Step 1: Importing Necessary Libraries

First, we need to import the required libraries that will help us manipulate the data and create visualizations.

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

Step 2: Loading the Dataset

Next, we load the hospital dataset into a pandas DataFrame. This allows us to inspect the data structure and understand what we're working with.

In [2]:
```
# Load the dataset
hosp_info = pd.read_csv('HospInfo.csv')

# Display the first few rows and basic information about the dataset
hosp_info.head()
hosp_info.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4812 entries, 0 to 4811
Data columns (total 29 columns):
 #   Column                                                Non-Null Count  Dtype
---  ------                                                --------------  -----
 0   Provider ID                                           4812 non-null   int64
 1   Hospital Name                                         4812 non-null   object
 2   Address                                               4812 non-null   object
 3   City                                                  4812 non-null   object
 4   State                                                 4812 non-null   object
 5   ZIP Code                                              4812 non-null   int64
 6   County Name                                           4797 non-null   object
 7   Phone Number                                          4812 non-null   int64
 8   Hospital Type                                         4812 non-null   object
 9   Hospital Ownership                                    4812 non-null   object
 10  Emergency Services                                    4812 non-null   bool
 11  Meets criteria for meaningful use of EHRs             4668 non-null   object
 12  Hospital overall rating                               4812 non-null   object
 13  Hospital overall rating footnote                      1398 non-null   object
 14  Mortality national comparison                         4812 non-null   object
 15  Mortality national comparison footnote                1352 non-null   object
 16  Safety of care national comparison                    4812 non-null   object
 17  Safety of care national comparison footnote           2168 non-null   object
 18  Readmission national comparison                       4812 non-null   object
 19  Readmission national comparison footnote              1017 non-null   object
 20  Patient experience national comparison                4812 non-null   object
 21  Patient experience national comparison footnote       1369 non-null   object
 22  Effectiveness of care national comparison             4812 non-null   object
 23  Effectiveness of care national comparison footnote    1202 non-null   object
 24  Timeliness of care national comparison                4812 non-null   object
 25  Timeliness of care national comparison footnote       1266 non-null   object
 26  Efficient use of medical imaging national comparison  4812 non-null   object
 27  Efficient use of medical imaging national comparison footnote  2033 non-null   object
 28  Location                                              4812 non-null   object
dtypes: bool(1), int64(3), object(25)
memory usage: 1.0+ MB
```

What We Did: We loaded the dataset from a CSV file and displayed the first few rows to get an initial look at the data. We also checked the structure of the dataset to understand the number of rows, columns, and the types of data we're dealing with.

What We Got: We observed the top few rows, which gave us a snapshot of the data. The .info() output showed us that there are 4,812 rows and 29 columns, with a mix of categorical and numerical data types.

Next Steps: Now that we understand the structure of the data, we need to clean it, particularly by handling missing values and preparing it for analysis.

Step 3: Data Cleaning - Handling Missing Values

We'll now check for missing values in the dataset. This is important because missing data can lead to incorrect analysis if not handled properly.

In [3]:
```
# Check for missing values
missing_values = hosp_info.isnull().sum()

# Display missing values
missing_values
```

Out[3]:
```
Provider ID                                                  0
Hospital Name                                                0
Address                                                      0
City                                                         0
State                                                        0
ZIP Code                                                     0
County Name                                                 15
Phone Number                                                 0
Hospital Type                                                0
Hospital Ownership                                           0
Emergency Services                                           0
Meets criteria for meaningful use of EHRs                  144
Hospital overall rating                                      0
Hospital overall rating footnote                          3414
Mortality national comparison                                0
Mortality national comparison footnote                    3460
Safety of care national comparison                           0
Safety of care national comparison footnote               2644
Readmission national comparison                              0
Readmission national comparison footnote                  3795
Patient experience national comparison                       0
Patient experience national comparison footnote           3443
Effectiveness of care national comparison                    0
Effectiveness of care national comparison footnote        3610
Timeliness of care national comparison                       0
Timeliness of care national comparison footnote           3546
Efficient use of medical imaging national comparison         0
Efficient use of medical imaging national comparison footnote  2779
Location                                                     0
dtype: int64
```

What We Did: We checked each column for missing values by summing up the number of NaN (not a number) entries.

What We Got: The output showed the number of missing values in each column. This is crucial for determining how to handle these gaps, as some columns might have too many missing values to be useful.

Next Steps: Based on the number of missing values, we need to decide which columns to keep, which to drop, and how to handle the remaining missing data. We'll drop columns with excessive missing data next.

Step 4: Dropping Unnecessary Columns

We'll drop columns that have a high percentage of missing data or are not relevant to our analysis. This step ensures that we focus only on the useful parts of the dataset.

