In [73]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import GradientBoostingRegressor



# Load the dataset from CSV
# Adding dataset using desired file path
df = pd.read_csv('Cardiac_Surgery_and_Percutaneous_Coronary_Interventions

# Defining dictionary to map domain concepts to their corresponding featu
# For example, the domain concept 'Hospital Identifier' corresponds to fe
domain_concepts = {
    'Hospital Identifier': ['Facility ID', 'Hospital Name'],
    'Hospital Location': ['Detailed Region', 'Region'],
    'Medical Procedure': ['Procedure'],
    'Time Period': ['Year of Hospital Discharge'],
    'Hospital Procedure Data': ['Number of Cases'],
    'Hospital Outcome Data': ['Number of Deaths', 'Observed Mortality Rat
                              'Expected Mortality Rate', 'Risk_Adjusted_M
                              'Lower Limit of Confidence Interval', 'Uppe
    'Comparison of Mortality Rate': ['Comparison Results']
}

# Initializing an empty DataFrame for the Analytics Base Table (ABT)
# By creating a structured table that will be used for data analysis or m
# It allows us to gradually populate it with relevant information as we i
abt = pd.DataFrame()

# Checking each domian concept
for concept, features in domain_concepts.items():
    # Filtering out features that are not present in the dataset
    features = [col for col in features if col in df.columns]

    # Creating a subset of the dataset containing only the features relev
    subset = df[features]

    # Renaming the features to match the domain concept
    #This step is essential for maintaining consistency and clarity in th
    subset.columns = [concept if col != 'Procedure' else col for col in s

    # Concatenate the subset with the ABT aggregating and consolidating t
    abt = pd.concat([abt, subset], axis=1)

# Display the ABT
print(abt.head())
```

```
  Hospital Identifier     Hospital Identifier        Hospital Location  \
0                   1         Albany Med. Ctr         Capital District
1                1045   White Plains Hospital  NY Metro - New Rochelle
2                1438   Bellevue Hospital Ctr                Manhattan
3                1439     Beth Israel Med Ctr                Manhattan
4                1178  Bronx-Lebanon-Cncourse                    Bronx

        Hospital Location          Procedure Time Period  \
0         Capital District            All PCI        2016
1  NY Metro - New Rochelle  Non-Emergency PCI        2015
2           NY Metro - NYC            All PCI        2010
3           NY Metro - NYC            All PCI        2010
4           NY Metro - NYC            All PCI        2010

   Hospital Procedure Data  Hospital Outcome Data  Hospital Outcome Data
\
0                      680                     17                   2.50
1                      338                      1                   0.30
2                      448                      4                   0.89
3                     1762                     11                   0.62
4                       65                      4                   6.15

   Hospital Outcome Data  Hospital Outcome Data  Hospital Outcome Data  \
0                   1.52                   1.18                   3.26
1                   0.64                   0.00                   1.91
2                   0.89                   0.23                   2.16
3                   0.70                   0.38                   1.35
4                   2.41                   0.58                   5.50

            Comparison of Mortality Rate
0  Rate not different than Statewide Rate
1  Rate not different than Statewide Rate
2  Rate not different than Statewide Rate
3  Rate not different than Statewide Rate
4  Rate not different than Statewide Rate
```

Loading the Data

To start the code imports necessary libraries. It proceeds to load a dataset from a CSV file called 'Cardiac_Surgery_and_Percutaneous_Coronary_Interventions_by_Hospital___Beginning_ using the pd.read_csv() function.

Defining Key Concepts

Then a dictionary named domain_concepts is established to link domain concepts with their features in the dataset. Each domain concept represents a category of information such as 'Hospital Identifier' 'Hospital Location' 'Medical Procedure' among others.

The features linked with each domain concept are detailed as values in the dictionary.

Creating the Core Analytics Table (CAT)

1. A blank DataFrame named cat is set up to hold the Core Analytics Table data.
2. The code loops through each domain concept and its related features.

3. For every domain concept it extracts features present in the dataset and generates a subset of data containing only those features.
4. The feature names are adjusted to align with their domain concepts for clarity.
5. Lastly the subset is combined with CAT to merge and summarize information, from domain concepts into one comprehensive table view.

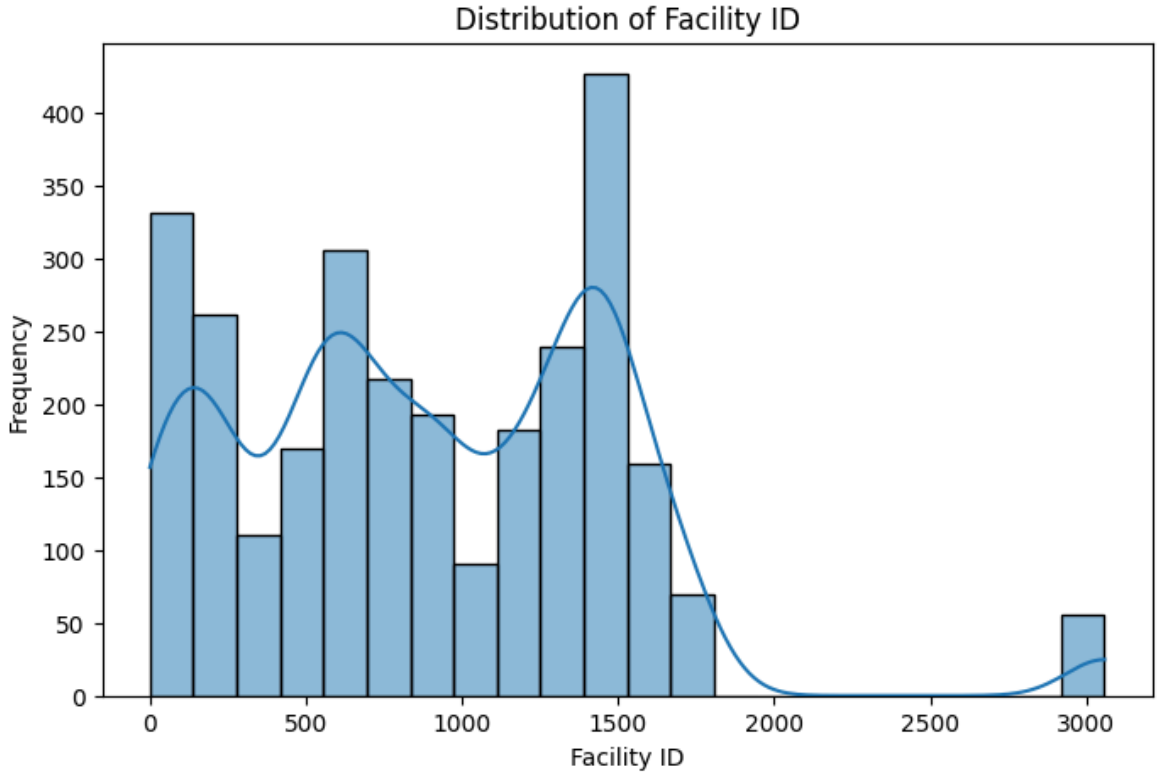
The output from the ABT is shown using the print() function to exhibit the rows of the table.

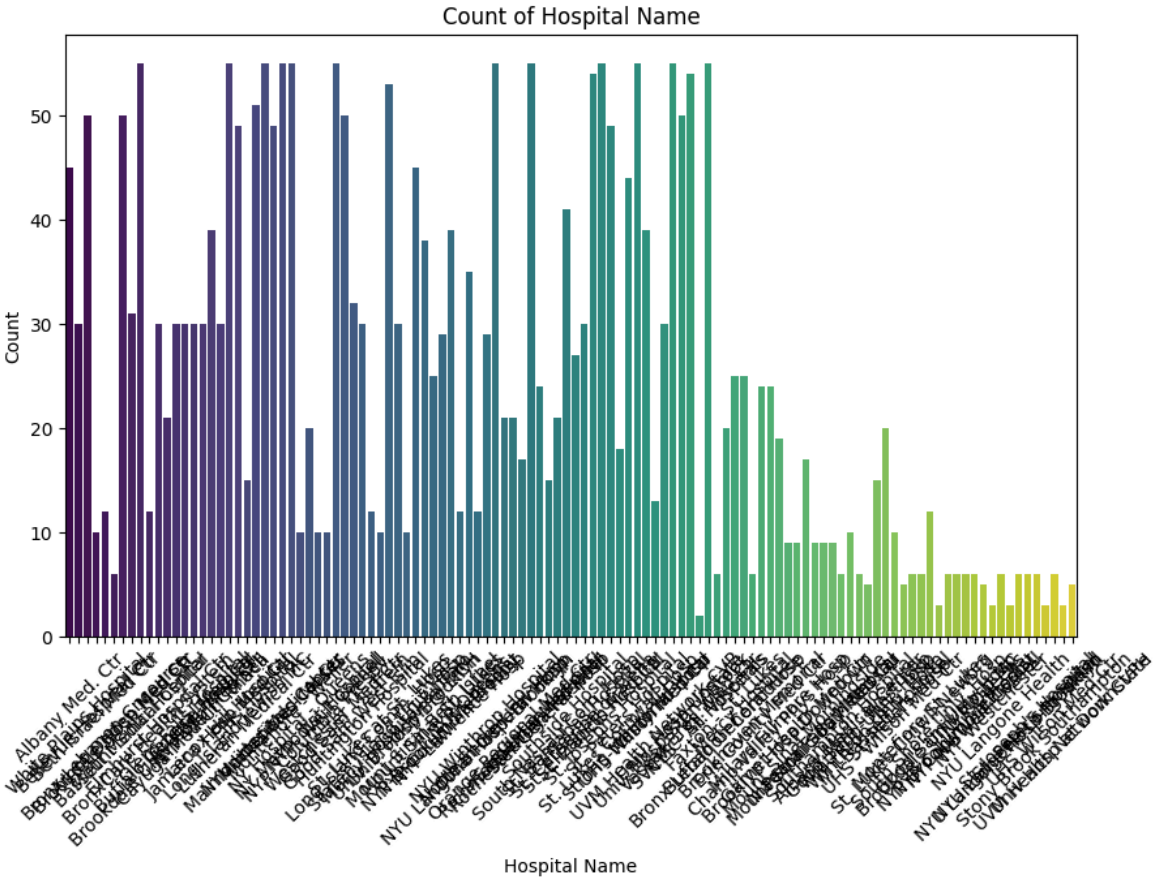
Interpreting the Results

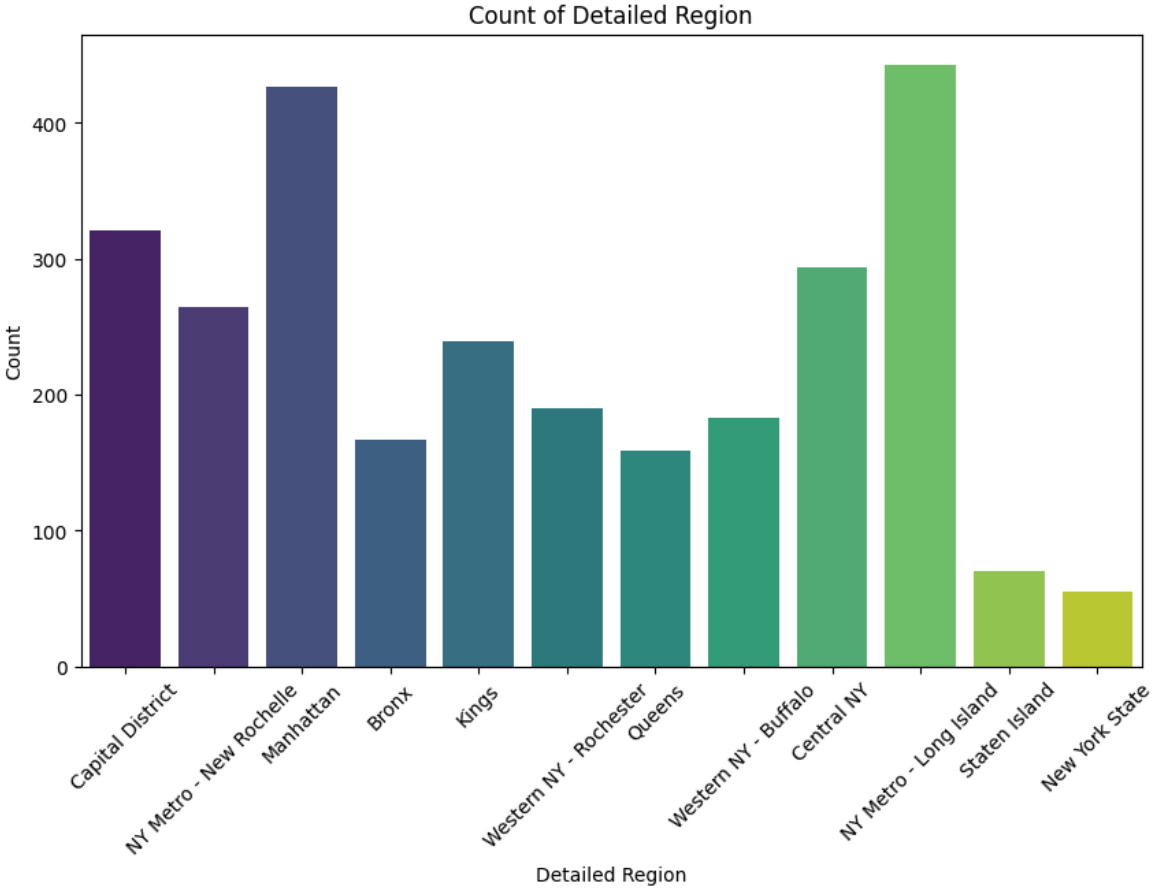
The ABT encompasses domain concepts like 'Hospital Identifier' 'Hospital Location' 'Medical Procedure' 'Time Period' 'Hospital Procedure Data' 'Hospital Outcome Data' and 'Comparison of Mortality Rate'.

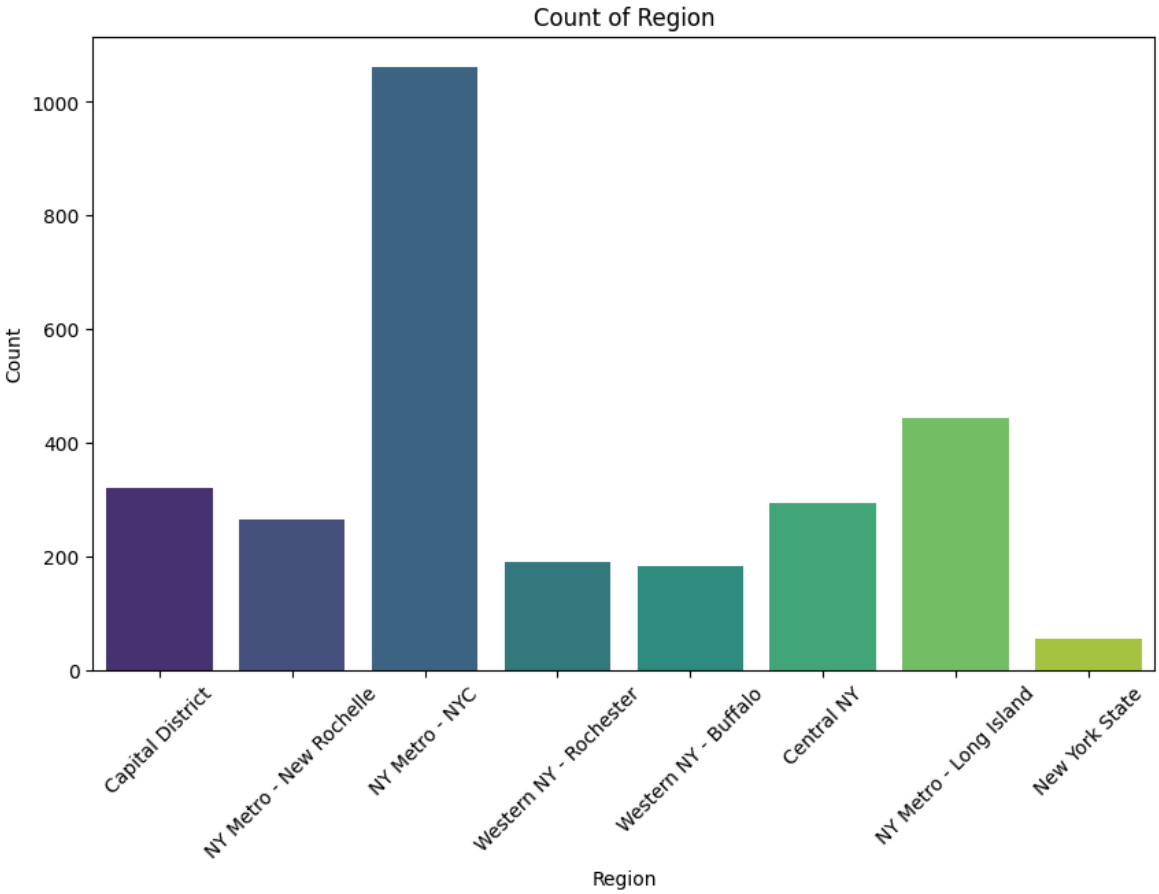
Each row in the ABT corresponds to a hospital procedure record with each column representing a domain concept or feature linked to it.

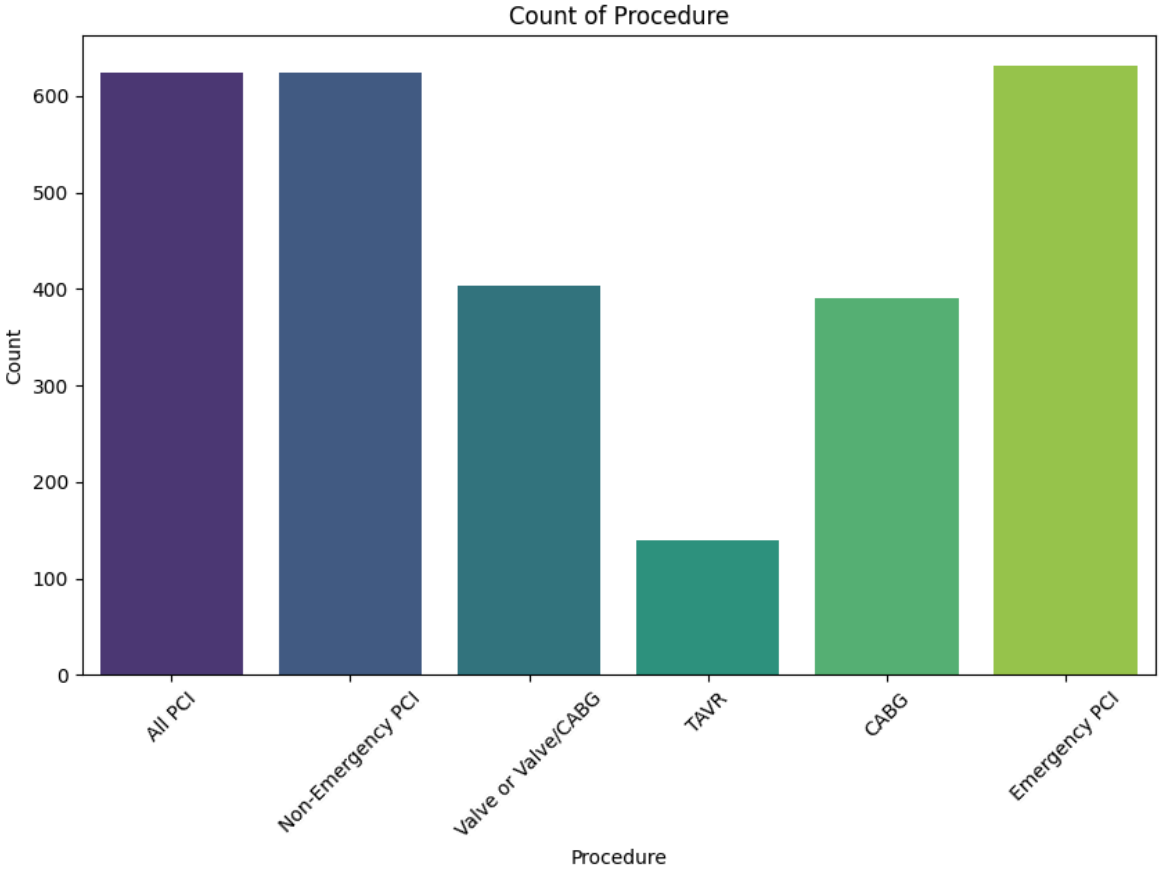
In [74]:
```
# Setting options to avoid truncation when displaying a dataframe
pd.set_option("display.max_rows", None)
pd.set_option("display.max_columns", None)
pd.set_option('display.float_format', '{:.2f}'.format)

