5,min_samples=50)
hdata_scaled['Cluster'] = dbscan.fit_predict(hdata_scaled.iloc[:,
hdata_scaled.sample(5)
```

```
Out[ ]:
```

	Number of Services	Number of Medicare Beneficiaries	Number of Distinct Medicare Beneficiary/Per Day Services	Average Medicare Allowed Amount	Average Submitted Charge Amount
59657	-0.070461	-0.041284	-0.047625	0.021300	-0.081437
41495	-0.091318	-0.070123	-0.079328	0.350802	-0.153280
27830	0.122466	0.308388	0.245628	-0.382342	-0.289368
1392	-0.090516	-0.068321	-0.078109	-0.349435	-0.300993
12398	-0.089312	-0.068321	-0.076280	-0.223425	-0.292292

```
In [ ]: 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'], cr
plt.title('DBSCAN Clustering ')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar(label='Cluster')
plt.show()
```

```
In [ ]: 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 D
               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 D

axs[1].scatter(hdata['Number of Services'], hdata['Number of Medicare Beneficiaries'],
               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'],
               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()
```

