In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

#filtering the warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
df = pd.read_csv('Healthcare Providers.csv')
```

In [3]:
```
df.shape
```

Out[3]:
```
(100000, 27)
```

Data Pre-processing

Handling Missing Values

In [4]:
```
# Checking missing values
df.isnull().sum()
```

Out[4]:
```
index                                                           0
National Provider Identifier                                    0
Last Name/Organization Name of the Provider                     0
First Name of the Provider                                   4255
Middle Initial of the Provider                              29331
Credentials of the Provider                                  7209
Gender of the Provider                                       4254
Entity Type of the Provider                                     0
Street Address 1 of the Provider                                0
Street Address 2 of the Provider                            59363
City of the Provider                                            0
Zip Code of the Provider                                        0
State Code of the Provider                                      0
Country Code of the Provider                                    0
Provider Type                                                   0
Medicare Participation Indicator                                0
Place of Service                                                0
HCPCS Code                                                      0
HCPCS Description                                               0
HCPCS Drug Indicator                                            0
Number of Services                                              0
Number of Medicare Beneficiaries                                0
Number of Distinct Medicare Beneficiary/Per Day Services        0
Average Medicare Allowed Amount                                 0
Average Submitted Charge Amount                                 0
Average Medicare Payment Amount                                 0
Average Medicare Standardized Amount                            0
dtype: int64
```

In [5]:
```
# Dropping columns which are not needed

DropColumns = ['index', 'National Provider Identifier','Last Name/Organization Name of the Provider',
       'First Name of the Provider', 'Middle Initial of the Provider','Street Address 1 of the Provider',
       'Street Address 2 of the Provider']
df = df.drop(DropColumns, axis=1)
```

In [6]:
```
# Filling missing with mode values
df["Credentials of the Provider"] = df["Credentials of the Provider"].fillna(df["Credentials of the Provider"].
df["Gender of the Provider"] = df["Gender of the Provider"].fillna(df["Gender of the Provider"].mode()[0])
```

In [7]:
```
df.isnull().sum()
```

Out[7]:

```
Credentials of the Provider                                 0
Gender of the Provider                                      0
Entity Type of the Provider                                 0
City of the Provider                                        0
Zip Code of the Provider                                    0
State Code of the Provider                                  0
Country Code of the Provider                                0
Provider Type                                               0
Medicare Participation Indicator                            0
Place of Service                                            0
HCPCS Code                                                  0
HCPCS Description                                           0
HCPCS Drug Indicator                                        0
Number of Services                                          0
Number of Medicare Beneficiaries                            0
Number of Distinct Medicare Beneficiary/Per Day Services    0
Average Medicare Allowed Amount                             0
Average Submitted Charge Amount                             0
Average Medicare Payment Amount                             0
Average Medicare Standardized Amount                        0
dtype: int64
```

In [8]:

```python
# Cleaning 'Credentials of the Provider' Column
# Removing periods '.' if present
df['Credentials of the Provider'] = df['Credentials of the Provider'].str.replace('.', '')
```

In [9]:

```python
df.head()
```

Out[9]:

	Credentials of the Provider	Gender of the Provider	Entity Type of the Provider	City of the Provider	Zip Code of the Provider	State Code of the Provider	Country Code of the Provider	Provider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Description
0	MD	F	I	SAINT LOUIS	631041004.0	MO	US	Internal Medicine	Y	F	99223	Initial hospita inpatient care, typically 70 ...
1	MD	F	I	FAYETTEVILLE	283043815.0	NC	US	Obstetrics & Gynecology	Y	O	G0202	Screening mammography, bilateral (2- view study...
2	DPM	M	I	NORTH HAVEN	64732343.0	CT	US	Podiatry	Y	O	99348	Established patient home visit, typically 25 m...
3	MD	M	I	KANSAS CITY	641183998.0	MO	US	Internal Medicine	Y	O	81002	Urinalysis, manual tes
4	DO	M	I	JUPITER	334585700.0	FL	US	Internal Medicine	Y	O	96372	Injection beneath the skin or intc muscle for ...

In [10]:

```python
# Removing 'Zip Code of the Provider' column as it was evenly distributed(Observed during EDA)
df.drop(columns=['Zip Code of the Provider'], inplace=True)
```

In [11]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 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   Provider Type                                             100000 non-null  object
 7   Medicare Participation Indicator                          100000 non-null  object
 8   Place of Service                                          100000 non-null  object
 9   HCPCS Code                                                100000 non-null  object
 10  HCPCS Description                                         100000 non-null  object
 11  HCPCS Drug Indicator                                      100000 non-null  object
 12  Number of Services                                        100000 non-null  object
 13  Number of Medicare Beneficiaries                          100000 non-null  object
 14  Number of Distinct Medicare Beneficiary/Per Day Services  100000 non-null  object
 15  Average Medicare Allowed Amount                           100000 non-null  object
 16  Average Submitted Charge Amount                           100000 non-null  object
 17  Average Medicare Payment Amount                           100000 non-null  object
 18  Average Medicare Standardized Amount                      100000 non-null  object
dtypes: object(19)
memory usage: 14.5+ MB
```

In [12]:

```python
df.head()
```

Out[12]:

	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	Provider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Description	HCPCS Drug Indicator
0	MD	F	I	SAINT LOUIS	MO	US	Internal Medicine	Y	F	99223	Initial hospital inpatient care, typically 70 ...	N
1	MD	F	I	FAYETTEVILLE	NC	US	Obstetrics & Gynecology	Y	O	G0202	Screening mammography, bilateral (2-view study...	N
2	DPM	M	I	NORTH HAVEN	CT	US	Podiatry	Y	O	99348	Established patient home visit, typically 25 m...	N
3	MD	M	I	KANSAS CITY	MO	US	Internal Medicine	Y	O	81002	Urinalysis, manual test	N
4	DO	M	I	JUPITER	FL	US	Internal Medicine	Y	O	96372	Injection beneath the skin or into muscle for ...	N

In [13]:

```
# Changing all needeed values to numeric

def RemoveComma(x):
    return str(x).replace(",","")  # Convert to string before replacing comma

numericCols = ['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']

df[numericCols] = df[numericCols].applymap(RemoveComma).apply(pd.to_numeric, errors='ignore')
```

In [14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 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   Provider Type                                             100000 non-null  object 
 7   Medicare Participation Indicator                          100000 non-null  object 
 8   Place of Service                                          100000 non-null  object 
 9   HCPCS Code                                                100000 non-null  object 
 10  HCPCS Description                                         100000 non-null  object 
 11  HCPCS Drug Indicator                                      100000 non-null  object 
 12  Number of Services                                        100000 non-null  float64
 13  Number of Medicare Beneficiaries                          100000 non-null  int64  
 14  Number of Distinct Medicare Beneficiary/Per Day Services  100000 non-null  int64  
 15  Average Medicare Allowed Amount                           100000 non-null  float64
 16  Average Submitted Charge Amount                           100000 non-null  float64
 17  Average Medicare Payment Amount                           100000 non-null  float64
 18  Average Medicare Standardized Amount                      100000 non-null  float64
dtypes: float64(5), int64(2), object(12)
memory usage: 14.5+ MB
```

Feature Engineering

Ratio of Medicare Payment Amount to Submitted Charge Amount:

This column would be (Average Medicare Payment Amount / Average Submitted Charge Amount), this would probably helps in providing insights into the ratio of what Medicare pays versus what providers charge on average.

In [15]:

```
# Adding new column as 'Ratio Medicare Payment to Submitted Charge'

# Calculating the ratio
df['Ratio Medicare Payment to Submitted Charge'] = df['Average Medicare Payment Amount'] / df['Average Submitte
```

In [16]:

```
df.head()
```

Out[16]:

	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	Provider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Description	HCPCS Drug Indicator
0	MD	F	I	SAINT LOUIS	MO	US	Internal Medicine	Y	F	99223	Initial hospital inpatient care, typically 70 ...	N
1	MD	F	I	FAYETTEVILLE	NC	US	Obstetrics & Gynecology	Y	O	G0202	Screening mammography, bilateral (2-view study...	N
2	DPM	M	I	NORTH HAVEN	CT	US	Podiatry	Y	O	99348	Established patient home visit, typically 25 m...	N
3	MD	M	I	KANSAS CITY	MO	US	Internal Medicine	Y	O	81002	Urinalysis, manual test	N
4	DO	M	I	JUPITER	FL	US	Internal Medicine	Y	O	96372	Injection beneath the skin or into muscle for ...	N

Encoding

One-hot Encoding

In [17]:

```
# Identify columns to be one-hot encoded
categorical_cols = [col for col in df.columns if df[col].dtype == 'O']

# Perform one-hot encoding
df_one_hot = pd.get_dummies(df, columns=categorical_cols)
```

In [18]:

```
df_one_hot.head()
```

Out[18]:

	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	Ratio Medicare Payment to Submitted Charge	Credentials of the Provider_A-NP	Credentials of the Provider_A/ACNP-BC	...
0	27.0	24	27	200.587778	305.211111	157.262222	160.908889	0.515257	False	False	...
1	175.0	175	175	123.730000	548.800000	118.830000	135.315257	0.216527	False	False	...
2	32.0	13	32	90.650000	155.000000	64.439688	60.595937	0.415740	False	False	...
3	20.0	18	20	3.500000	5.000000	3.430000	3.430000	0.686000	False	False	...
4	33.0	24	31	26.520000	40.000000	19.539394	19.057576	0.488485	False	False	...

5 rows × 12641 columns

One-hot encoding creates huge number of columns i.e., 12641 columns which are unneccessary and creates further difficulties.

Binary Encoding

In [19]:

```
import category_encoders as ce
from sklearn.preprocessing import StandardScaler

BEcols = [var for var in df.columns if df[var].dtype == "O"]

# Create a DataFrame for binary encoding
df_bin_enc = df.copy()

#Performing binary encoding
for col in BEcols:
        encoder = ce.BinaryEncoder(cols = [col])
        dfbin = encoder.fit_transform(df[col])
        df_bin_enc = pd.concat([df_bin_enc,dfbin], axis = 1)
        df_bin_enc.drop(columns=[col], inplace=True)
```

In [20]:

```
df_bin_enc.head()
```

Out[20]:

	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	Ratio Medicare Payment to Submitted Charge	Credentials of the Provider_0	Credentials of the Provider_1	...	H Descrip
0	27.0	24	27	200.587778	305.211111	157.262222	160.908889	0.515257	0	0	...	
1	175.0	175	175	123.730000	548.800000	118.830000	135.315257	0.216527	0	0	...	
2	32.0	13	32	90.650000	155.000000	64.439688	60.595937	0.415740	0	0	...	
3	20.0	18	20	3.500000	5.000000	3.430000	3.430000	0.686000	0	0	...	
4	33.0	24	31	26.520000	40.000000	19.539394	19.057576	0.488485	0	0	...	

5 rows × 82 columns

After encoding using binary encoded technique we get 82 columns only.

Frequency Encoding

In [21]:

```
# Identify columns to be frequency encoded (assuming all categorical columns)
FEcols = [var for var in df.columns if df[var].dtype == "O"]

# Create a DataFrame for frequency encoding
df_freq_enc = df.copy()

# Performing frequency encoding
for col in FEcols:
    encoder = ce.CountEncoder(cols=[col])
    df_fe = encoder.fit_transform(df[col])
    df_freq_enc[col] = df_fe[col]
```

In [22]:

```
df_freq_enc.head()
```

Out[22]:

	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	Provider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Description	HCPCS Drug Indicator	Number of Services	Be
0	73827	29105	95746	500	1997	99994	11366	99969	38384	1297	1297	93802	27.0	
1	73827	29105	95746	209	3725	99994	1028	99969	61616	243	243	93802	175.0	
2	1915	70895	95746	10	1403	99994	2027	99969	61616	44	44	93802	32.0	
3	73827	70895	95746	317	1997	99994	11366	99969	61616	460	460	93802	20.0	
4	6176	70895	95746	51	7263	99994	11366	99969	61616	732	732	93802	33.0	

Standardization

StandardScaler transforms data to have a mean of 0 and a standard deviation of 1.

Standardizing binary encoded data

In [23]:

```
from sklearn.preprocessing import StandardScaler

def Preprocessing(df_bin_enc):
    # Extract column names
    df_cols = df_bin_enc.columns

    # Instantiate StandardScaler
    std = StandardScaler()

    # Apply StandardScaler to numerical columns
    numeric_cols = df_bin_enc.select_dtypes(include=['float64', 'int64']).columns
    df_bin_enc[numeric_cols] = std.fit_transform(df_bin_enc[numeric_cols])

    return df_bin_enc

# Call the Preprocessing function
df_bin_enc = Preprocessing(df_bin_enc)
```

In [24]:

```
df_bin_enc.head()
```

Out[24]:

	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	Ratio Medicare Payment to Submitted Charge	Credentials of the Provider_0	Credentials of the Provider_1	...	HCP Descriptio
0	-0.085301	-0.059308	-0.070183	0.385450	-0.046433	0.400082	0.414299	0.871094	-0.075915	-0.090254	...	-0.674
1	-0.025939	0.076775	0.020049	0.086673	0.182805	0.207649	0.286359	-0.592234	-0.075915	-0.090254	...	-0.674
2	-0.083296	-0.069222	-0.067135	-0.041922	-0.187794	-0.064687	-0.087154	0.383609	-0.075915	-0.090254	...	-0.674
3	-0.088109	-0.064716	-0.074451	-0.380709	-0.328957	-0.370166	-0.372921	1.707476	-0.075915	-0.090254	...	-0.674
4	-0.082895	-0.059308	-0.067744	-0.291221	-0.296019	-0.289505	-0.294800	0.739950	-0.075915	-0.090254	...	-0.674

5 rows × 82 columns

Standardizing frequency encoded data

In [25]:

```python
from sklearn.preprocessing import StandardScaler

def Preprocessing(df_freq_enc):
    # Extract column names
    df_cols = df_freq_enc.columns

    # Instantiate StandardScaler
    std = StandardScaler()

    # Apply StandardScaler to numerical columns
    numeric_cols = df_freq_enc.select_dtypes(include=['float64', 'int64']).columns
    df_freq_enc[numeric_cols] = std.fit_transform(df_freq_enc[numeric_cols])

    return df_freq_enc

# Call the Preprocessing function
df_freq_enc = Preprocessing(df_freq_enc)
```

In [26]:

```python
df_freq_enc.head()
```

Out[26]:

	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	Provider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Description	HCPCS Drug Indicator	Nur Serv
0	0.594983	-1.560716	0.210784	1.571686	-0.737342	0.007746	1.336743	0.01761	-1.266985	0.397579	0.389268	0.257051	-0.08
1	0.594983	-1.560716	0.210784	0.189180	-0.004973	0.007746	-0.940500	0.01761	0.789275	-0.439989	-0.450300	0.257051	-0.02
2	-1.684316	0.640731	0.210784	-0.756245	-0.989093	0.007746	-0.720441	0.01761	0.789275	-0.598126	-0.608815	0.257051	-0.08
3	0.594983	0.640731	0.210784	0.702275	-0.737342	0.007746	1.336743	0.01761	0.789275	-0.267549	-0.277448	0.257051	-0.08
4	-1.549260	0.640731	0.210784	-0.561459	1.494517	0.007746	1.336743	0.01761	0.789275	-0.051402	-0.060785	0.257051	-0.08

Normalization

MinMaxScaler scales data to a [0, 1] range.

Normalizing binary encoded data

In [27]:

```python
from sklearn.preprocessing import MinMaxScaler

def Preprocessing(df_bin_enc):
    # Extract column names
    df_cols = df_bin_enc.columns

    # Instantiate MinMaxScaler
    minmax_scaler = MinMaxScaler()

    # Apply MinMaxScaler to numerical columns
    numeric_cols = df_bin_enc.select_dtypes(include=['float64', 'int64']).columns
    df_bin_enc[numeric_cols] = minmax_scaler.fit_transform(df_bin_enc[numeric_cols])

    return df_bin_enc

# Call the Preprocessing function
df_bin_enc = Preprocessing(df_bin_enc)
```

In [28]:

```python
df_bin_enc.head()
```

Out[28]:

	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	Ratio Medicare Payment to Submitted Charge	Credentials of the Provider_0	Credentials of the Provider_1	...	HCPC Description_
0	0.000057	0.000068	0.000057	0.009787	0.004868	0.009787	0.009489	0.455113	0.0	0.0	...	0
1	0.000580	0.000862	0.000580	0.006037	0.008753	0.007395	0.007979	0.191061	0.0	0.0	...	0
2	0.000074	0.000011	0.000074	0.004423	0.002472	0.004010	0.003573	0.367148	0.0	0.0	...	0
3	0.000032	0.000037	0.000032	0.000170	0.000080	0.000213	0.000202	0.606035	0.0	0.0	...	0
4	0.000078	0.000068	0.000071	0.001294	0.000638	0.001216	0.001123	0.431449	0.0	0.0	...	0

5 rows × 82 columns

Normalizing frequency encoded data

In [29]:

```
from sklearn.preprocessing import MinMaxScaler

def Preprocessing(df_freq_enc):
    # Extract column names
    df_cols = df_freq_enc.columns

    # Instantiate MinMaxScaler
    minmax_scaler = MinMaxScaler()

    # Apply MinMaxScaler to numerical columns
    numeric_cols = df_freq_enc.select_dtypes(include=['float64', 'int64']).columns
    df_freq_enc[numeric_cols] = minmax_scaler.fit_transform(df_freq_enc[numeric_cols])

    return df_freq_enc

# Call the Preprocessing function
df_freq_enc = Preprocessing(df_freq_enc)
```

In [30]:

```
df_freq_enc.head()
```

Out[30]:

	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	Provider Type	Medicare Participation Indicator	Place of Service	HCPCS Code	HCPCS Description	HCPCS Drug Indicator	Number of Services
0	1.000000	0.0	1.0	0.470755	0.256753	1.0	0.906589	1.0	0.0	0.283155	0.283155	1.0	0.000057
1	1.000000	0.0	1.0	0.196226	0.479033	1.0	0.081924	1.0	1.0	0.052873	0.052873	1.0	0.000580
2	0.025926	1.0	1.0	0.008491	0.180345	1.0	0.161615	1.0	1.0	0.009395	0.009395	1.0	0.000074
3	1.000000	1.0	1.0	0.298113	0.256753	1.0	0.906589	1.0	1.0	0.100284	0.100284	1.0	0.000032
4	0.083643	1.0	1.0	0.047170	0.934139	1.0	0.906589	1.0	1.0	0.159712	0.159712	1.0	0.000078

Dimentionality Reduction

Principal Component Analysis (PCA)

PCA for Binary Encoded Data

In [31]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

scaler = StandardScaler()
scaler.fit(df_bin_enc)

scaled_data_bin = scaler.transform(df_bin_enc)

pca = PCA(n_components=2)
pca.fit(scaled_data_bin)

x_pca_bin = pca.transform(scaled_data_bin)

print(scaled_data_bin.shape)
print(x_pca_bin.shape)
```

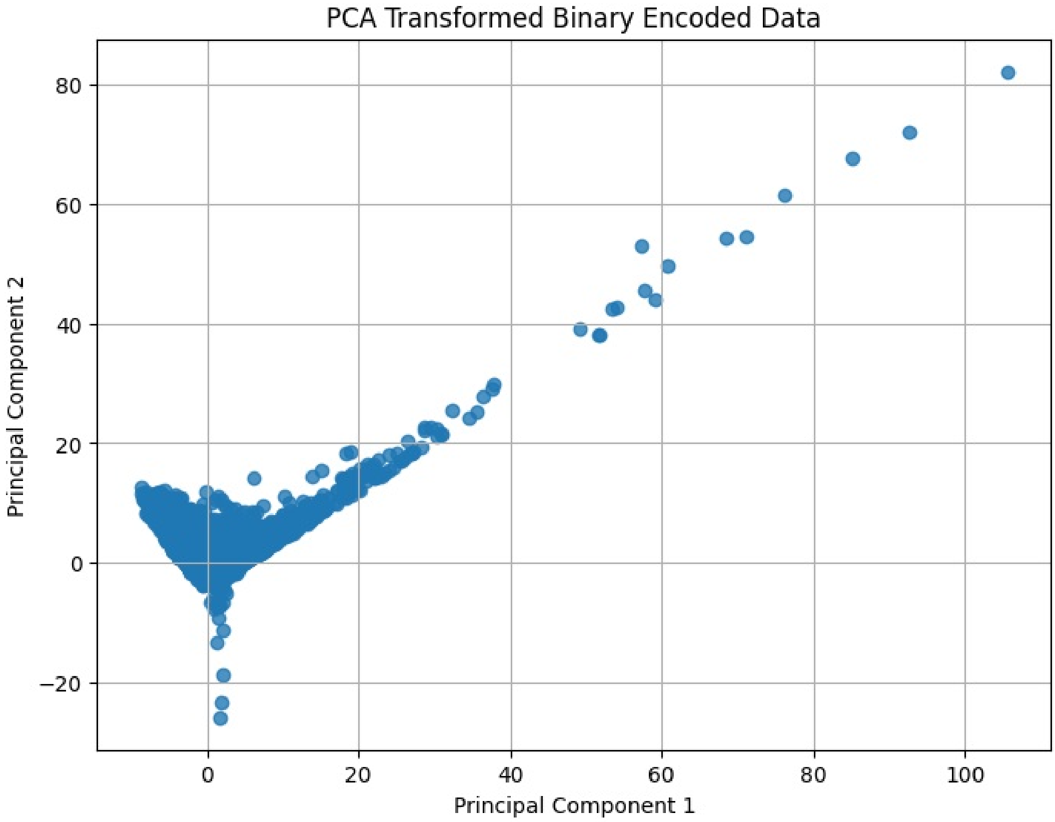
```
(100000, 82)
(100000, 2)
```

After applying PCA we have reduced all the dimensions in just two columns.

After applying PCA we have reduced all the dimensions in just two columns.