# Tabular report for continuous features
continuous_report = df.describe()

# Tabular report for categorical features
categorical_report = df.describe(include='O')

# Data visualizations of values in each feature
for column in df.columns:
    # For continuous features
    if df[column].dtype in ['int64', 'float64']:
        plt.figure(figsize=(8, 5))
        sns.histplot(df[column].dropna(), kde=True)
        plt.title(f'Distribution of {column}')
        plt.xlabel(column)
        plt.ylabel('Frequency')
        plt.show()
    # For categorical features
    else:
        plt.figure(figsize=(10, 6))
        sns.countplot(x=column, data=df, palette='viridis')
        plt.title(f'Count of {column}')
        plt.xlabel(column)
        plt.ylabel('Count')
        plt.xticks(rotation=45)
        plt.show()

# Displaying tabular reports
print("Tabular Report for Continuous Features:")
print(continuous_report)

print("\nTabular Report for Categorical Features:")
print(categorical_report)
```

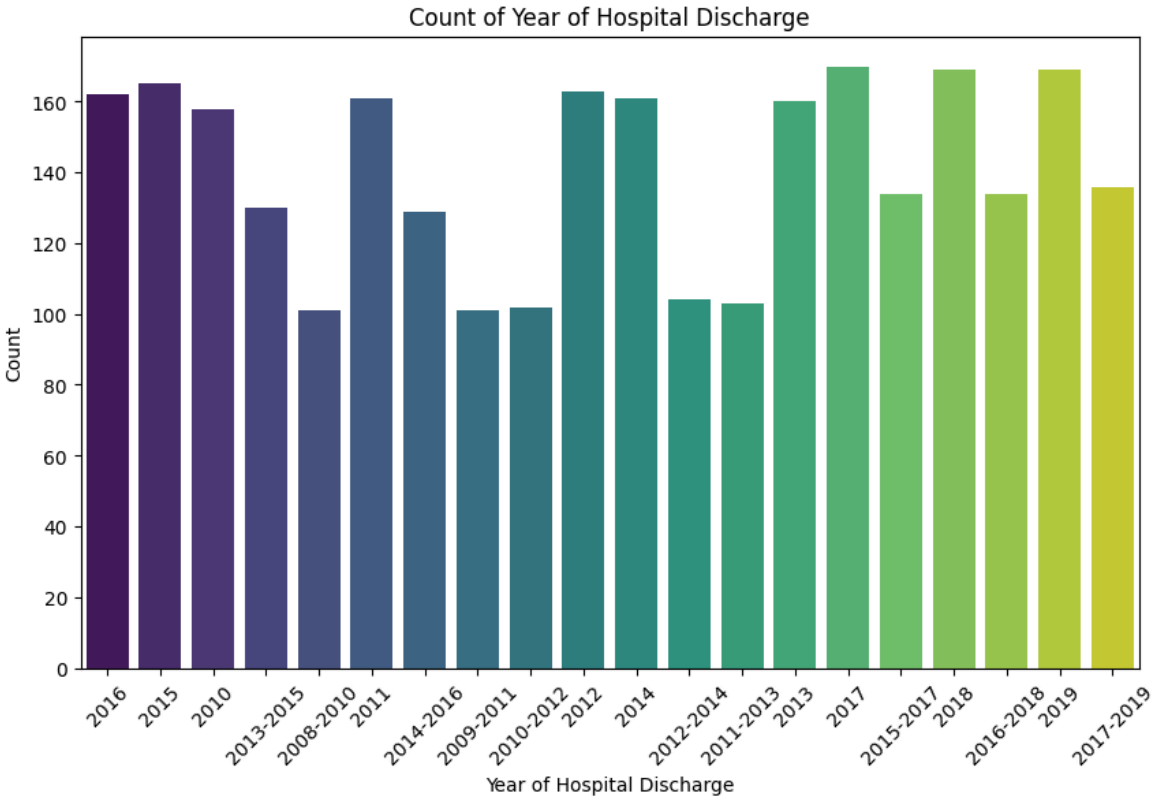

Distribution of Facility ID
Frequency
Facility ID
0
50
100
150
200
250
300
350
400
0
500
1000
1500
2000
2500
3000

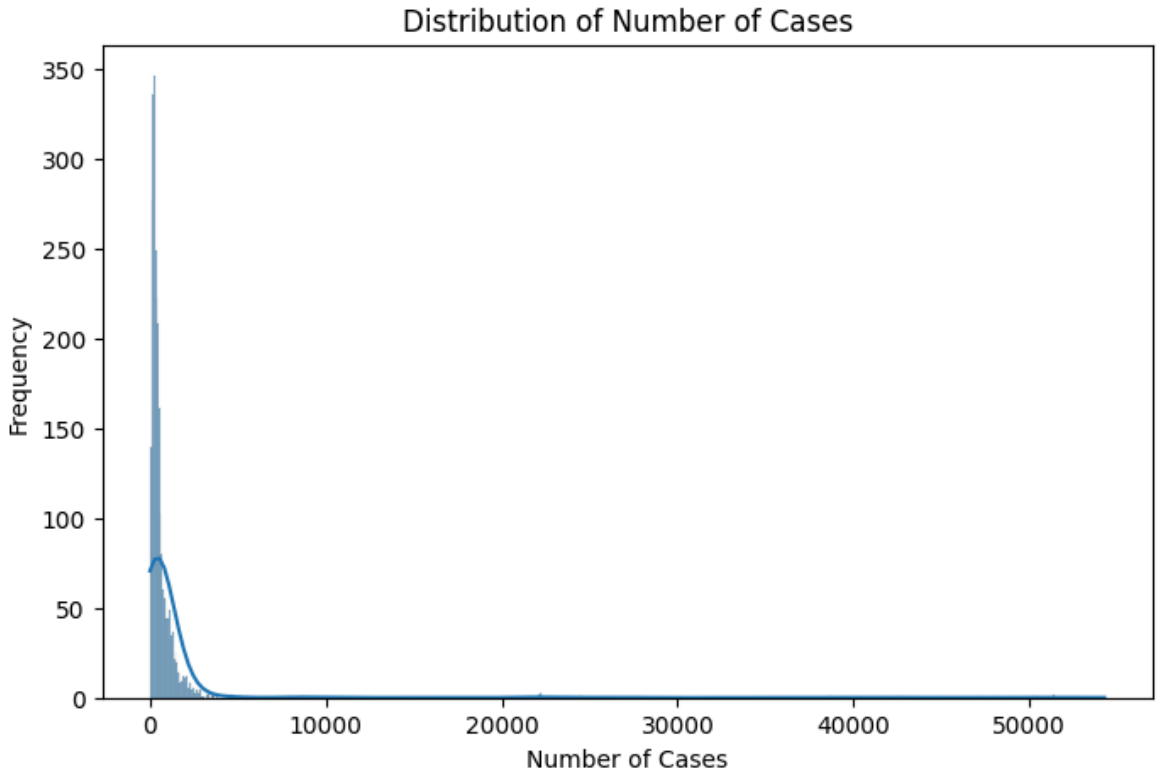

Count of Hospital Name
Count
Hospital Name
0
10
20
30
40
50

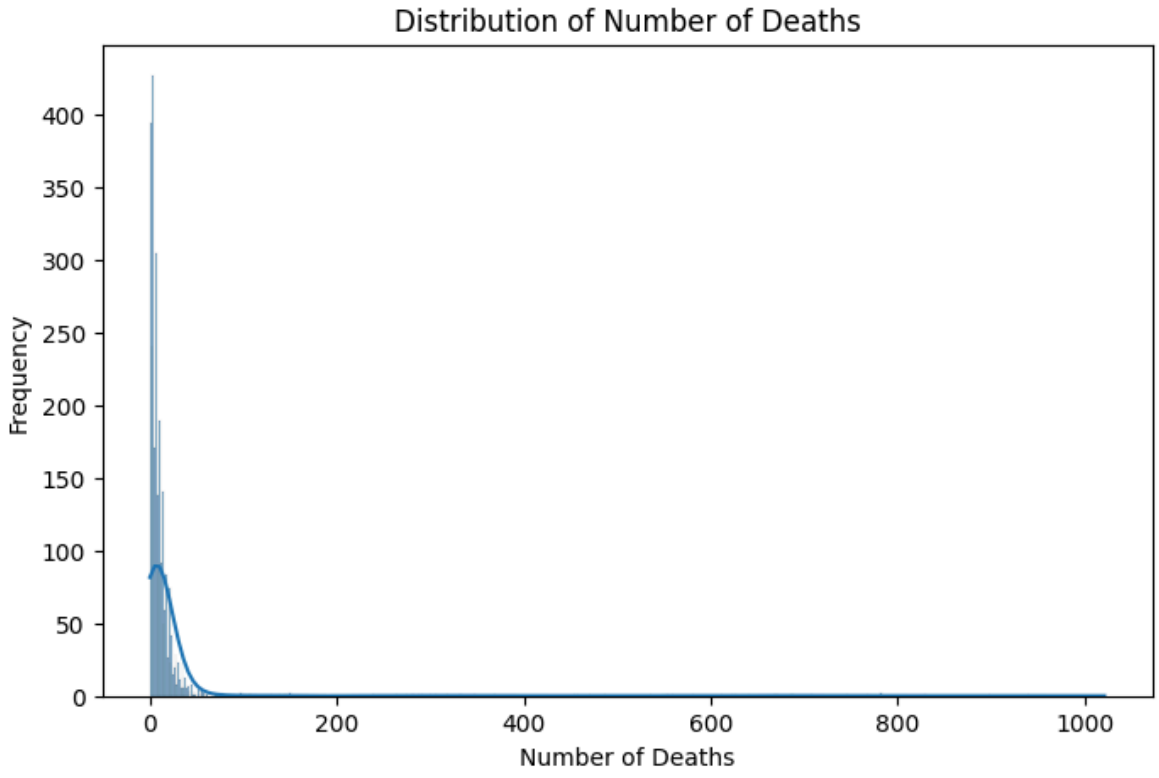

Count of Detailed Region
Count
0
100
200
300
400
Capital District
NY Metro - New Rochelle
Manhattan
Bronx
Kings
Western NY - Rochester
Queens
Western NY - Buffalo
Central NY
NY Metro - Long Island
Staten Island
New York State
Detailed Region

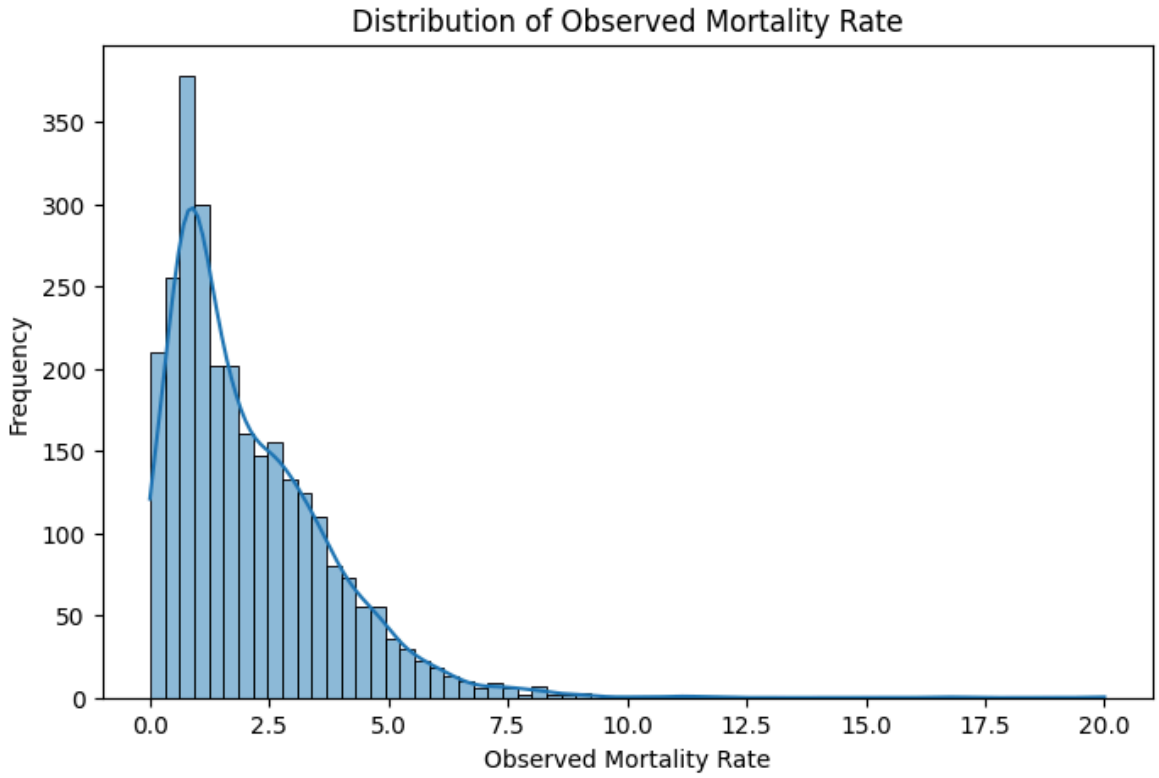

Count of Region
Count
0
200
400
600
800
1000
Capital District
NY Metro - New Rochelle
NY Metro - NYC
Western NY - Rochester
Western NY - Buffalo
Central NY
NY Metro - Long Island
New York State
Region

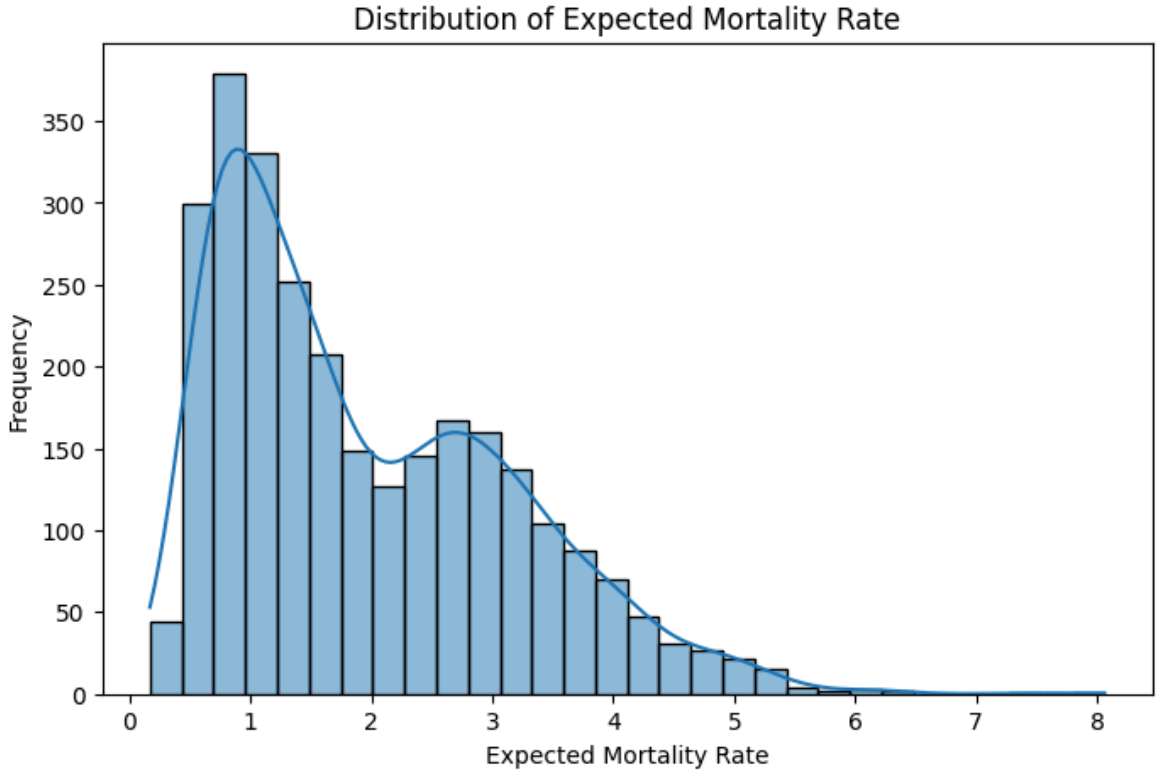

Count of Procedure
Count
0
100
200
300
400
500
600
All PCI
Non-Emergency PCI
Valve or Valve/CABG
TAVR
CABG
Emergency PCI
Procedure

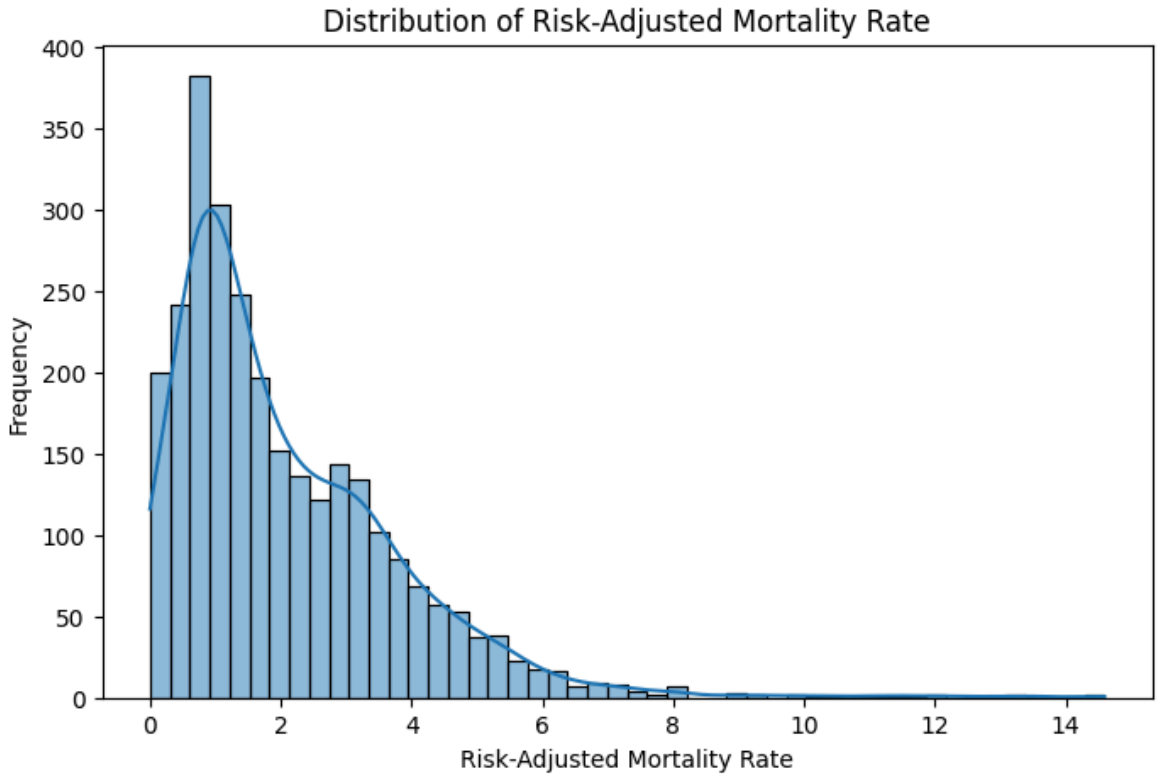

Count of Year of Hospital Discharge
Count
0
20
40
60
80
100
120
140
160
2016
2015
2010
2013-2015
2008-2010
2011
2014-2016
2009-2011
2010-2012
2012
2014
2012-2014
2011-2013
2013
2017
2015-2017
2018
2016-2018
2019
2017-2019
Year of Hospital Discharge

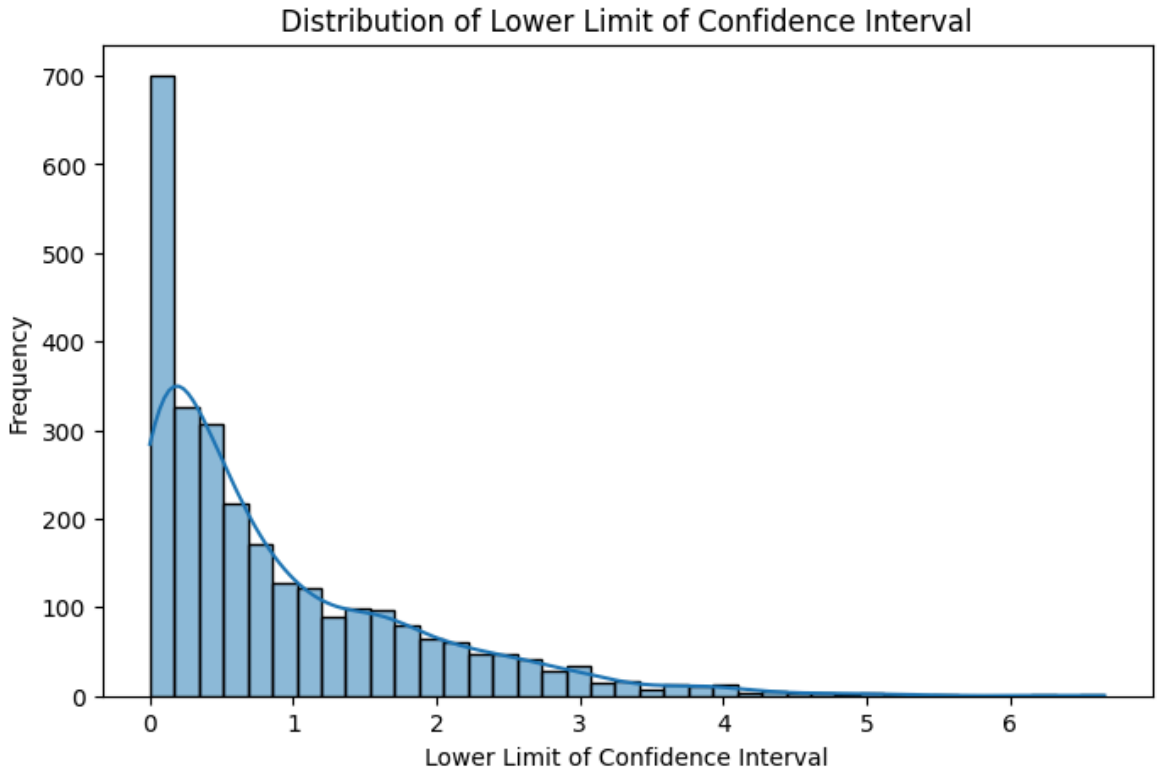

Distribution of Number of Cases
Frequency
Number of Cases
0
50
100
150
200
250
300
350
0
10000
20000
30000
40000
50000

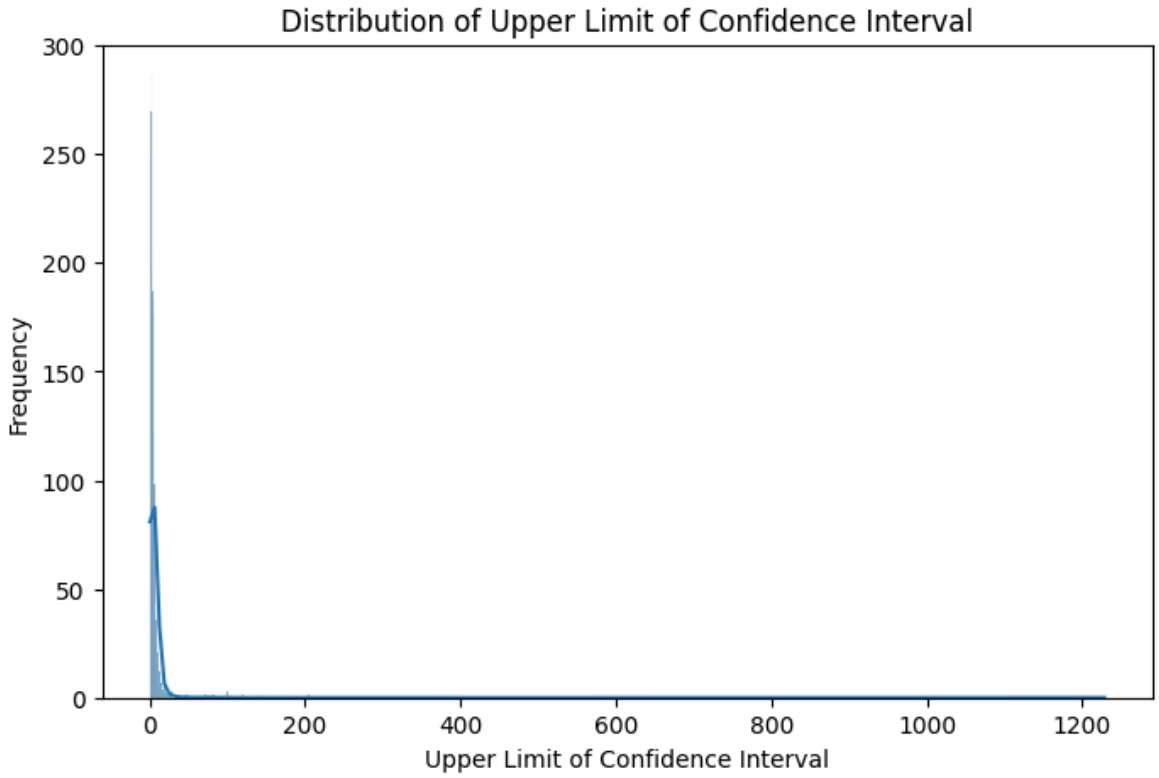

Distribution of Number of Deaths
Frequency
Number of Deaths
0
50
100
150
200
250
300
350
400
0
200
400
600
800
1000

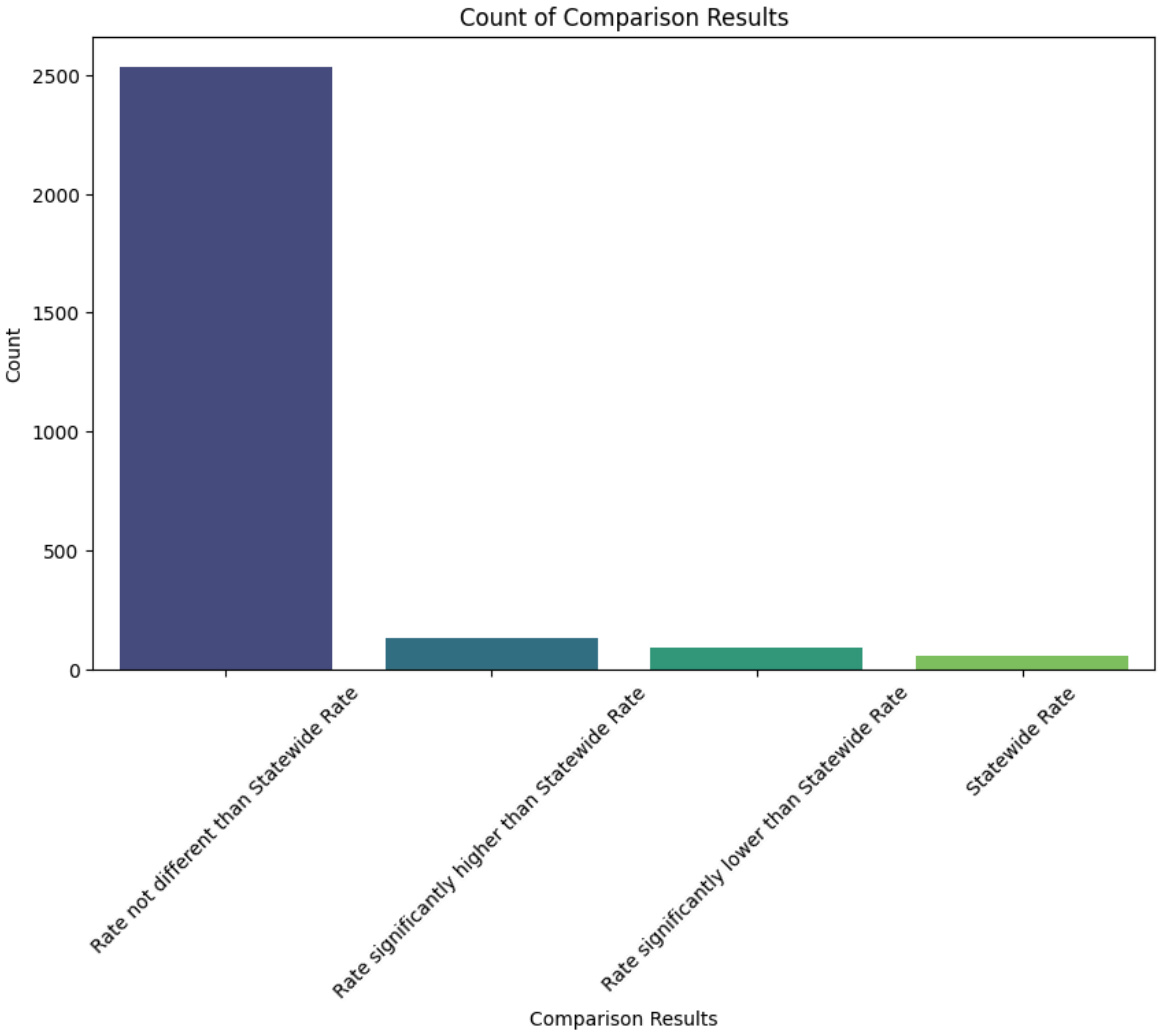

Distribution of Observed Mortality Rate
Frequency
Observed Mortality Rate
0
50
100
150
200
250
300
350
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

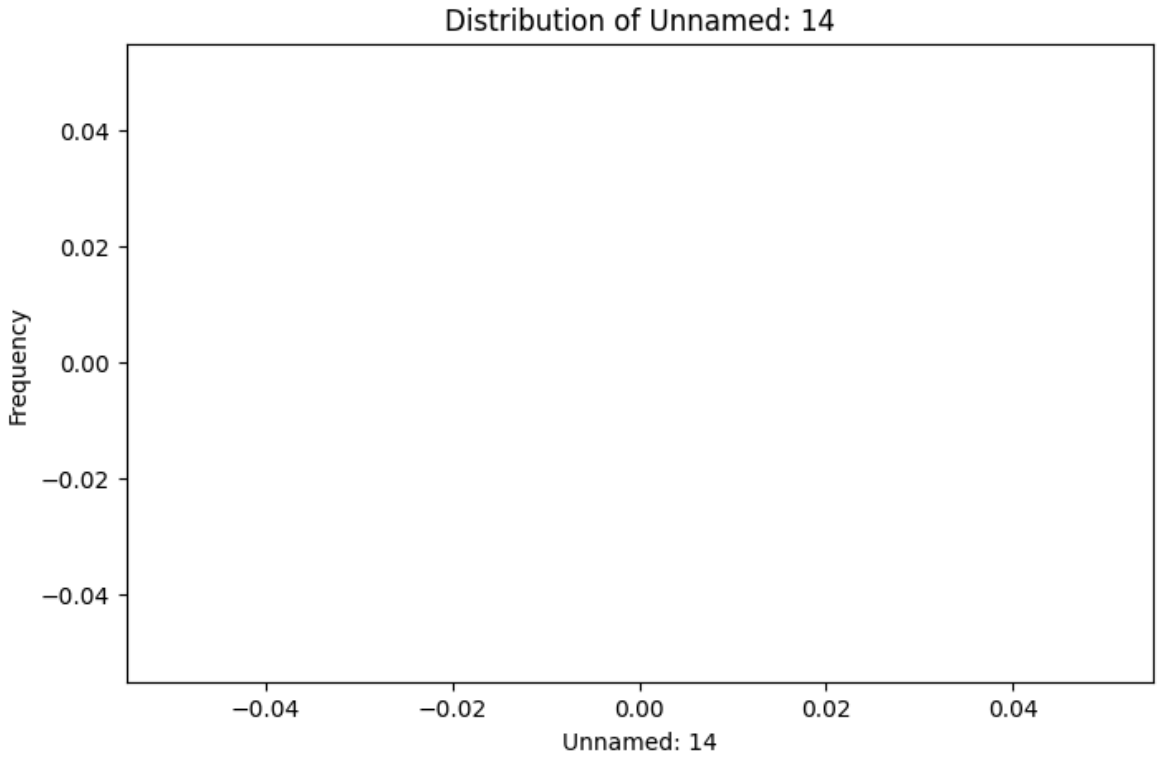

Distribution of Expected Mortality Rate
Frequency
Expected Mortality Rate
0
50
100
150
200
250
300
350
0
1
2
3
4
5
6
7
8

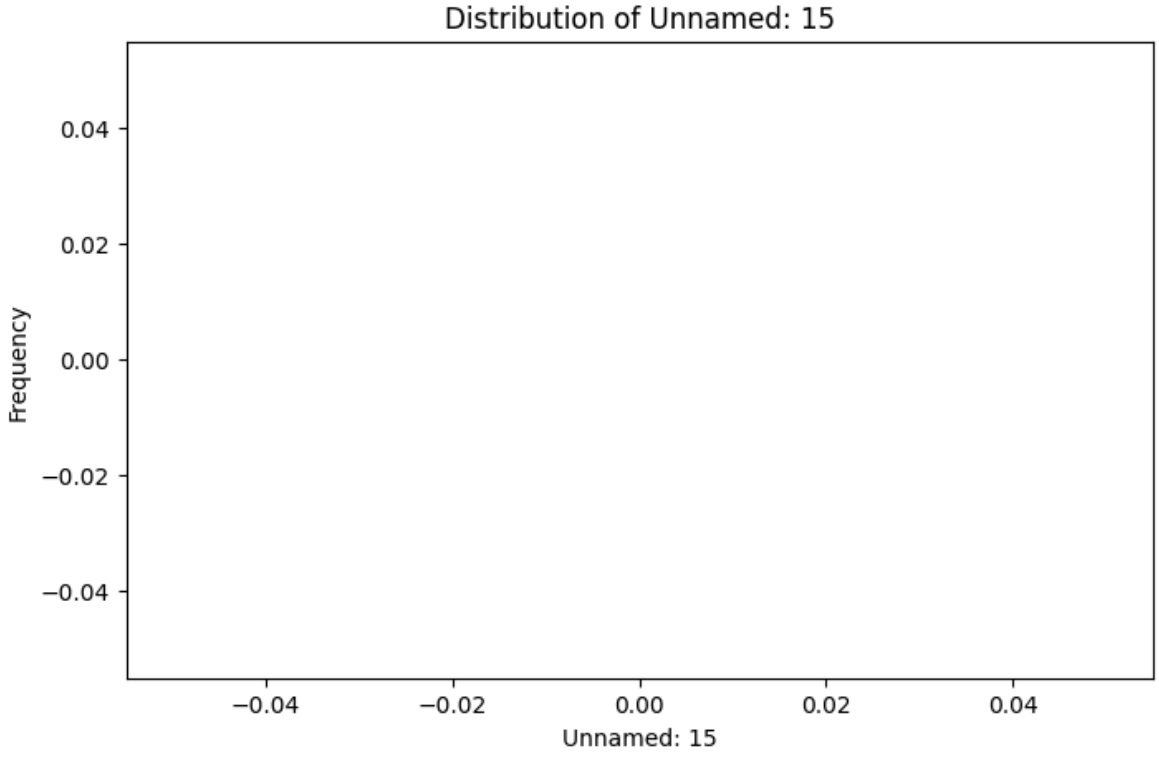

Distribution of Risk-Adjusted Mortality Rate
Frequency
Risk-Adjusted Mortality Rate
0
50
100
150
200
250
300
350
400
0
2
4
6
8
10
12
14

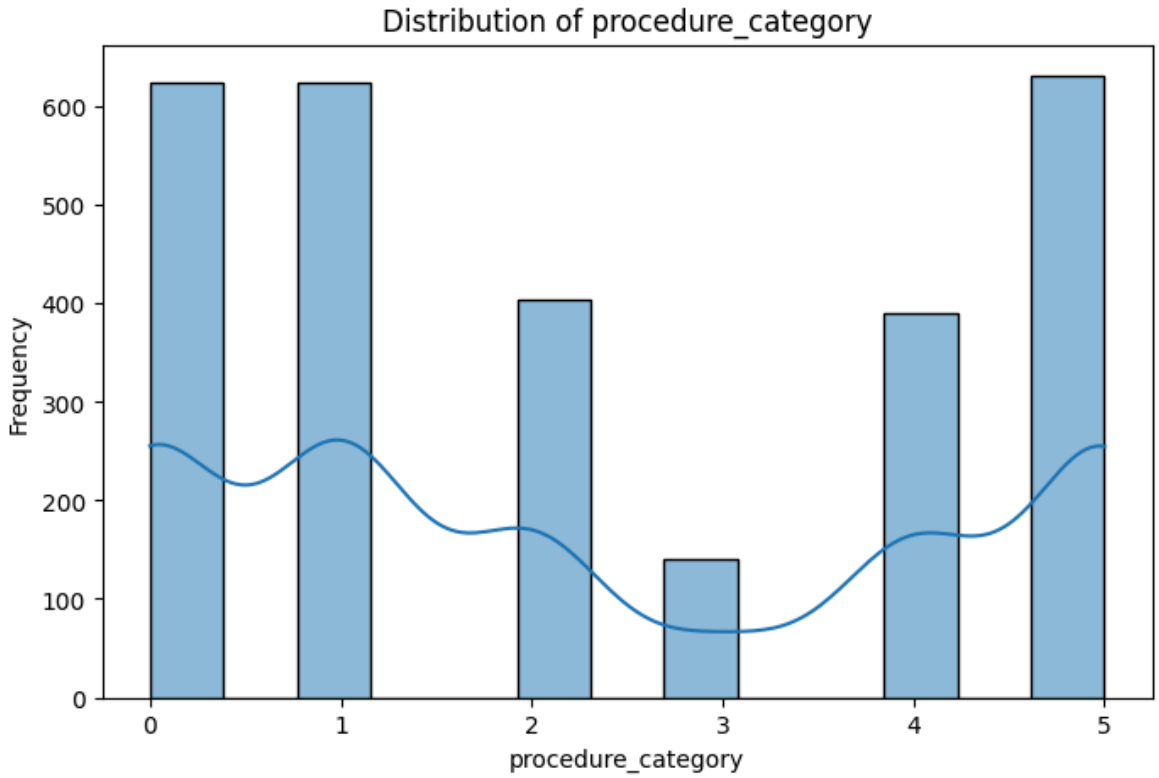

Distribution of Lower Limit of Confidence Interval
Frequency
Lower Limit of Confidence Interval
0
100
200
300
400
500
600
700
0
1
2
3
4
5
6


Distribution of Upper Limit of Confidence Interval
Frequency
Upper Limit of Confidence Interval
0
50
100
150
200
250
300
0
200
400
600
800
1000
1200


Count of Comparison Results
Count
Comparison Results
0
500
1000
1500
2000
2500
Rate not different than Statewide Rate
Rate significantly higher than Statewide Rate
Rate significantly lower than Statewide Rate
Statewide Rate


Distribution of Unnamed: 14
Frequency
0.04
0.02
0.00
−0.02
−0.04
−0.04
−0.02
0.00
0.02
0.04
Unnamed: 14


Distribution of Unnamed: 15
Frequency
0.04
0.02
0.00
−0.02
−0.04
−0.04
−0.02
0.00
0.02
0.04
Unnamed: 15


Distribution of procedure_category
Frequency
0
100
200
300
400
500
600
0
1
2
3
4
5
procedure_category

Tabular Report for Continuous Features:

	Facility ID	Number of Cases	Number of Deaths
count	2812.00	2812.00	2812.00
mean	897.57	1080.07	18.60
std	602.38	4359.57	74.35
min	0.00	1.00	0.00
25%	456.50	202.00	3.00
50%	885.00	375.00	7.00
75%	1438.00	702.25	13.00
max	3058.00	54276.00	1021.00

	Observed Mortality Rate	Expected Mortality Rate
count	2812.00	2812.00
mean	2.07	1.95
std	1.70	1.20
min	0.00	0.17
25%	0.81	0.95
50%	1.60	1.61
75%	2.99	2.81
max	20.00	8.06

	Risk-Adjusted Mortality Rate	Lower Limit of Confidence Interval
count	2812.00	2757.00
mean	2.08	0.90
std	1.75	0.97
min	0.00	0.00
25%	0.84	0.17
50%	1.56	0.54
75%	3.00	1.37
max	14.59	6.66

	Upper Limit of Confidence Interval	Unnamed: 14	Unnamed: 15
count	2757.00	0.00	0.00
mean	5.93	NaN	NaN
std	24.89	NaN	NaN
min	0.58	NaN	NaN
25%	2.30	NaN	NaN
50%	3.98	NaN	NaN
75%	6.00	NaN	NaN
max	1228.42	NaN	NaN

	procedure_category
count	2812.00
mean	2.33
std	1.90
min	0.00
25%	1.00
50%	2.00
75%	4.00
max	5.00

Tabular Report for Categorical Features:

	Hospital Name	Detailed Region	Region
count	2812	2812	2812
unique	114	12	8
top	North Shore Univ Hosp	NY Metro - Long Island	NY Metro - NYC
freq	55	443	1062

	Procedure Year of Hospital Discharge
count	2812