In [32]:

```
# Plotting PCA transformed binary encoded data

plt.figure(figsize=(8, 6))
plt.scatter(x_pca_bin[:, 0], x_pca_bin[:, 1], alpha=0.8)
plt.title('PCA Transformed Binary Encoded Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```



PCA for Frequency Encoded Data

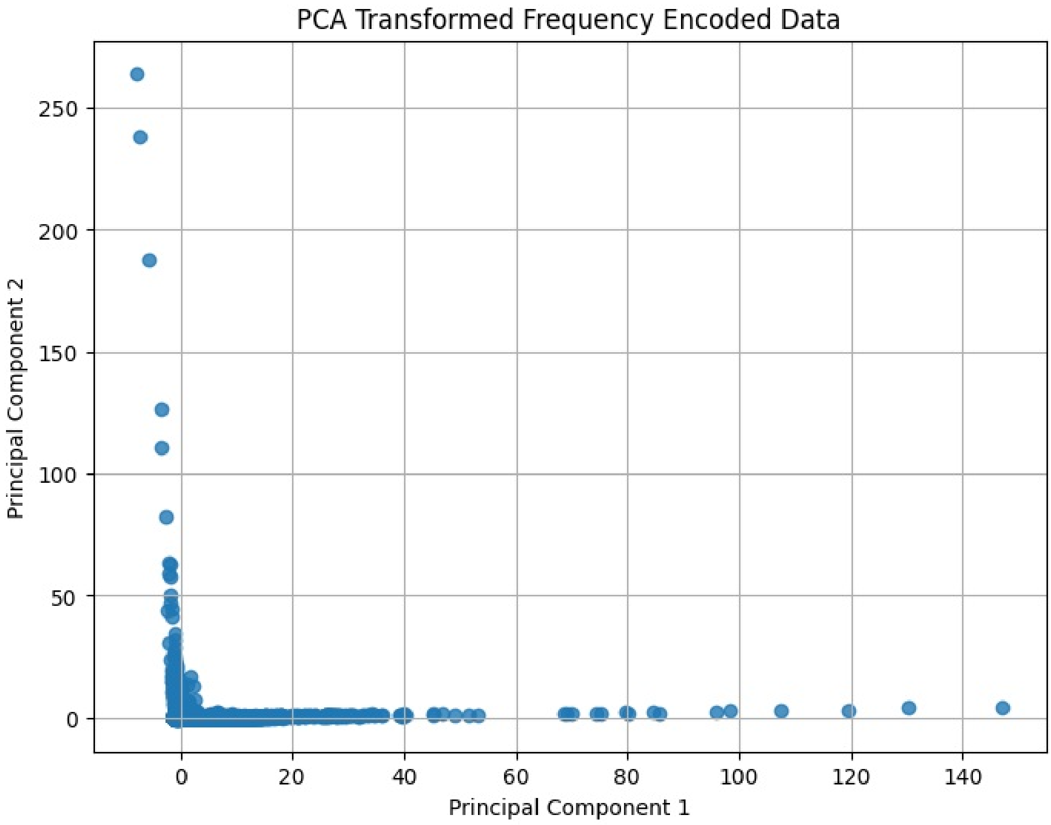
In [33]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA

scaler = StandardScaler()
scaler.fit(df_freq_enc)

scaled_data_freq = scaler.transform(df_freq_enc)

pca = PCA(n_components=2)
pca.fit(scaled_data_freq)

x_pca_freq = pca.transform(scaled_data_freq)

print(scaled_data_freq.shape)
print(x_pca_freq.shape)
```

```
(100000, 20)
(100000, 2)
```

In [34]:

```
# Plotting PCA transformed frequency encoded data

plt.figure(figsize=(8, 6))
plt.scatter(x_pca_freq[:, 0], x_pca_freq[:, 1], alpha=0.8)
plt.title('PCA Transformed Frequency Encoded Data')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```



Insights: Here, we are proceeding with frequency encoding data rather than binary encoded one as frequency encoded has less number of attributes and tells us which data have occurred most of the time which makes our model understands better to detect the anomalies.

Clustering

Frequency Encoded Data

K-means Clustering

In [35]:

```
from sklearn.cluster import KMeans
wcss_freq = []

for i in range(1,11):
    kmeans = KMeans(n_clusters = i, init='k-means++', random_state=42)
    kmeans.fit(x_pca_freq)
    wcss_freq.append(kmeans.inertia_)
```

In [36]:

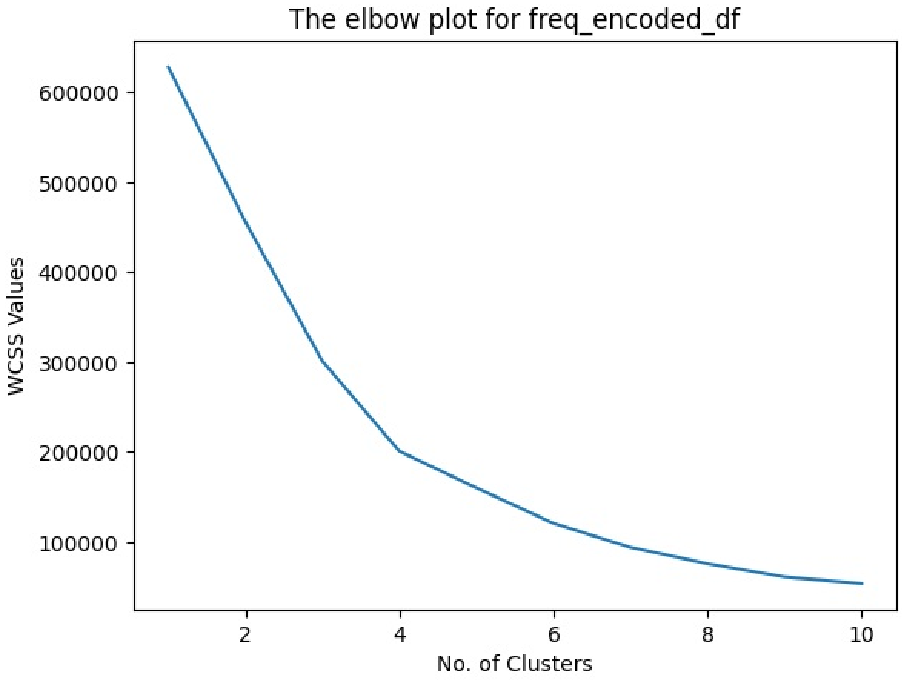
```
wcss_freq
```

Out[36]:

```
[628279.4708994608,
 456218.55243241834,
 301240.6107061327,
 200869.60409980651,
 160175.81555747622,
 120758.53531742952,
 93972.06950264452,
 75852.21434115288,
 61065.027706446905,
 53783.87424097911]
```

In [37]:

```
plt.plot(range(1,11), wcss_freq)
plt.title('The elbow plot for freq_encoded_df')
plt.xlabel('No. of Clusters')
plt.ylabel('WCSS Values')
plt.show()
```



Here, we can see that 4 is one of the point from where the graph is declining, so the ideal number of cluster can be 4.