```
unique              6                          20
top     Emergency PCI                        2017
freq              631                         170

                            Comparison Results
count                                       2812
unique                                         4
top     Rate not different than Statewide Rate
freq                                        2535
```

Analyzing Data Summarization and Visualization

Data Summarization

1. To start the code is set up to display data in its entirety ensuring that all rows and columns are visible. Next it generates reports for both continuous and categorical features using the describe() function.
2. For features it calculates and presents statistics like count mean standard deviation, minimum value, 25th percentile, median (50th percentile) 75th percentile and maximum value.
3. Regarding features it provides information on unique counts, top occurrences and frequencies.

Data Visualization

1. Following the creation of reports the code then moves on to visually represent the data by utilizing histograms for continuous features and count plots for categorical features.
2. Histograms present the distribution of values for features by showing how data is distributed across different ranges.
3. Count plots illustrate the frequency of each category for features to shed light on how these variables are distributed within the dataset.

Interpretation

Continuous Features: By examining the tabular report details such as values and percentiles for attributes like 'Number of Cases 'Number of Deaths 'Observed Mortality Rate' etc. we can gain insights, into both central tendencies and spread of numerical data.When looking at features the tabular report gives details on unique categories the most common category and how often it appears. This helps us see how different categories are spread out within the variables.

For data visualization histograms show us the distribution of features pointing out patterns like skewness and central tendency. Count plots for features display how often each category appears, helping us spot dominant categories and understand their distribution in different groups.

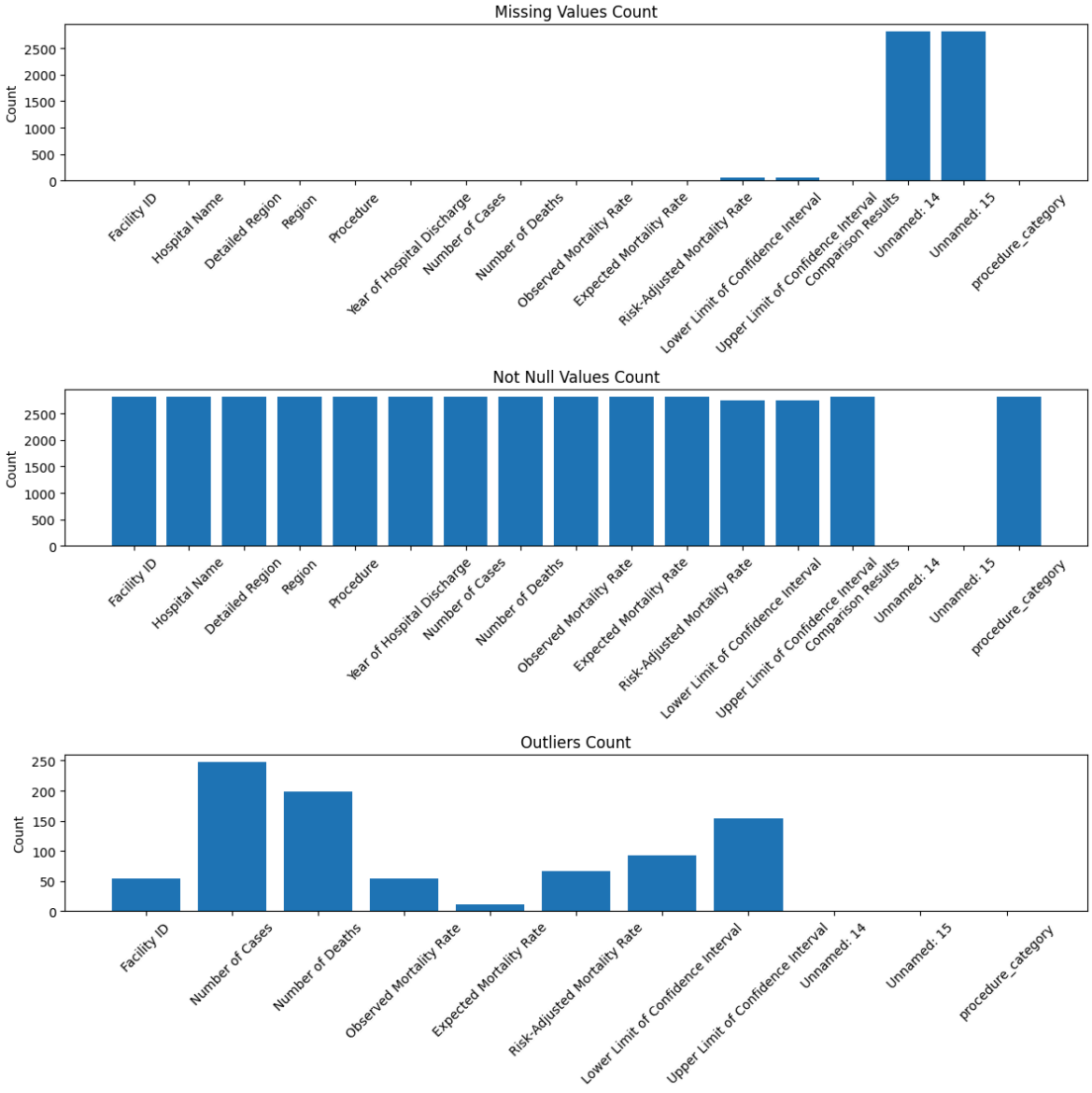
Key Points

The data seems evenly spread out across categories and ranges for both continuous and categorical features. There don't seem to be any outliers or irregularities in the histograms and count plots. Understanding how features are distributed and summarized is vital, for data analysis, modeling and decision making processes.

In [55]:
```
# Counting missing values
missing_values = df.isnull()
print("Count of missing values:\n", missing_values.sum())

# Count of not null values for each column
not_null_values = {col: len(df[col].dropna()) for col in df.columns}
print("\nCount of not null values for each column:\n", not_null_values)

# Identifying outliers
numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
outliers = {}
lower_bound = {}
upper_bound = {}

for col in numeric_columns:
    q1 = df[col].quantile(0.25)
    q3 = df[col].quantile(0.75)
    iqr = q3 - q1
    lower_bound[col] = q1 - 1.5 * iqr
    upper_bound[col] = q3 + 1.5 * iqr
    outliers[col] = df[col][(df[col] < lower_bound[col]) | (df[col] > upp

print("\nCount of outliers for each numeric column:\n", {col: len(values)

# Detecting duplicate rows
duplicate_rows = df[df.duplicated()]
print("\nCount of duplicate rows:", len(duplicate_rows))

# Inconsistent data formats
inconsistent_formats = df.applymap(lambda x: isinstance(x, str))
print("\nCount of inconsistent data formats:\n", inconsistent_formats.sum

# Incomplete or inconsistent data entries
incomplete_data_entries = df[df.isnull().any(axis=1)]
print("\nCount of incomplete data entries:", len(incomplete_data_entries)


# Plotting the counts
fig, axes = plt.subplots(3, 1, figsize=(12, 12))

# Plot missing values count
axes[0].bar(missing_values.sum().index, missing_values.sum().values)
axes[0].set_title('Missing Values Count')
axes[0].set_ylabel('Count')
axes[0].tick_params(axis='x', rotation=45)

# Plot not null values count
axes[1].bar(not_null_values.keys(), not_null_values.values())
axes[1].set_title('Not Null Values Count')
axes[1].set_ylabel('Count')
axes[1].tick_params(axis='x', rotation=45)

# Plot outliers count
```

```
axes[2].bar(outliers.keys(), [len(values) for values in outliers.values()])
axes[2].set_title('Outliers Count')
axes[2].set_ylabel('Count')
axes[2].tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.show()
```

```
Count of missing values:
 Facility ID                              0
Hospital Name                             0
Detailed Region                           0
Region                                    0
Procedure                                 0
Year of Hospital Discharge                0
Number of Cases                           0
Number of Deaths                          0
Observed Mortality Rate                   0
Expected Mortality Rate                   0
Risk-Adjusted Mortality Rate              0
Lower Limit of Confidence Interval       55
Upper Limit of Confidence Interval       55
Comparison Results                        0
Unnamed: 14                            2812
Unnamed: 15                            2812
procedure_category                        0
dtype: int64

Count of not null values for each column:
 {'Facility ID': 2812, 'Hospital Name': 2812, 'Detailed Region': 2812, 'Re
gion': 2812, 'Procedure': 2812, 'Year of Hospital Discharge': 2812, 'Numbe
r of Cases': 2812, 'Number of Deaths': 2812, 'Observed Mortality Rate': 28
12, 'Expected Mortality Rate': 2812, 'Risk-Adjusted Mortality Rate': 2812,
'Lower Limit of Confidence Interval': 2757, 'Upper Limit of Confidence Int
erval': 2757, 'Comparison Results': 2812, 'Unnamed: 14': 0, 'Unnamed: 15':
0, 'procedure_category': 2812}

Count of outliers for each numeric column:
 {'Facility ID': 55, 'Number of Cases': 247, 'Number of Deaths': 199, 'Obs
erved Mortality Rate': 55, 'Expected Mortality Rate': 11, 'Risk-Adjusted M
ortality Rate': 66, 'Lower Limit of Confidence Interval': 93, 'Upper Limit
of Confidence Interval': 154, 'Unnamed: 14': 0, 'Unnamed: 15': 0, 'procedu
re_category': 0}

Count of duplicate rows: 0

Count of inconsistent data formats:
 16872

Count of incomplete data entries: 2812
```



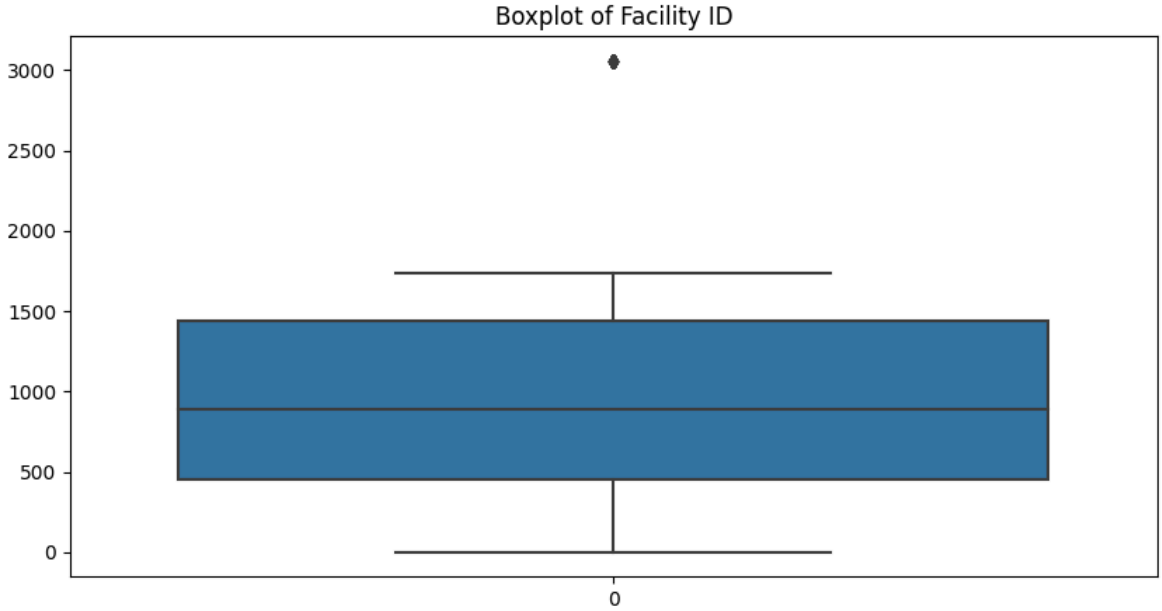
In [56]:
```
# 1. Imputation: Missing Values
# Numeric columns: Impute with the mean
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns
for col in numeric_cols:
    df[col].fillna(df[col].mean(), inplace=True)

# Categorical columns: Impute with the mode
categorical_cols = df.select_dtypes(include=['object']).columns
for col in categorical_cols:
    df[col].fillna(df[col].mode()[0], inplace=True)

# 2. Duplicate Rows: dropping
df.drop_duplicates(inplace=True)

# 3. Inconsistent Data Formats: Standardize data formats
for col in categorical_cols:
    df[col] = df[col].str.strip()

# Removing the fully empty columns only if they exist
columns_to_drop = ['Unnamed: 14', 'Unnamed: 15']
df = df.drop(columns=[col for col in columns_to_drop if col in df.columns


# Year Extraction
```

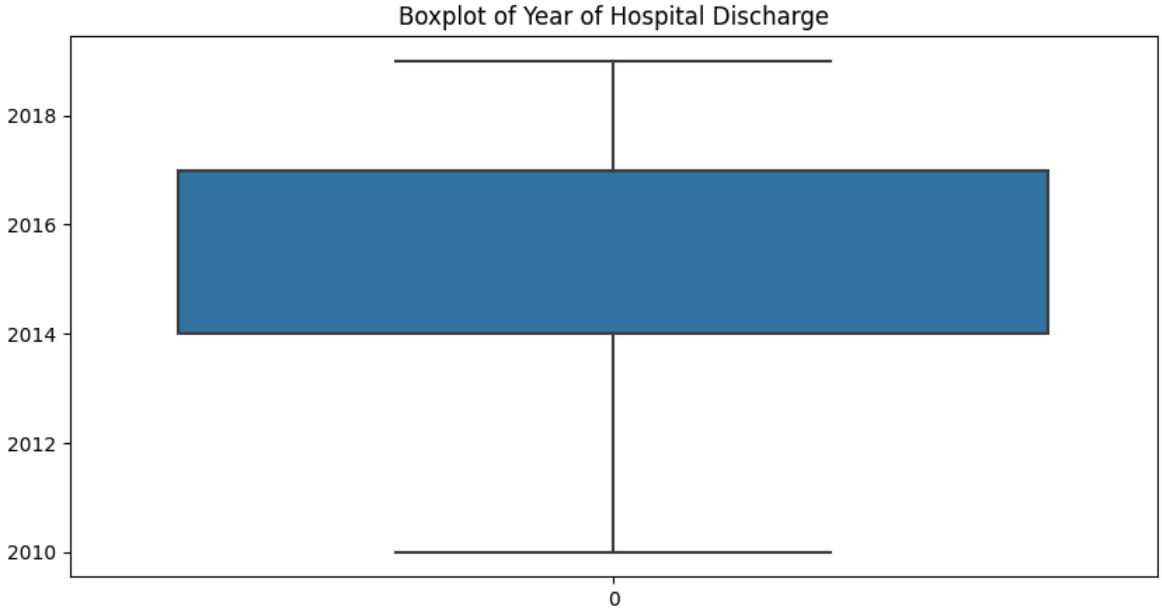
```
def extract_year(date_str):
    # Handle the conversion to year if the format is correct, otherwise r
    try:
        return pd.to_datetime(date_str, format='%Y', errors='raise').year
    except ValueError:
        return np.nan

# Applying the function to the 'Year of Hospital Discharge' column
df['Year of Hospital Discharge'] = df['Year of Hospital Discharge'].apply

# Handling possible NaT values after conversion
df['Year of Hospital Discharge'].fillna(df['Year of Hospital Discharge'].

# Saving the cleaned dataset
df.to_csv("cleaned_dataset.csv", index=False)
```

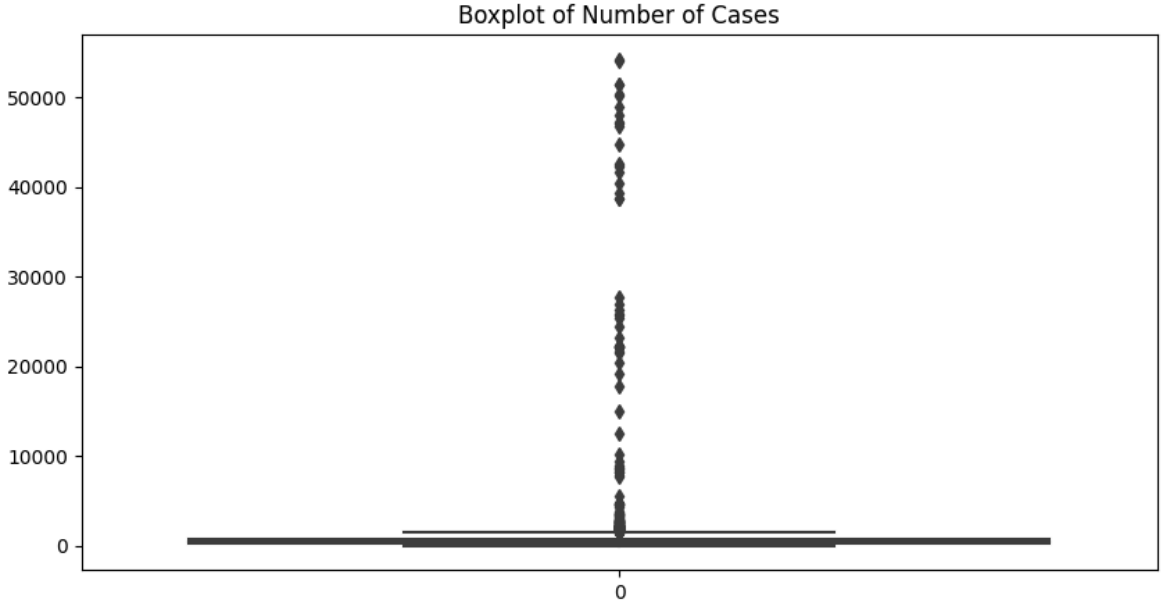
In [57]:

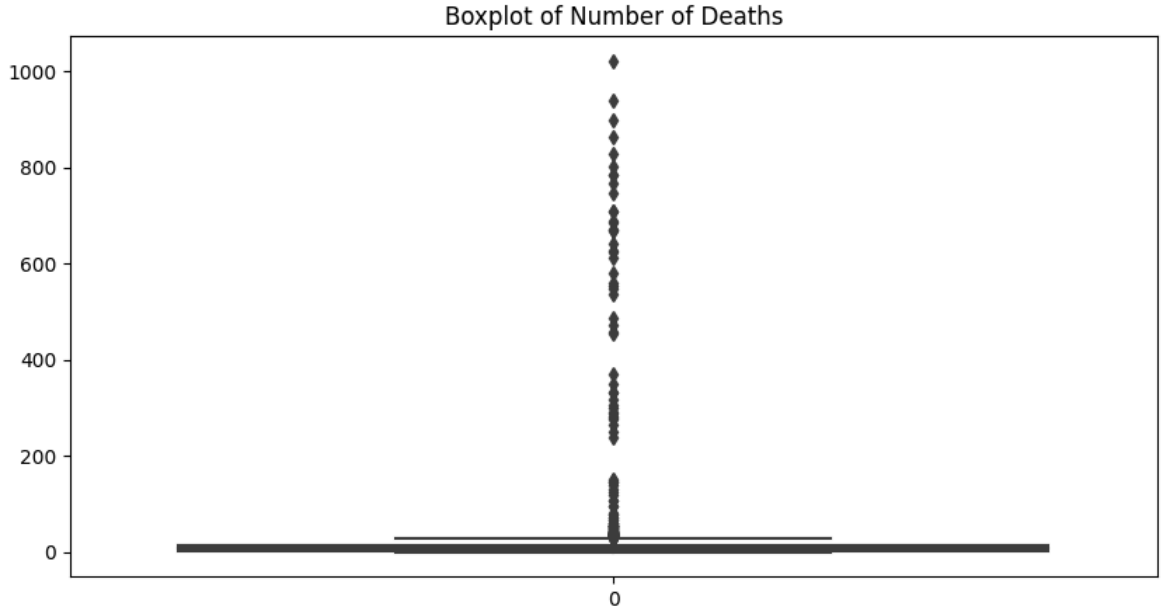
```
# Loading the cleaned dataset
df = pd.read_csv("cleaned_dataset.csv")

# Re-checking for missing values
missing_values = df.isnull().sum()
print("Missing values in each column:\n", missing_values)

# Inspecting data types
print("\nData types:\n", df.dtypes)

# Re-checking for duplicates
duplicates = df.duplicated().sum()
print("\nNumber of duplicate rows:", duplicates)

# Outlier detection with boxplots for numerical columns
import seaborn as sns
import matplotlib.pyplot as plt

numeric_columns = df.select_dtypes(include=['int64', 'float64']).columns
for column in numeric_columns:
    plt.figure(figsize=(10, 5))
    sns.boxplot(df[column])
    plt.title(f'Boxplot of {column}')
    plt.show()
```

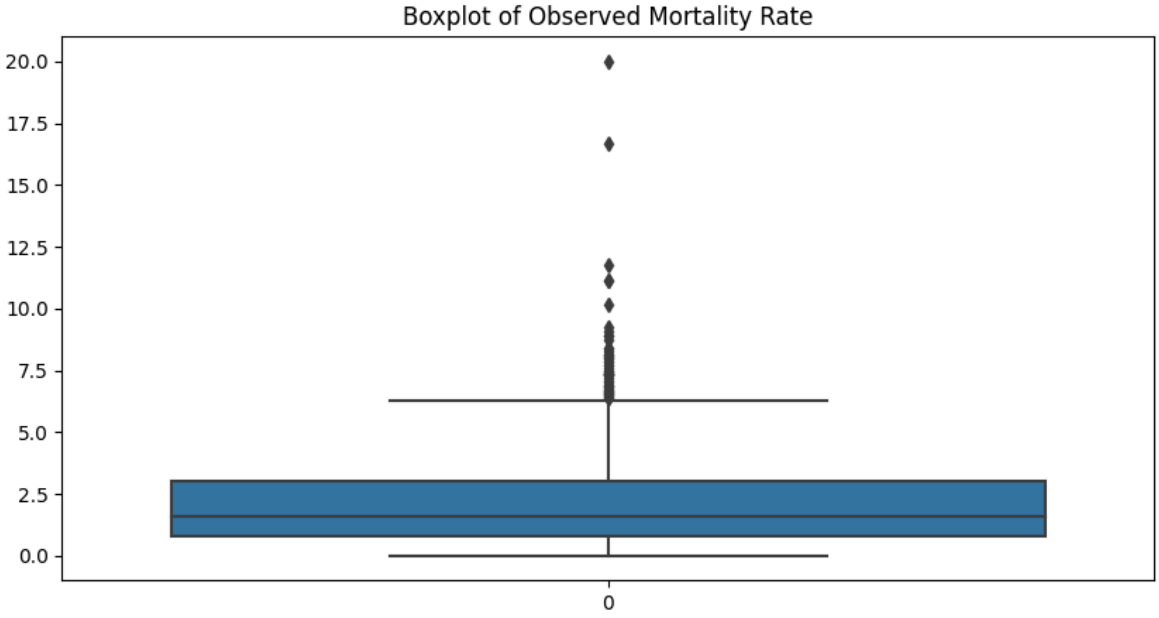
```
Missing values in each column:
 Facility ID                           0
Hospital Name                          0
Detailed Region                        0
Region                                 0
Procedure                              0
Year of Hospital Discharge             0
Number of Cases                        0
Number of Deaths                       0
Observed Mortality Rate                0
Expected Mortality Rate                0
Risk-Adjusted Mortality Rate           0
Lower Limit of Confidence Interval     0
Upper Limit of Confidence Interval     0
Comparison Results                     0
procedure_category                     0
dtype: int64

Data types:
 Facility ID                            int64
Hospital Name                          object
Detailed Region                        object
Region                                 object
Procedure                              object
Year of Hospital Discharge            float64
Number of Cases                         int64
Number of Deaths                        int64
Observed Mortality Rate               float64
Expected Mortality Rate               float64
Risk-Adjusted Mortality Rate          float64
Lower Limit of Confidence Interval    float64
Upper Limit of Confidence Interval    float64
Comparison Results                     object
procedure_category                      int64
dtype: object