```
In [38]: # Applying K-means clustering
X = x_pca_freq
km = KMeans(n_clusters=4, random_state=42)
y_means = km.fit_predict(X)
```

```
In [39]: y_means
```

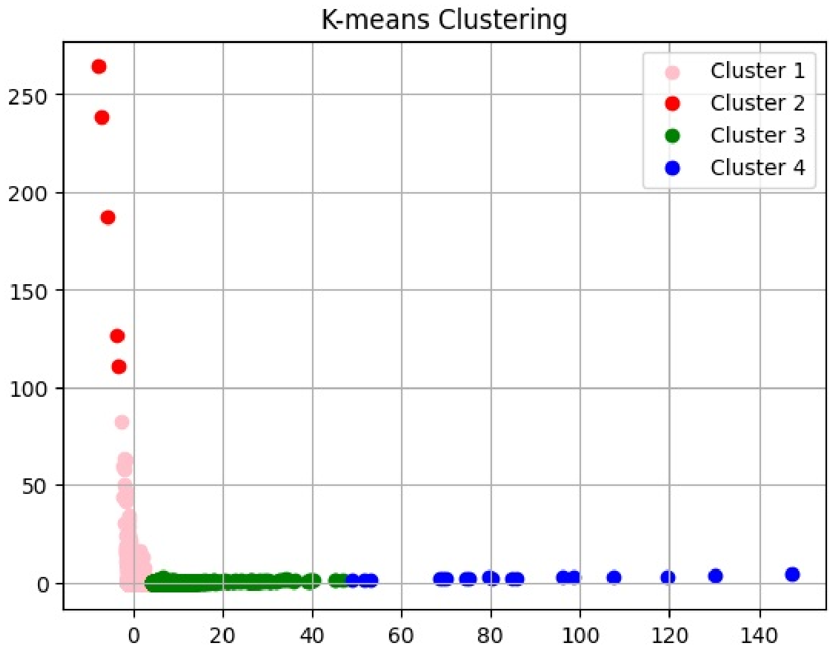
```
Out[39]: array([0, 0, 0, ..., 0, 0, 0])
```

```
In [40]: X[y_means == 3,1]
```

```
Out[40]: array([1.31908248, 3.97064042, 4.16964588, 1.94813348, 3.13588796,
       1.76745244, 1.85376745, 2.01831691, 2.5570339 , 1.99628158,
       1.20508176, 2.0371576 , 1.28954248, 2.55909546, 2.11154878,
       1.98678741, 2.72588213, 2.92745291])
```

Visualization of K-means Clustering

```
In [41]: #Scatter Plot for k-means clustering

plt.scatter(X[y_means == 0,0],X[y_means == 0,1],color='pink',label='Cluster 1')
plt.scatter(X[y_means == 1,0],X[y_means == 1,1],color='red',label='Cluster 2')
plt.scatter(X[y_means == 2,0],X[y_means == 2,1],color='green',label='Cluster 3')
plt.scatter(X[y_means == 3,0],X[y_means == 3,1],color='blue',label='Cluster 4')
plt.title('K-means Clustering')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend()
plt.grid(True)
plt.show()
```



Here we see that there are 4 distinct clusters formed by the data frame encoded using frequency encoding and reduced into two PCA components i.e., PCA1 and PCA2

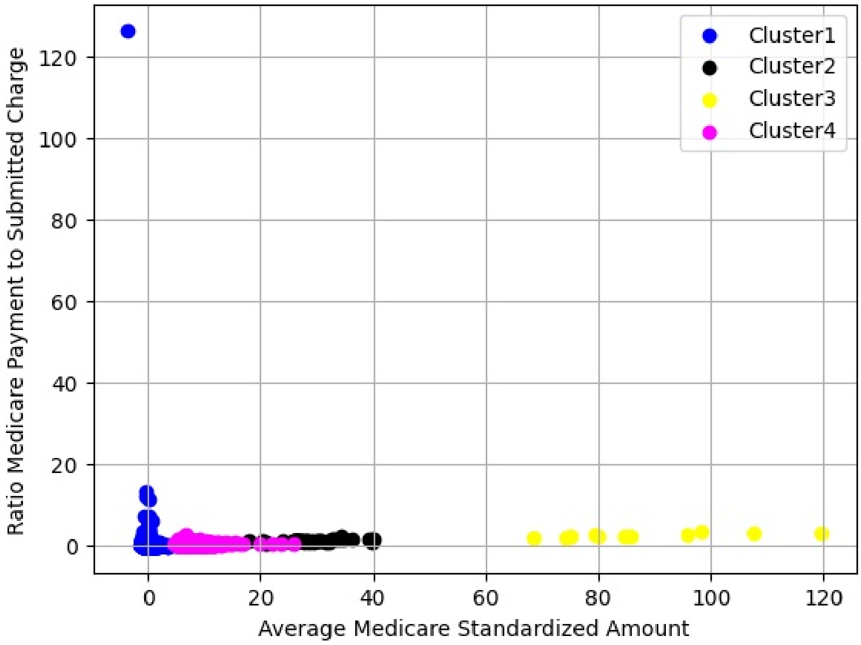
Scatter Plot of K-means Clusters (Different Columns)

In [42]:
```
X2 = df.iloc[:, [18, 19]].values

kmeans2 = KMeans(n_clusters = 4, init = "k-means++", random_state = 42)
y_kmeans2 = kmeans.fit_predict(X2)
```

In [43]:
```
df.columns
```

Out[43]:
```
Index(['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',
       'Provider Type', 'Medicare Participation Indicator', 'Place of Service',
       'HCPCS Code', 'HCPCS Description', 'HCPCS Drug Indicator',
       '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',
       'Ratio Medicare Payment to Submitted Charge'],
      dtype='object')
```

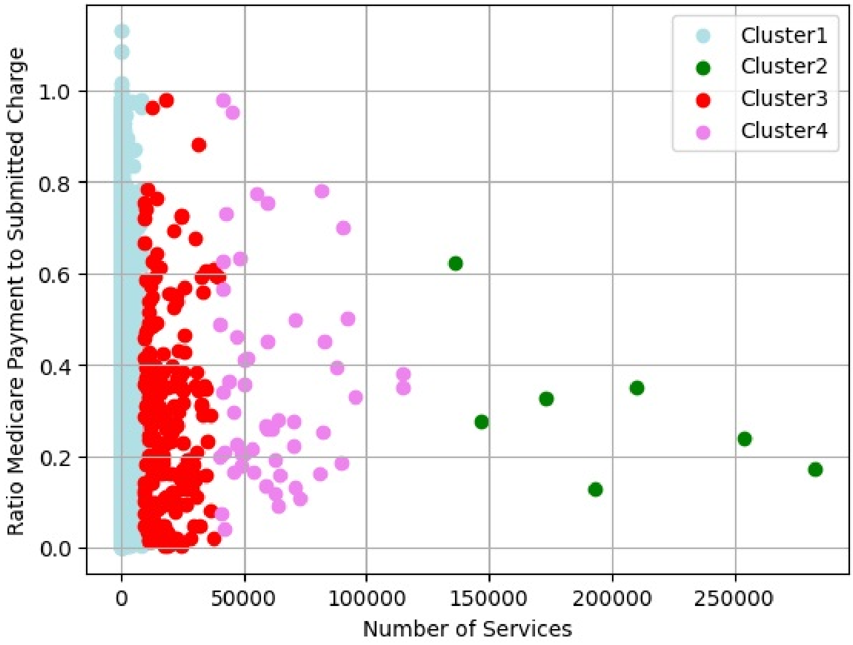
In [44]:
```
plt.scatter(X[y_kmeans2 == 0, 0], X[y_kmeans2 == 0, 1], c = 'blue', label = 'Cluster1')
plt.scatter(X[y_kmeans2 == 1, 0], X[y_kmeans2 == 1, 1], c = 'black', label = 'Cluster2')
plt.scatter(X[y_kmeans2 == 2, 0], X[y_kmeans2 == 2, 1], c = 'yellow', label = 'Cluster3')
plt.scatter(X[y_kmeans2 == 3, 0], X[y_kmeans2 == 3, 1], c = 'magenta', label = 'Cluster4')

plt.xlabel('Average Medicare Standardized Amount')
plt.ylabel('Ratio Medicare Payment to Submitted Charge')
plt.legend()
plt.grid(True)
plt.show()
```



Here, we can see that the Cluster1 , Cluster2, and Cluster4 are tightly packet while the Cluster3 is disperse.
Also, some parts of Cluster1 is very far away from the rest which may be an anomaly.

In [45]:

```
X3 = df.iloc[:, [12, 19]].values

kmeans3 = KMeans(n_clusters=4, init='k-means++', random_state=42)
y_kmeans3 = kmeans3.fit_predict(X3)

plt.scatter(X3[y_kmeans3 == 0, 0], X3[y_kmeans3 == 0, 1], c='powderblue', label='Cluster1')
plt.scatter(X3[y_kmeans3 == 1, 0], X3[y_kmeans3 == 1, 1], c='green', label='Cluster2')
plt.scatter(X3[y_kmeans3 == 2, 0], X3[y_kmeans3 == 2, 1], c='red', label='Cluster3')
plt.scatter(X3[y_kmeans3 == 3, 0], X3[y_kmeans3 == 3, 1], c='violet', label='Cluster4')

plt.xlabel('Number of Services')
plt.ylabel('Ratio Medicare Payment to Submitted Charge')
plt.legend()
plt.grid(True)
plt.show()
```

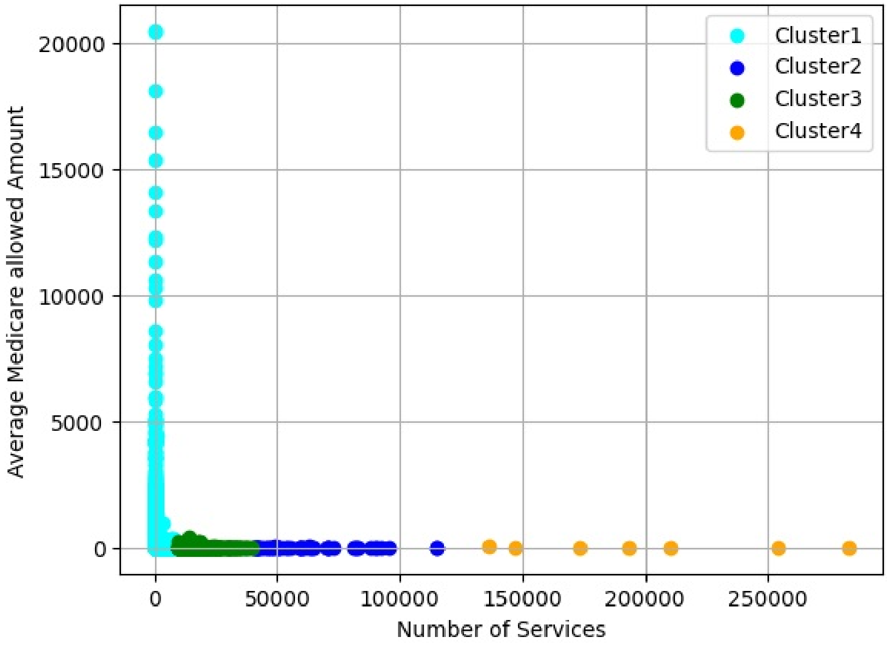


Here, we can see that the Cluster1 and Cluster3 are tightly packet while Cluster2 and Cluster4 are disperse.
The Cluster2 denoted by green is too widely spread along the later half of the plot.

Here, we can see that the Cluster1 and Cluster3 are tightly packet while Cluster2 and Cluster4 are disperse.
Cluster1 and Cluster3 lies within 50K range while other too are way beyond especially Cluster2.

In [46]:

```
X4 = df.iloc[:, [12, 15]].values

kmeans4 = KMeans(n_clusters=4, init='k-means++', random_state=42)
y_kmeans4 = kmeans4.fit_predict(X4)

plt.scatter(X4[y_kmeans4 == 0, 0], X4[y_kmeans4 == 0, 1], c='cyan', label='Cluster1')
plt.scatter(X4[y_kmeans4 == 1, 0], X4[y_kmeans4 == 1, 1], c='blue', label='Cluster2')
plt.scatter(X4[y_kmeans4 == 2, 0], X4[y_kmeans4 == 2, 1], c='green', label='Cluster3')
plt.scatter(X4[y_kmeans4 == 3, 0], X4[y_kmeans4 == 3, 1], c='orange', label='Cluster4')

plt.xlabel('Number of Services')
plt.ylabel('Average Medicare allowed Amount')
plt.legend()
plt.grid(True)
plt.show()
```



Here, we can see that Cluster2, Cluster3, and some parts of Cluster1 are tightly packet while the Cluster4 is too disperse. Also, some parts of Cluster1 among with Cluster4 are very far away from the rest which may be an anomaly.

DBSCAN Clustering

In [47]:

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.11, min_samples=500)
dbscan.fit(X)
```

Out[47]:

```
▾ DBSCAN
DBSCAN(eps=0.11, min_samples=500)
```

In [48]:

```
DBcluster = dbscan.labels_
DBcluster
```

Out[48]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [49]:

```
len(set(DBcluster))
```

Out[49]:

```
4
```

In [51]:

```
#Checking Silhouette Score
from sklearn.metrics import silhouette_score

silhouette_score(X, DBcluster)
```