Number of duplicate rows: 0
```

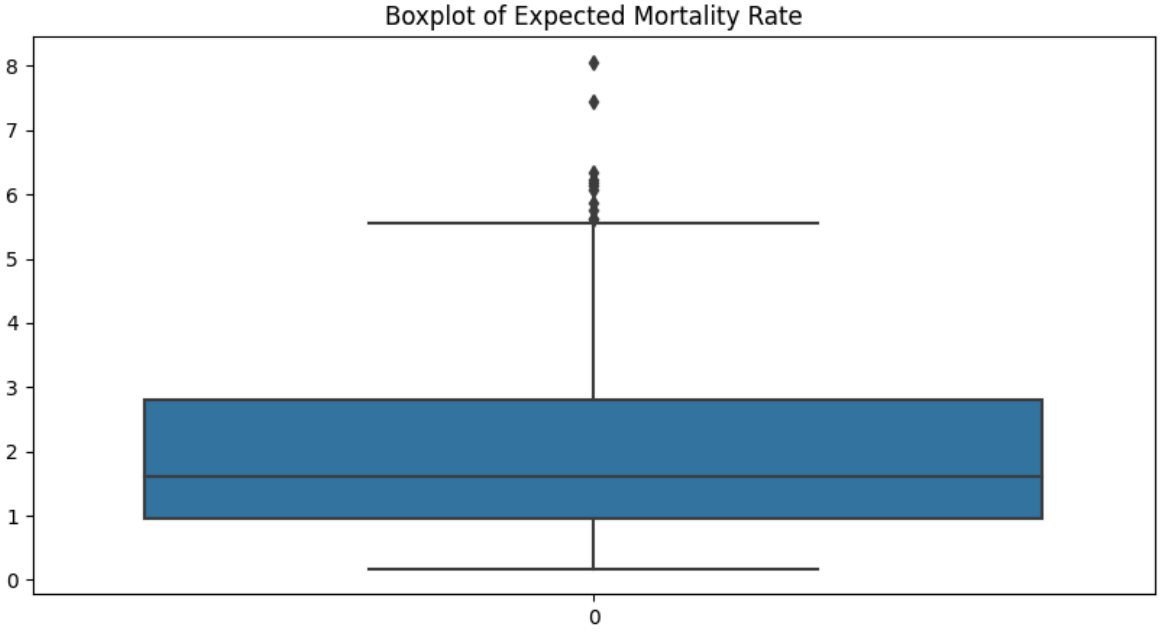


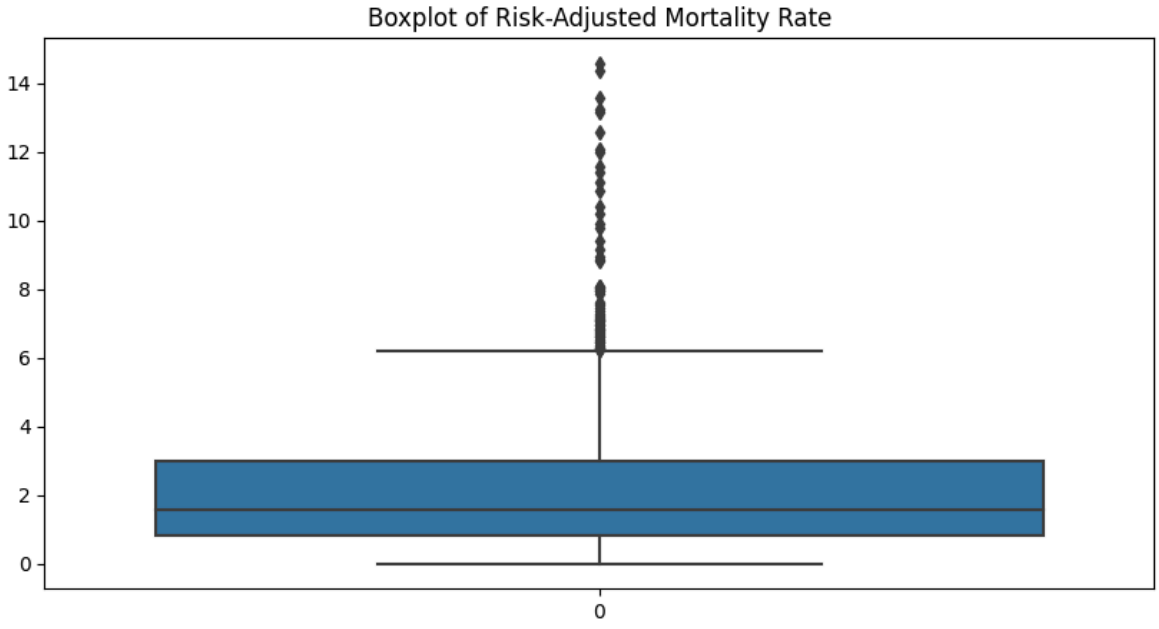

Boxplot of Year of Hospital Discharge
2018
2016
2014
2012
2010
0

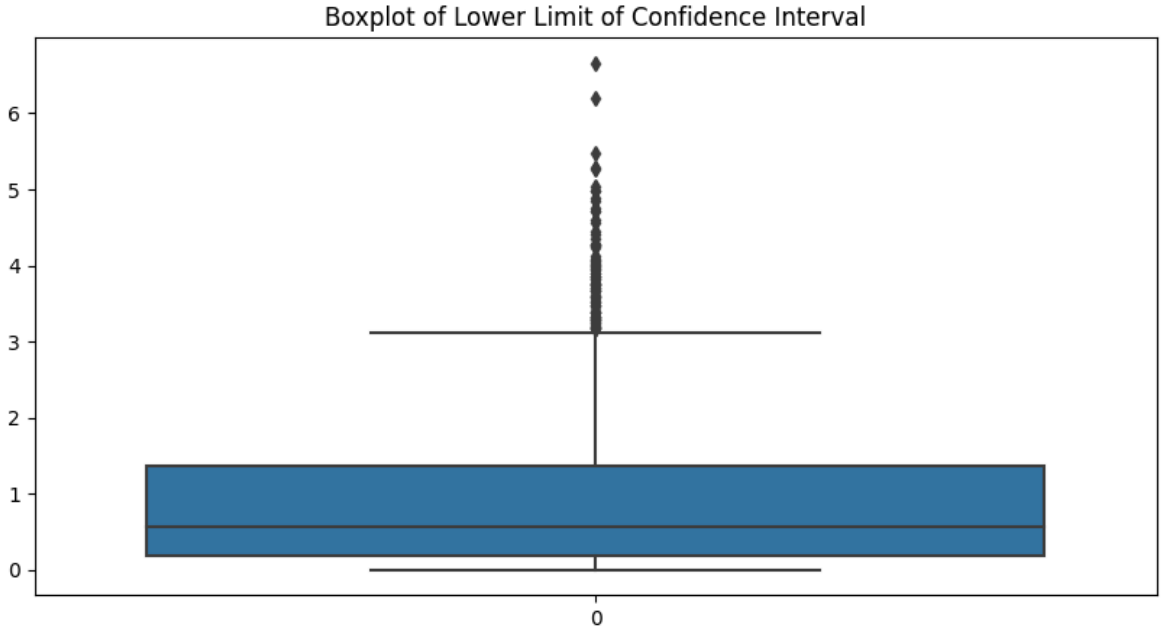

Boxplot of Number of Cases
50000
40000
30000
20000
10000
0
0

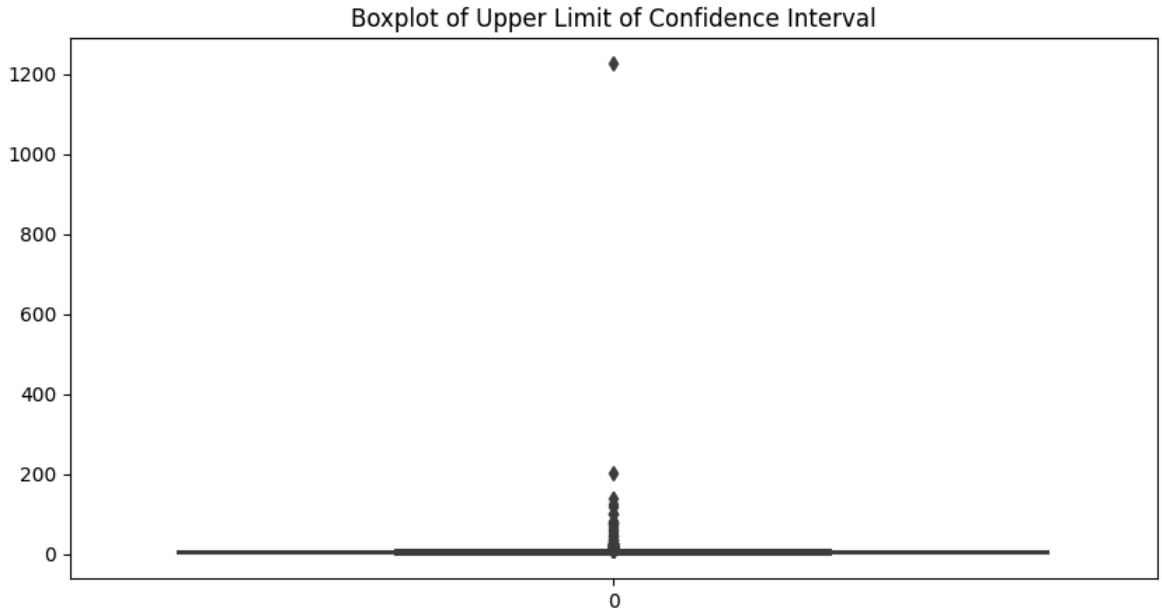

Boxplot of Number of Deaths
1000
800
600
400
200
0
0

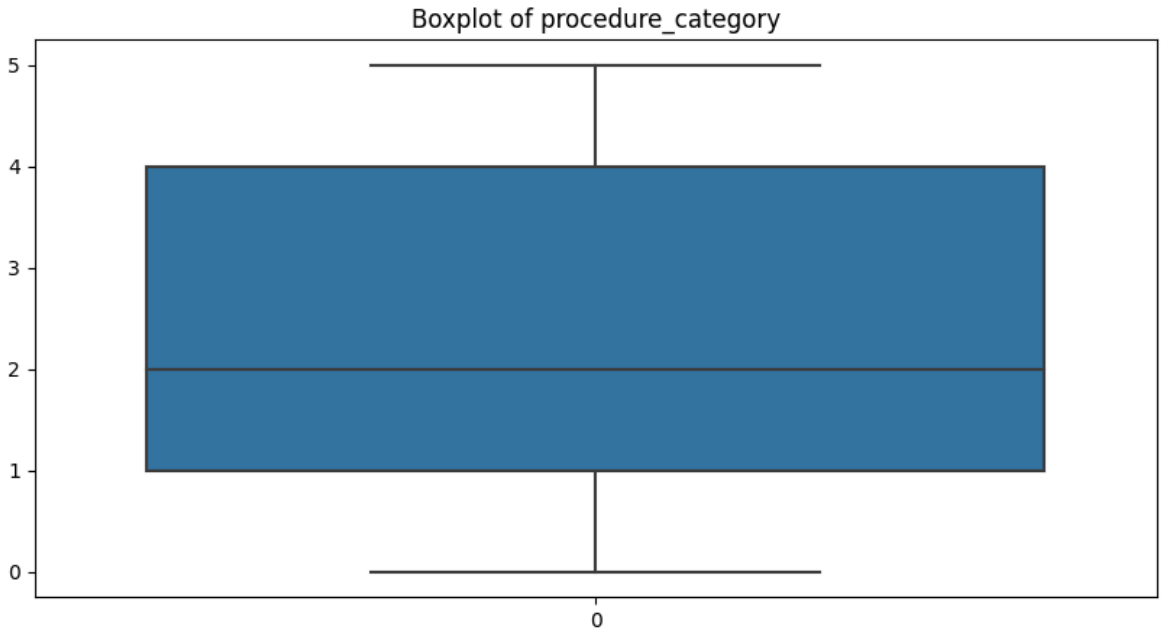

Boxplot of Observed Mortality Rate
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0


Boxplot of Expected Mortality Rate
8
7
6
5
4
3
2
1
0
0


Boxplot of Risk-Adjusted Mortality Rate
14
12
10
8
6
4
2
0
0







Inconsistent Data Formats:

1. The program detects inconsistencies in data formats by examining whether each value in the dataset is a string.
2. It tallies up the occurrences of mismatched data formats across all columns. Presents the total count, incomplete Data Entries.
3. It spots data entries by searching for rows with at least one missing value.
4. The number of entries is shown, revealing a total of 2812 such instances.

Data Quality Visualization:

Facility Identification: In this box plot there is one data point that stands out significantly from the rest. Since Facility ID is likely an identifier rather than a numerical value this outlier could simply be the highest assigned ID and may not impact numerical analysis.

Year of Hospital Discharge: The data appears to be concentrated around years with no clear outliers. No action is necessary regarding outliers in this case.

Number of Cases & Number of Deaths: There are data points above the upper whisker indicating that certain facilities have notably higher numbers of cases and deaths. This might be typical for hospitals or those specializing in cardiac care. Considering these points could offer insights for your analysis it's advisable not to remove or modify them.

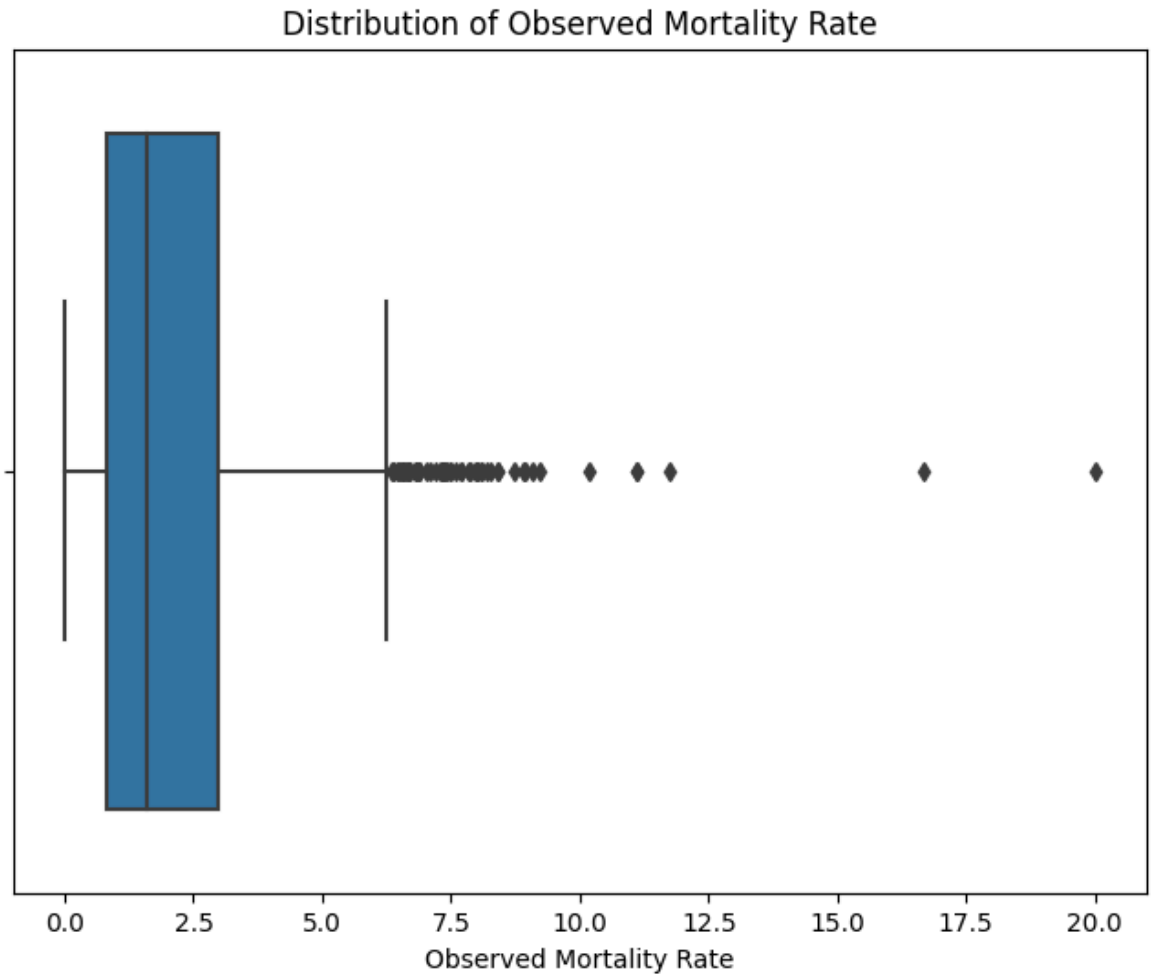
Observed Mortality Rate, Expected Mortality Rate & Risk Adjusted Mortality Rate: The presence of outliers, in these rates could signal instances of high or low mortality rates. These outliers may hold significance for your analysis as they could represent cases or important anomalies. Its recommended to retain these outliers in the dataset for an examination. The lower boundary of the confidence interval and the upper boundary of the confidence interval Certain values fall beyond the range but similar to mortality rates these extreme cases might hold significance in comprehending the diversity, in your data. It is advisable to retain them as they could provide insights.

In [58]:
```
# Summary statistics for numeric data
summary_stats = df.describe()
print(summary_stats)
```

```
       Facility ID  Year of Hospital Discharge  Number of Cases  \
count      2812.00                     2812.00          2812.00
mean        897.57                     2015.58          1080.07
std         602.38                        2.50          4359.57
min           0.00                     2010.00             1.00
25%         456.50                     2014.00           202.00
50%         885.00                     2017.00           375.00
75%        1438.00                     2017.00           702.25
max        3058.00                     2019.00         54276.00

       Number of Deaths  Observed Mortality Rate  Expected Mortality Rate
\
count           2812.00                 2812.00                  2812.00
mean              18.60                    2.07                     1.95
std               74.35                    1.70                     1.20
min                0.00                    0.00                     0.17
25%                3.00                    0.81                     0.95
50%                7.00                    1.60                     1.61
75%               13.00                    2.99                     2.81
max             1021.00                   20.00                     8.06

       Risk-Adjusted Mortality Rate  Lower Limit of Confidence Interval  \
count                       2812.00                             2812.00
mean                           2.08                                0.90
std                            1.75                                0.96
min                            0.00                                0.00
25%                            0.84                                0.18
50%                            1.56                                0.56
75%                            3.00                                1.36
max                           14.59                                6.66

       Upper Limit of Confidence Interval  procedure_category
count                             2812.00            2812.00
mean                                 5.93               2.33
std                                 24.64               1.90
min                                  0.58               0.00
25%                                  2.33               1.00
50%                                  4.06               2.00
75%                                  5.97               4.00
max                               1228.42               5.00
```

In [59]:
```
# Visualizing the distribution of observed mortality rate
plt.figure(figsize=(8, 6))
sns.boxplot(x=df['Observed Mortality Rate'])
plt.title('Distribution of Observed Mortality Rate')
plt.xlabel('Observed Mortality Rate')
plt.show()

# Calculating the interquartile range (IQR)
Q1 = df['Observed Mortality Rate'].quantile(0.25)
Q3 = df['Observed Mortality Rate'].quantile(0.75)
IQR = Q3 - Q1

# Defining the threshold for outliers
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Identifying outliers
outliers = df[(df['Observed Mortality Rate'] < lower_bound) | (df['Observ
```

```
# Displaying the outliers
print("Outliers in Observed Mortality Rate:")
print(outliers[['Hospital Name', 'Observed Mortality Rate']])
```



```
Outliers in Observed Mortality Rate:
                Hospital Name  Observed Mortality Rate
142      Mount Sinai Hospital                     7.30
159     Bellevue Hospital Ctr                     8.08
161    Bronx-Lebanon-Cncourse                     6.80
162    Brookdale Hosp Med Ctr                     8.02
194       SVCMC- St. Vincents                    11.76
215     White Plains Hospital                     7.32
220       Beth Israel Med Ctr                     7.58
229       Lenox Hill Hospital                     7.35
236       Montefiore - Weiler                     6.59
245       SVCMC- St. Vincents                     7.84
387       Univ. Hosp-Brooklyn                    16.67
393    NYP-Brooklyn Methodist                     7.50
419       Univ. Hosp-Brooklyn                    11.11
435        Faxton - St. Lukes                     7.89
479    NYP-Brooklyn Methodist                     6.63
489       Beth Israel Med Ctr                     6.53
502       Montefiore - Weiler                     6.89
515    St. Lukes at St. Lukes                     6.39
576       St. Peters Hospital                     7.35
644     NY Methodist Hospital                     7.10
800       Univ. Hosp-Brooklyn                     8.25
874    NYP-Brooklyn Methodist                     6.55
875     St. Barnabas Hospital                     6.67
876    Univ. Hosp-Stony Brook                     8.41
889        Faxton - St. Lukes                     7.38
1015     St. Josephs Hospital                     6.47
1029       Faxton - St. Lukes                     7.20
1103       Montefiore - Moses                     6.36
1115     Strong Memorial Hosp                     8.18
1168      Univ. Hosp-Brooklyn                     7.35
1175       Univ. Hosp-Upstate                     6.67
1186   Univ. Hosp-Stony Brook                     8.73
1188     Strong Memorial Hosp                     7.41
1302     Southampton Hospital                     8.00
1332   Champ.Valley Phys Hosp                     9.09
1337   Univ. Hosp-Stony Brook                     6.82
1393      Arnot Ogden Med Ctr                     8.41
1424      Arnot Ogden Med Ctr                     8.08
1663      Univ. Hosp-Brooklyn                    11.11
1742   Univ. Hosp-Stony Brook                     6.85
1756      Peconic Bay Med Ctr                     7.69
1788    St. Elizabeth Med Ctr                     6.48
1921    St. Elizabeth Med Ctr                     8.00
2021      Arnot Ogden Med Ctr                     7.32
2080       Faxton - St. Lukes                     7.69
2114     Southampton Hospital                    20.00
2168       Univ. Hosp-Upstate                     9.23
2417     Southampton Hospital                     7.50
2428           Unity Hospital                     7.03
2436      Arnot Ogden Med Ctr                     8.89
2449  Mount Sinai Beth Israel                     6.56
2703   NYP Brooklyn Methodist                     6.71
2704     NYP Columbia Presby.                     6.89
2739      Arnot Ogden Med Ctr                     8.93
2771   Univ Hosp at Downstate                    10.17
```

It seems like there are several hospitals with outlier observed mortality rates. These outlier values may indicate potential areas of concern or excellence in hospital

performance.

Outliers can sometimes provide valuable insights into our data, and they should not be removed without proper justification or understanding of their underlying reasons. Since we are missing dataset that can describe the hospital demographics in detail like size, number of doctors, cardiac specialist hospital etc. we cannot perform further outlier analysis. Theefore, we will keep the outliers in our model as they can prove to be effecient.

Objective:

The aim is to find the successful medical procedures based on a hospitals track record and various factors like location, efficiency metrics of procedures (such as mortality rates) and the number of cases handled.

Key Factors

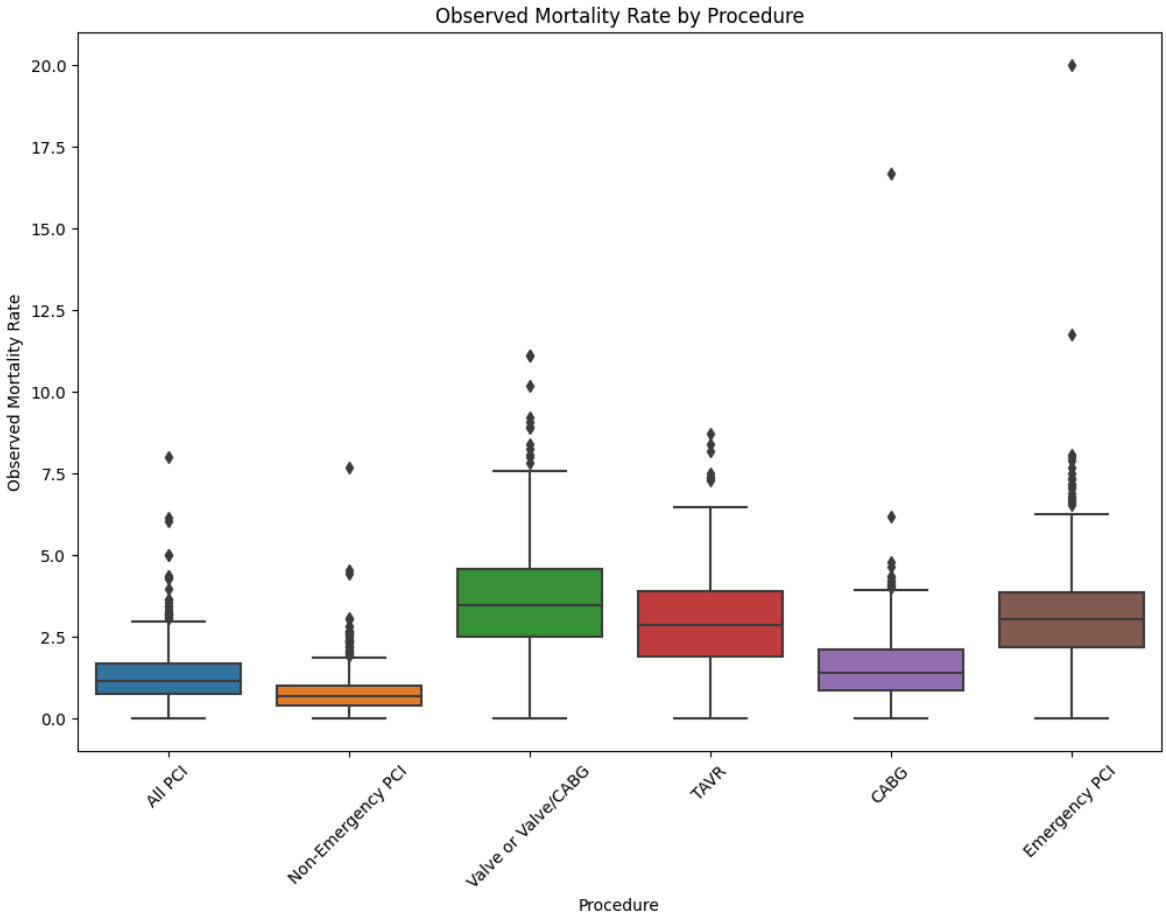
Mortality Rate Observed: This reflects the mortality rate seen in patients undergoing a specific medical procedure. A observed mortality rate might suggest a more favorable outcome for the procedure.

Case Volume: The quantity of cases for each procedure can offer insights into its popularity and how experienced the hospital is with it.

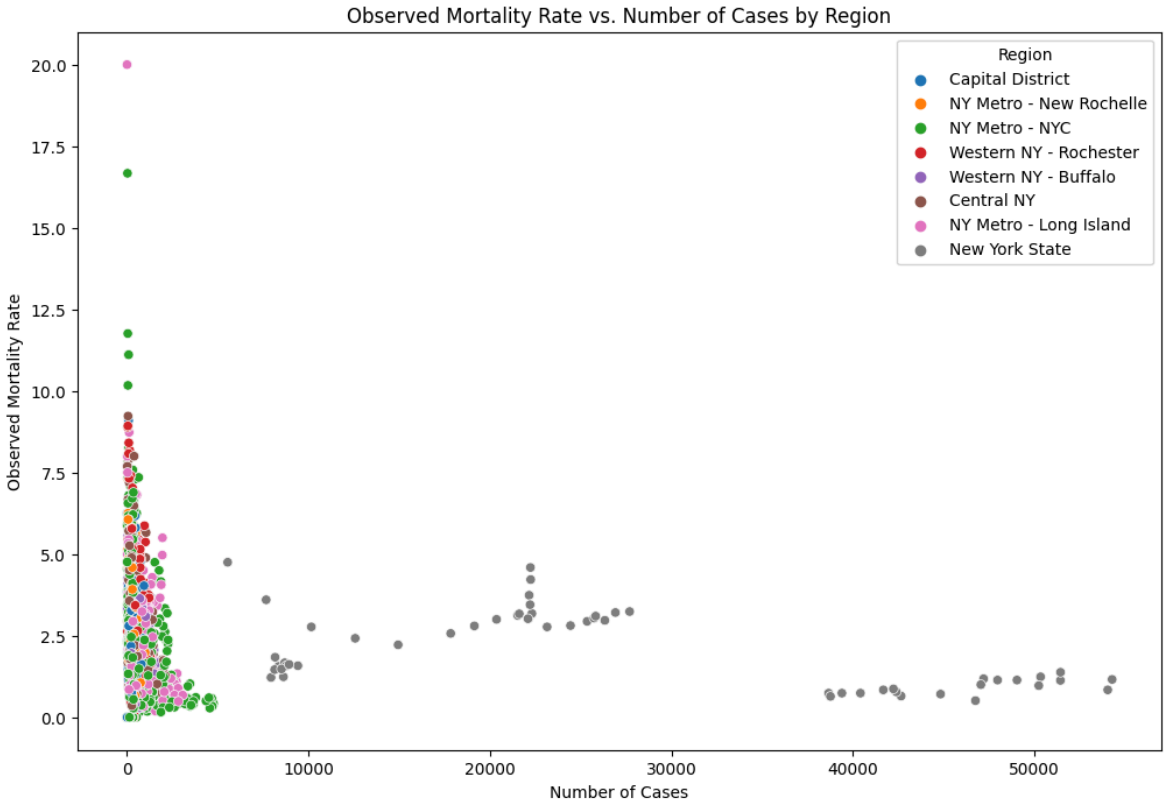
Number of Deaths: Knowing the number of deaths linked to each procedure is essential for evaluating its safety and efficacy.

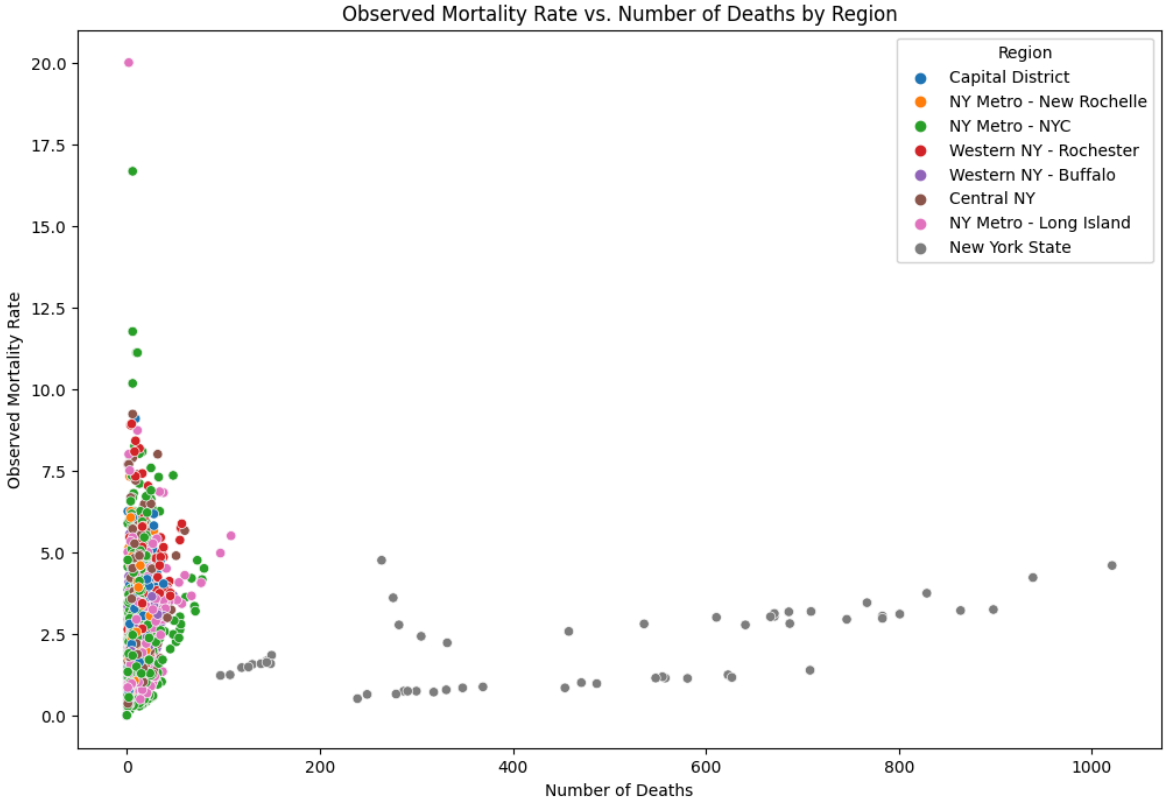
Location: Differences in regions could impact healthcare quality, resources and patient characteristics influencing results.

In [60]:
```
# Box plot for observed mortality rate by procedure
plt.figure(figsize=(12, 8))
sns.boxplot(x='Procedure', y='Observed Mortality Rate', data=df)
plt.xticks(rotation=45)
plt.title('Observed Mortality Rate by Procedure')
plt.xlabel('Procedure')
plt.ylabel('Observed Mortality Rate')
plt.show()
```

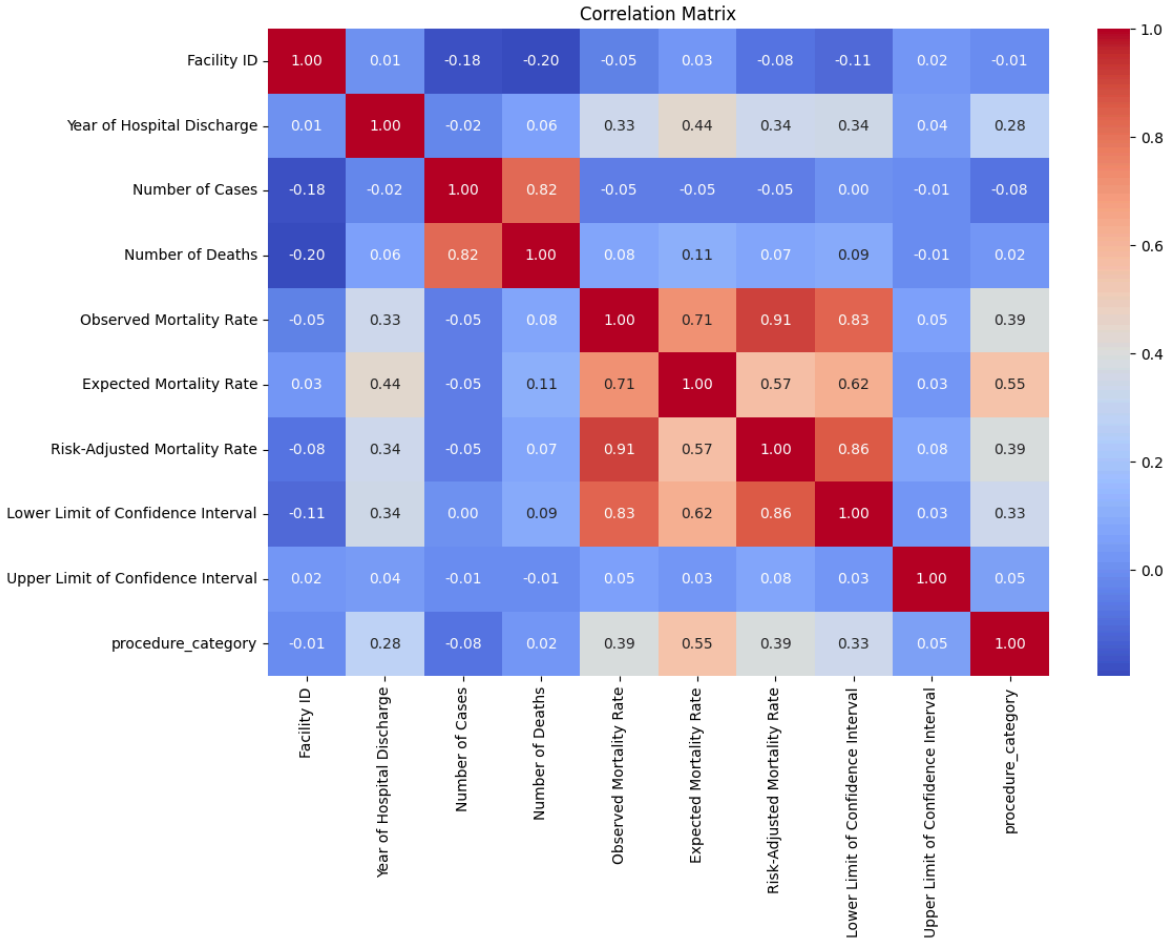


In [61]:
```
# Scatter plot of observed mortality rate vs. number of cases colored by
plt.figure(figsize=(12, 8))
sns.scatterplot(x='Number of Cases', y='Observed Mortality Rate', hue='Re
plt.title('Observed Mortality Rate vs. Number of Cases by Region')
plt.xlabel('Number of Cases')
plt.ylabel('Observed Mortality Rate')
plt.legend(title='Region', loc='upper right')
plt.show()

# Scatter plot of observed mortality rate vs. number of deaths colored by
plt.figure(figsize=(12, 8))
sns.scatterplot(x='Number of Deaths', y='Observed Mortality Rate', hue='R
plt.title('Observed Mortality Rate vs. Number of Deaths by Region')
plt.xlabel('Number of Deaths')
plt.ylabel('Observed Mortality Rate')
plt.legend(title='Region', loc='upper right')
plt.show()
```





In [62]:
```
# Plotting the correlation matrix as a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```



The correlation matrix displays how variables in the dataset are related to each other. Here's a brief explanation of the correlation coefficients.

Facility ID vs. Variables: There seems to be a slight negative correlation between Facility ID and certain variables suggesting that higher Facility IDs might be linked to slightly lower values in those variables.

Year of Hospital Discharge vs. Other Variables: The correlation with other variables is generally weak except for a moderate positive correlation with some factors related to mortality.

Number of Cases vs. Variables: There are strong positive correlations between the number of cases and mortality related factors indicating that higher case numbers could be linked to higher mortality rates.

Number of Deaths vs. Variables: As expected there are strong positive correlations between the number of deaths and mortality related factors.