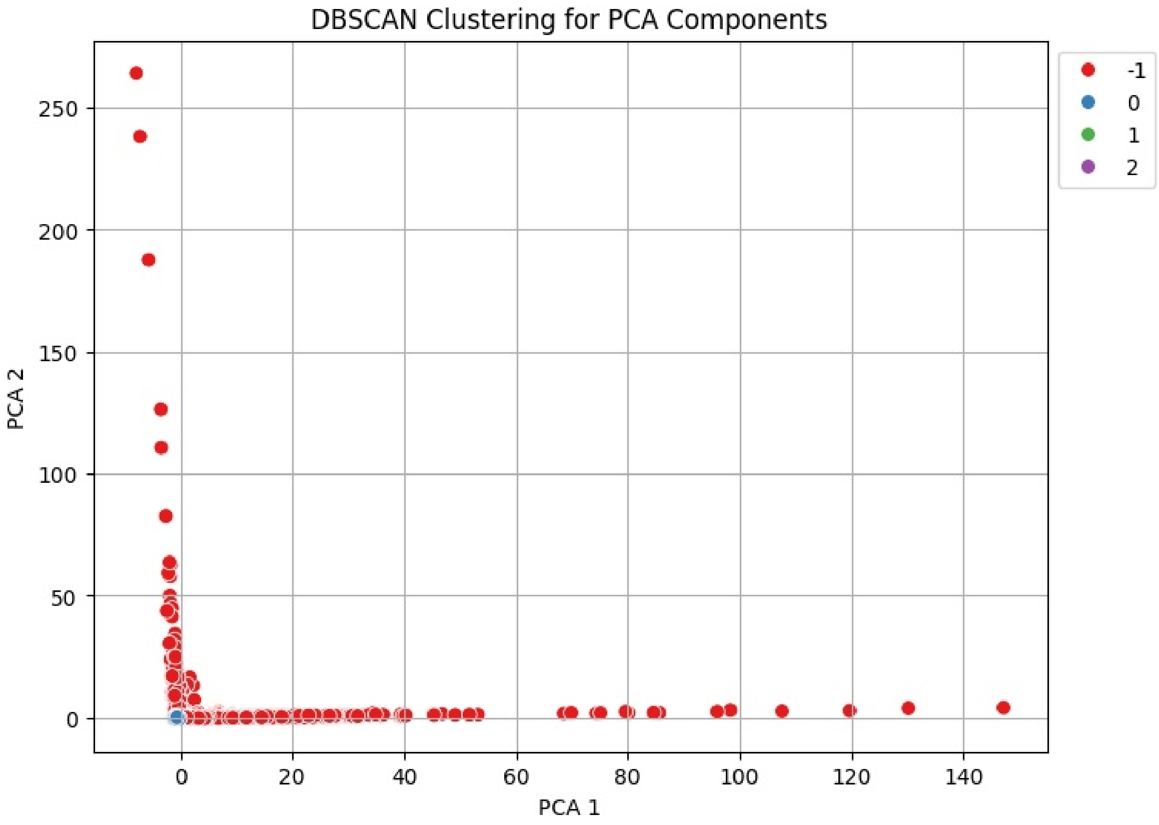
Out[51]:

```
0.1349982480639689
```

Visualizing DBSCAN Clusters

In [70]:

```
# Create a scatter plot of your data points, colored by DBSCAN cluster labels
plt.figure(figsize=(8, 6))

# Assign a different color to each cluster (including noise points labeled as -1)
sns.scatterplot(x=X[:, 0], y=X[:, 1], hue=dbscan.labels_, palette='Set1', s=50)
plt.title('DBSCAN Clustering for PCA Components')
plt.xlabel('PCA 1')
plt.ylabel('PCA 2')
plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
plt.grid(True)
plt.show()
```



The silhouette score and the scatter plot shows that the PCA components would not be good for the clustering, since the noise points are clearly observed in the almost all over the plot.

DBSCAN Cluster Scatter Plots for different columns

DBSCAN2 Cluster

('Average Medicare Standardized Amount') v/s ('Ratio Medicare Payment to Submitted Charge')

In [52]:

```
dbscan2 = DBSCAN(eps=0.5, min_samples=100)
dbscan2.fit(X2)
len(set(dbscan2.labels_))
```

Out[52]:

```
12
```

In [54]:

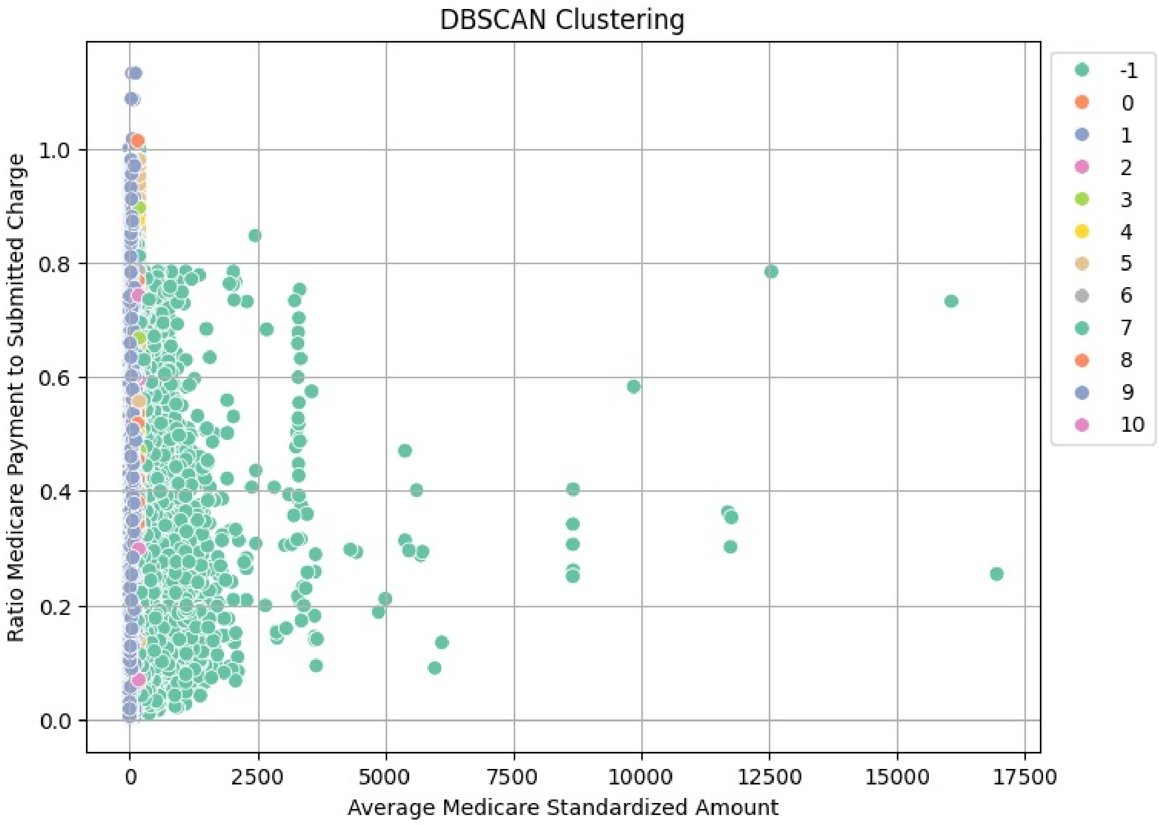
```
silhouette_score(X2, dbscan2.labels_)
```

Out[54]:

```
0.4479268672705822
```

In [71]:

```
# Create a scatter plot of your data points, colored by DBSCAN cluster labels
plt.figure(figsize=(8, 6))

# Assign a different color to each cluster (including noise points labeled as -1)
sns.scatterplot(x=X2[:, 0], y=X2[:, 1], hue=dbscan2.labels_, palette='Set2', s=50)
plt.title('DBSCAN Clustering')
plt.xlabel('Average Medicare Standardized Amount')
plt.ylabel('Ratio Medicare Payment to Submitted Charge')
plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
plt.grid(True)
plt.show()
```



DBSCAN3 Cluster

('Number of Services') v/s ('Ratio Medicare Payment to Submitted Charge')

In [76]:

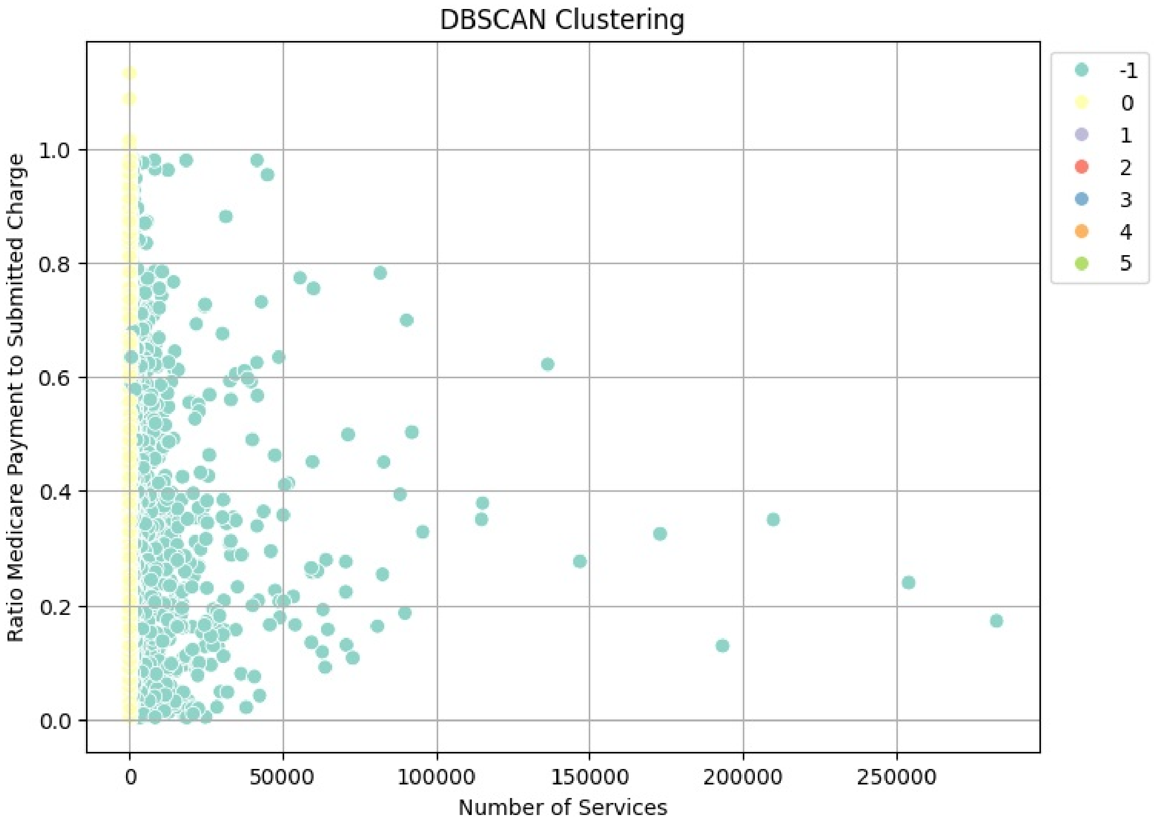
```
dbscan3 = DBSCAN(eps=2, min_samples=100)
dbscan3.fit(X3)

silhouette_score(X3, dbscan3.labels_)
```

Out[76]:

```
0.5489013902043637
```

In [77]:

```
# Create a scatter plot of your data points, colored by DBSCAN cluster labels
plt.figure(figsize=(8, 6))

# Assign a different color to each cluster (including noise points labeled as -1)
sns.scatterplot(x=X3[:, 0], y=X3[:, 1], hue=dbscan3.labels_, palette='Set3', s=50)
plt.title('DBSCAN Clustering')
plt.xlabel('Number of Services')
plt.ylabel('Ratio Medicare Payment to Submitted Charge')
plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
plt.grid(True)
plt.show()
```



This scatter plot has good silhouette score(0.5489013902043637) > 0.5
It formed 7 clusters among which the cluster (-1) denotes the noise points.

DBSCAN4 Cluster

('Number of Services') v/s ('Average Medicare allowed Amount')

In [78]:

```
dbscan4 = DBSCAN(eps=1.5, min_samples=100)
dbscan4.fit(X4)
```

Out[78]:

```
▾        DBSCAN
DBSCAN(eps=1.5, min_samples=100)
```

In [79]:

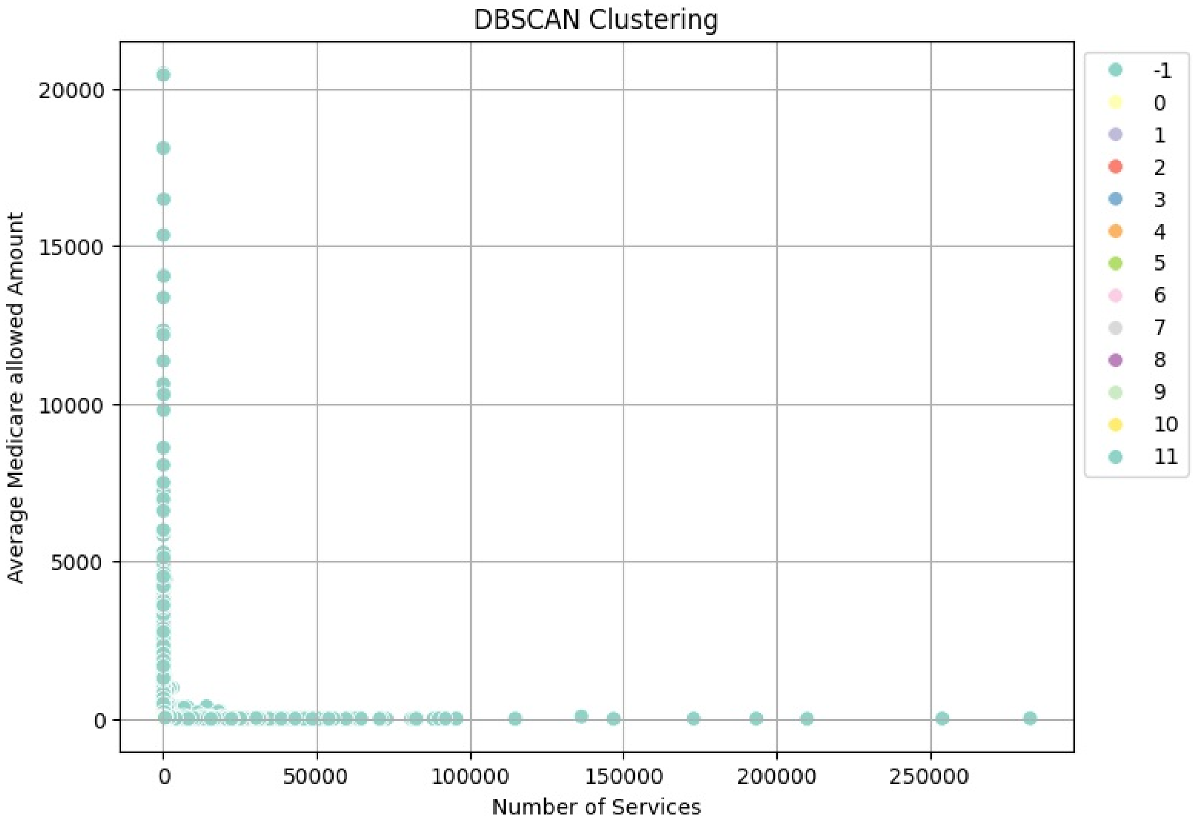
```
silhouette_score(X4, dbscan4.labels_)
```

Out[79]:

```
-0.5723500656206232
```

In [80]:

```
# Create a scatter plot of your data points, colored by DBSCAN cluster labels
plt.figure(figsize=(8, 6))

# Assign a different color to each cluster (including noise points labeled as -1)
sns.scatterplot(x=X4[:, 0], y=X4[:, 1], hue=dbscan4.labels_, palette='Set3', s=50)
plt.title('DBSCAN Clustering')
plt.xlabel('Number of Services')
plt.ylabel('Average Medicare allowed Amount')
plt.legend(bbox_to_anchor=(1, 1), loc='upper left')
plt.grid(True)
plt.show()
```



These columns { ('Number of Services') v/s ('Average Medicare allowed Amount') } aren't good for DBSCAN clustering as its silhouette score is negative(-0.5723500656206232), the noise points(-1) is also dominating all over the plot.

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js