Observed Mortality Rate vs. Variables: Strong positive correlations exist between the mortality rate and other mortality related factors showing consistency in measuring mortality rates.

Expected Mortality Rate vs. Variables: Positive correlations are seen between the expected mortality rate and other mortality related factors.

Risk Adjusted Mortality Rate, vs. Variables: Strong positive correlations exist between the risk adjusted mortality rate and other mortality related factors.

Positive relationships are found between the limit of confidence interval and factors related to mortality. Similarly there are connections between the upper limit of confidence interval and variables associated with mortality. In addition there are positive links between the procedure category and certain mortality related factors suggesting potential variations in mortality rates, among different procedure categories.

Model

In [63]:
```
# Feature Selection based on correlation
correlation_matrix = df.corr()
# Considering only features with a moderate amount of correlation with 'O
selected_features = correlation_matrix['Observed Mortality Rate'][(correl

# Ensuring the target variable 'Observed Mortality Rate' is present in th
if 'Observed Mortality Rate' in selected_features:
    selected_features.remove('Observed Mortality Rate')  # Removing the t

# Preparing Data for Model
X = df[selected_features]  # Features
y = df['Observed Mortality Rate']  # Target

# Splitting dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,

# Standardizing the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Initializing models
model_lr = LinearRegression()
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)

# Training models
model_lr.fit(X_train_scaled, y_train)
model_rf.fit(X_train_scaled, y_train)

# Performing cross-validation
cv_scores_lr = cross_val_score(model_lr, X_train_scaled, y_train, cv=5, s
cv_scores_rf = cross_val_score(model_rf, X_train_scaled, y_train, cv=5, s

# Cross-validation results
print(f"Linear Regression CV MSE: {-cv_scores_lr.mean()}, STD: {cv_scores
print(f"Random Forest CV MSE: {-cv_scores_rf.mean()}, STD: {cv_scores_rf.

# Predictions and evaluation based on the test set
y_pred_lr = model_lr.predict(X_test_scaled)
y_pred_rf = model_rf.predict(X_test_scaled)
mse_lr = mean_squared_error(y_test, y_pred_lr)
r2_lr = r2_score(y_test, y_pred_lr)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
```

```
# Test set results
print(f"Linear Regression Test MSE: {mse_lr}, R2: {r2_lr}")
print(f"Random Forest Test MSE: {mse_rf}, R2: {r2_rf}")

# Print the features used in model training
print("Features used in model training:", selected_features)
```

```
Linear Regression CV MSE: 1.363622401503116, STD: 0.15197336259148633
Random Forest CV MSE: 1.7405585172217368, STD: 0.1524472099027905
Linear Regression Test MSE: 1.6281602371642618, R2: 0.5143328517798206
Random Forest Test MSE: 2.0219749255198325, R2: 0.39686108686681176
Features used in model training: ['Year of Hospital Discharge', 'Expected
Mortality Rate', 'procedure_category']
```

Why Specific Models Were Chosen:

Linear Regression (LR):

1. Interpretation; LR assigns coefficients to each feature making it simple to understand how each feature impacts the predicted outcome.
2. Linearity Assumption; LR assumes a connection between the features and the target variable, which is often valid in real world situations.
3. Simplicity; LR is efficient computationally and straightforward to implement making it an ideal starting point for regression tasks.
4. Baseline Comparison; LR acts as a model for comparing more complex models.

Random Forest Regressor (RF):

1. Handling Non linearity; RF can capture non linear relationships between features and the target variable that may not be captured by LR.
2. Overfitting Resilience; RF is less susceptible to overfitting than decision trees since it averages predictions from multiple trees.
3. Feature Significance; RF offers insights into feature importance aiding in identifying predictors of the target variable.
4. Model Performance; RF generally performs across various datasets in practical settings making it a popular choice, for regression tasks.

The trained models are utilized to predict outcomes on test data.

They are assessed using two measures:

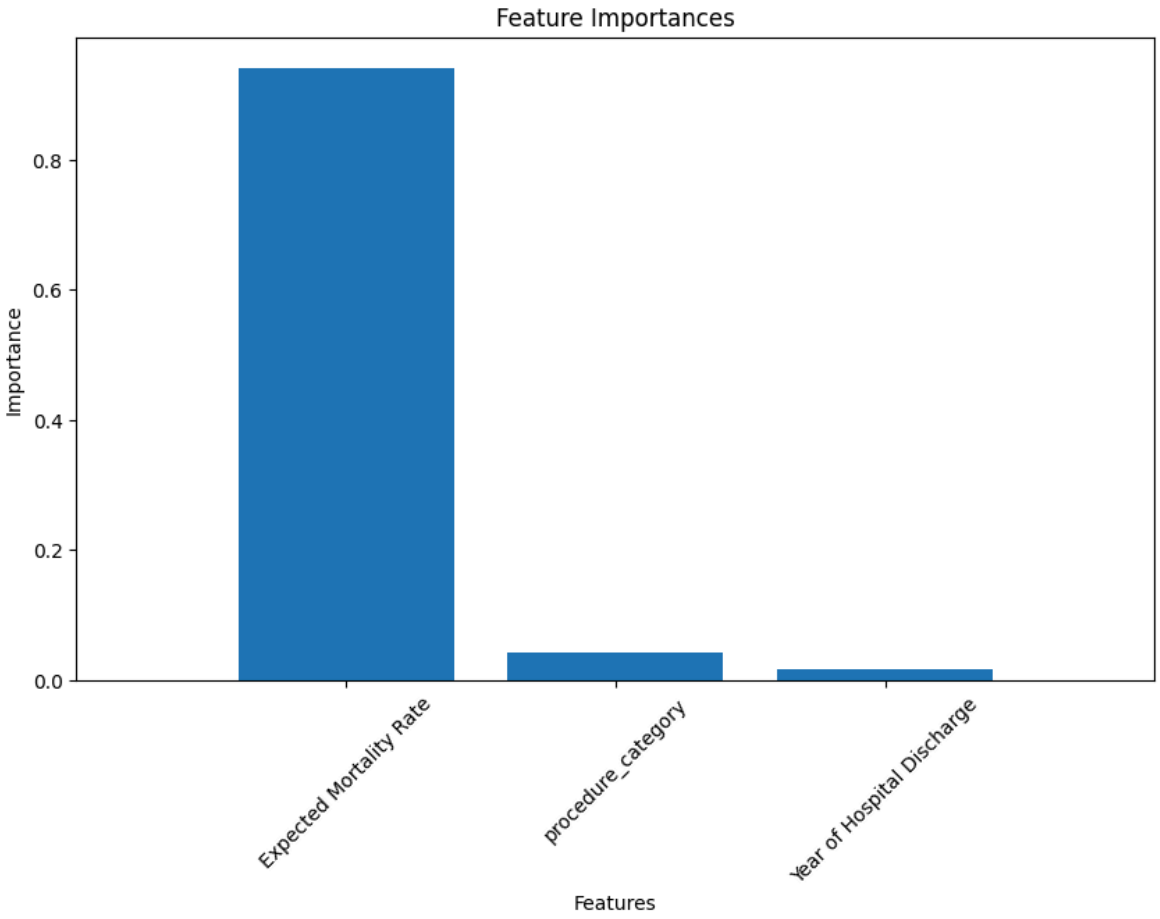
1. Mean Squared Error (MSE); This calculates the average of the errors representing the average squared variance between the estimated values and the actual value.

2. R squared (R^2); This is an indicator of how closely the data align with the regression line. Its also termed as the coefficient of determination or multiple determination for regression.

What is Being Predicted by the Model? The model predicts the 'Observed Mortality Rate' for a procedure based on various included variables serving as features. Simply put, given information about a procedure and its location within a hospital (along with possibly other relevant factors) the model forecasts what the mortality rate might be.

Current Outcomes: The outcomes illustrate how two models performed. The Linear Regression model seems to exhibit performance based on MSE and R^2 scores suggesting it fits your provided data better. Conversely the Random Forest model showed performance possibly due to overfitting or requiring adjustments, in hyperparameters.

In [66]:
```
# Get feature importances from the model
importances = best_grid.feature_importances_
indices = np.argsort(importances)[::-1]
feature_names = X.columns

# Plot the feature importances
plt.figure(figsize=(10, 6))
plt.title("Feature Importances")
plt.bar(range(X.shape[1]), importances[indices], align="center")
plt.xticks(range(X.shape[1]), feature_names[indices], rotation=45)
plt.xlim([-1, X.shape[1]])
plt.ylabel('Importance')
plt.xlabel('Features')
plt.show()
```



Fine tuning Hyperparameters:

To improve the Random Forest model we can refine it through hyperparameter tuning using methods like grid search or random search. Grid search involves exploring specific parameter values to find the best ones for a model. It's an approach that leaves no stone unturned. On the hand random search selects hyperparameter values randomly and is less systematic.

In this case we'll opt for grid search due, to its simplicity. Because the exhaustive process is not a major concern since we only need to adjust a few parameters in this scenario.

In [67]:
```
# Parameters for tuning
param_grid = {
    'n_estimators': [100, 200, 300],  # Number of trees in the random for
    'max_depth': [None, 10, 20, 30],  # Maximum depth of the trees
    'min_samples_split': [2, 5, 10],  # Minimum number of samples require
    'min_samples_leaf': [1, 2, 4]  # Minimum number of samples required a
}

# Initializing the grid search model
grid_search = GridSearchCV(estimator=RandomForestRegressor(random_state=4
                           param_grid=param_grid,
                           cv=3,  # Number of folds in cross-validation
                           n_jobs=-1,  # Use all processors
                           verbose=2,  # Controls the verbosity: the high
                           scoring='neg_mean_squared_error')  # Negative

# Fitting the grid search to the data
grid_search.fit(X_train_scaled, y_train)

# Best parameters found
print(f"Best parameters: {grid_search.best_params_}")

# Initialize the best model from grid search
best_grid = grid_search.best_estimator_

# Evaluating the best model on the test set
y_pred_best_grid = best_grid.predict(X_test_scaled)
mse_best_grid = mean_squared_error(y_test, y_pred_best_grid)
r2_best_grid = r2_score(y_test, y_pred_best_grid)

print(f"Optimized Random Forest Test MSE: {mse_best_grid}, R2: {r2_best_g
```

```
Fitting 3 folds for each of 108 candidates, totalling 324 fits
Best parameters: {'max_depth': 10, 'min_samples_leaf': 4, 'min_samples_spl
it': 10, 'n_estimators': 100}
Optimized Random Forest Test MSE: 1.7130704143788613, R2: 0.48900482651462
42
```

The hyperparameter tuning has identified the best parameters from the given range for the Random Forest model:

max_depth: 10 min_samples_leaf: 4 min_samples_split: 10 n_estimators: 100

Using these parameters, the Random Forest's performance on the test set has an MSE of 1.7131 and an R^2 of 0.4890. Compared to the untuned Random Forest model's test MSE of 2.0219 and R^2 of 0.3969, the tuned model has a lower MSE, which is an improvement. However, the R^2 is higher (closer to 0.5), indicating that the tuned model explains almost half of the variability in the observed mortality rate, which is also an improvement.

In [68]:
```
# Initialize and train the Gradient Boosting model
model_gbm = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1
```

```
model_gbm.fit(X_train_scaled, y_train)

# Make predictions and evaluate
y_pred_gbm = model_gbm.predict(X_test_scaled)
mse_gbm = mean_squared_error(y_test, y_pred_gbm)
r2_gbm = r2_score(y_test, y_pred_gbm)

print(f"Gradient Boosting Test MSE: {mse_gbm}, R2: {r2_gbm}")
```

```
Gradient Boosting Test MSE: 1.6753714393925643, R2: 0.500250128576728
```

Further Tuning of the Gradient Boosting Model

To further tune the Gradient Boosting model, we can use GridSearchCV to experiment with a few key hyperparameters.

In [69]:
```
# Hyperparameter grid
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 4, 5]
}

# Initialize the model
gbm = GradientBoostingRegressor(random_state=42)

# Initialize the grid search model
grid_search = GridSearchCV(estimator=gbm, param_grid=param_grid, cv=3, sc

# Fit the grid search to the data
grid_search.fit(X_train_scaled, y_train)

# Best parameters found
print("Best parameters:", grid_search.best_params_)

# Best model
best_gbm = grid_search.best_estimator_

# Predictions
y_pred_gbm = best_gbm.predict(X_test_scaled)

# Evaluation
mse_gbm = mean_squared_error(y_test, y_pred_gbm)
r2_gbm = r2_score(y_test, y_pred_gbm)
print(f"Optimized Gradient Boosting Test MSE: {mse_gbm}, R2: {r2_gbm}")
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
Best parameters: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 1
00}
Optimized Gradient Boosting Test MSE: 1.6581961565682204, R2: 0.5053733777
746707
```

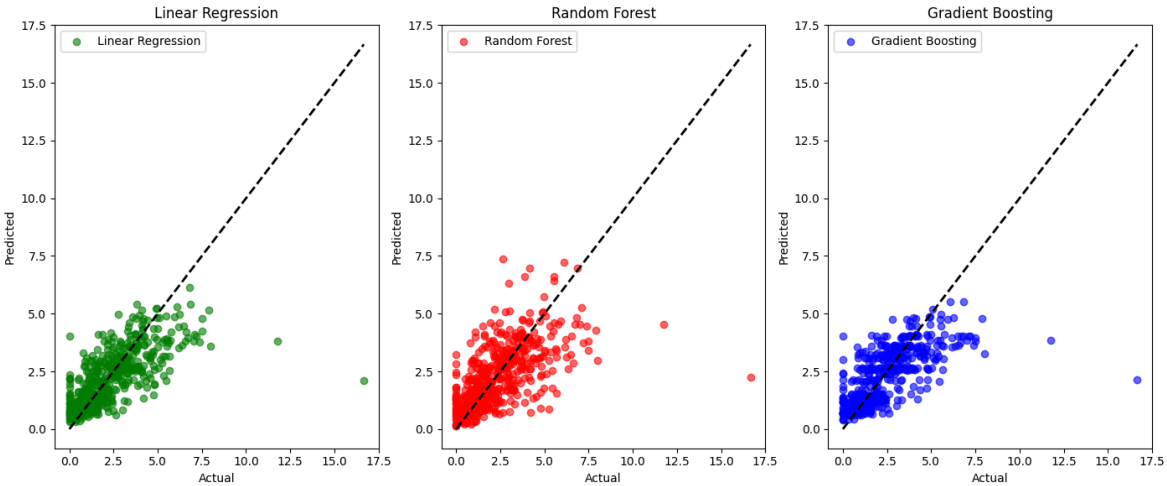
The hyperparameter tuning for the Gradient Boosting model has resulted in the identification of the best parameters as follows:

learning_rate: 0.05 max_depth: 3 n_estimators: 100

Using these optimized parameters, the Gradient Boosting model achieved a Mean Squared Error (MSE) of 1.6582 and an R-squared (R^2) value of 0.5054 on the test set.

This performance is slightly better than the previous iteration of the Gradient Boosting model and indicates that about 50.5% of the variance in the observed mortality rate can be explained by the model. It also represents a good balance between bias and variance, avoiding overfitting while maintaining a decent level of predictive accuracy.

In [70]:
```
y_pred_lr = model_lr.predict(X_test_scaled)  # Linear Regression predicti
y_pred_rf = model_rf.predict(X_test_scaled)  # Random Forest predictions
y_pred_gbm = best_gbm.predict(X_test_scaled)  # Gradient Boosting predict

# Plotting the results
plt.figure(figsize=(14, 6))

# Linear Regression
plt.subplot(1, 3, 1)
plt.scatter(y_test, y_pred_lr, alpha=0.6, color='green', label='Linear Re
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Linear Regression')
plt.legend()

# Random Forest
plt.subplot(1, 3, 2)
plt.scatter(y_test, y_pred_rf, alpha=0.6, color='red', label='Random Fore
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Random Forest')
plt.legend()

# Gradient Boosting
plt.subplot(1, 3, 3)
plt.scatter(y_test, y_pred_gbm, alpha=0.6, color='blue', label='Gradient
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--
plt.xlabel('Actual')
plt.ylabel('Predicted')
plt.title('Gradient Boosting')
plt.legend()

plt.tight_layout()
plt.show()
```



Data Points Relative to the Line: The closer the data points are to the dashed line (which represents perfect predictions), the more accurate the model's predictions are for those points.

Spread of Data Points: The scatter of points gives an indication of variance in the predictions. A tighter cluster along the diagonal suggests higher accuracy.

Outliers: Points that are far from the diagonal line could be considered outliers or instances where the model's predictions deviate significantly from the actual values.

Outcome:

The model has a decent number of predictions close to the line, indicating accurate predictions for those instances.

There is a spread that grows as the actual observed mortality rate increases, suggesting the model might be less accurate at higher mortality rates.

The model seems to be fairly consistent with predictions for lower mortality rates, but the spread increases with higher rates.

In [72]:
```
# Get the predicted mortality rates for each procedure from each model
y_pred_lr = model_lr.predict(X_test_scaled)
y_pred_rf = model_rf.predict(X_test_scaled)
y_pred_gbm = model_gbm.predict(X_test_scaled)

# Calculate the average predicted mortality rate for each procedure acros
average_predicted_mortality = (y_pred_lr + y_pred_rf + y_pred_gbm) / 3

# Find the procedure with the lowest average predicted mortality rate
best_procedure_index = np.argmin(average_predicted_mortality)
best_procedure = df['Procedure'].iloc[best_procedure_index]

print(f"The best procedure is: {best_procedure}")
```

```
The best procedure is: All PCI
```

In practice, this model could be used by healthcare professionals or hospital administrators to estimate and compare the expected mortality rates for various procedures across different facilities, aiding in decision-making processes about where to direct patients or how to improve certain procedures.

In [ ]:
```
```