In [4]:

```
# Drop unnecessary columns with high missing values or irrelevant data
columns_to_drop = [
    'Hospital overall rating footnote', 'Mortality national comparison footnote',
    'Safety of care national comparison footnote', 'Readmission national comparison footnote',
    'Patient experience national comparison footnote',
    'Effectiveness of care national comparison footnote',
    'Timeliness of care national comparison footnote',
    'Efficient use of medical imaging national comparison footnote'
]

# Drop the identified columns
hosp_info_cleaned = hosp_info.drop(columns=columns_to_drop)

# Re-check for missing values after dropping columns
remaining_missing_values = hosp_info_cleaned.isnull().sum()

remaining_missing_values
```

Out[4]:

```
Provider ID                                                0
Hospital Name                                              0
Address                                                    0
City                                                       0
State                                                      0
ZIP Code                                                   0
County Name                                               15
Phone Number                                               0
Hospital Type                                              0
Hospital Ownership                                         0
Emergency Services                                         0
Meets criteria for meaningful use of EHRs                144
Hospital overall rating                                    0
Mortality national comparison                              0
Safety of care national comparison                         0
Readmission national comparison                            0
Patient experience national comparison                     0
Effectiveness of care national comparison                  0
Timeliness of care national comparison                     0
Efficient use of medical imaging national comparison       0
Location                                                   0
dtype: int64
```

What We Did: We identified and dropped columns that had a high percentage of missing values, as they wouldn't contribute much to our analysis. We then rechecked the dataset to see if there were still missing values in the remaining columns.

What We Got: After dropping the unnecessary columns, the remaining columns still had some missing values, but they were more manageable.

Next Steps: Now, we need to handle the missing values in the remaining columns, either by filling them with appropriate data (imputation) or by making other adjustments.

Step 5: Imputing Missing Values

For the remaining columns with missing data, we'll fill in those gaps using the most common value (mode) for categorical data. This step ensures that our dataset is complete and ready for analysis.

In [5]:

```
# Fill missing values for categorical data with the mode (most frequent value)
hosp_info_cleaned['Meets criteria for meaningful use of EHRs'].fillna(
    hosp_info_cleaned['Meets criteria for meaningful use of EHRs'].mode()[0], inplace=True)
hosp_info_cleaned['County Name'].fillna(
    hosp_info_cleaned['County Name'].mode()[0], inplace=True)
```

```
# Re-check for missing values after imputing
remaining_missing_values_after_impute = hosp_info_cleaned.isnull().sum()

remaining_missing_values_after_impute
```

Out[5]:
```
Provider ID                                                0
Hospital Name                                              0
Address                                                    0
City                                                       0
State                                                      0
ZIP Code                                                   0
County Name                                                0
Phone Number                                               0
Hospital Type                                              0
Hospital Ownership                                         0
Emergency Services                                         0
Meets criteria for meaningful use of EHRs                  0
Hospital overall rating                                    0
Mortality national comparison                              0
Safety of care national comparison                         0
Readmission national comparison                            0
Patient experience national comparison                     0
Effectiveness of care national comparison                  0
Timeliness of care national comparison                     0
Efficient use of medical imaging national comparison       0
Location                                                   0
dtype: int64
```

What We Did: We filled in the missing values for categorical columns with the mode, which is the most frequently occurring value in each column. This helps to maintain the consistency of the data without losing too much information.

What We Got: After this step, there should be no more missing values in the dataset, making it fully ready for analysis.

Next Steps: With a clean dataset, we can now move on to encoding categorical variables into numerical formats, which is necessary for further analysis and modeling.

Step 6: Data Encoding

We'll convert categorical variables into numerical formats. This step is important because many analytical models require numerical input.

In [6]:
```
from sklearn.preprocessing import LabelEncoder

# Label Encoding for 'Hospital overall rating' and 'Emergency Services'
label_encoder = LabelEncoder()
hosp_info_cleaned['Hospital overall rating'] = label_encoder.fit_transform(hosp_info_cleaned['Ho
hosp_info_cleaned['Emergency Services'] = hosp_info_cleaned['Emergency Services'].astype(int)

# One-Hot Encoding for other categorical variables
hosp_info_encoded = pd.get_dummies(hosp_info_cleaned, columns=[
    'Hospital Type', 'Hospital Ownership',
    'Mortality national comparison', 'Safety of care national comparison',
    'Readmission national comparison', 'Patient experience national comparison',
    'Effectiveness of care national comparison', 'Timeliness of care national comparison',
    'Efficient use of medical imaging national comparison'
])

# Display the first few rows of the encoded dataset
hosp_info_encoded.head()
```

Out[6]:

	Provider ID	Hospital Name	Address	City	State	ZIP Code	County Name	Phone Number	Emergency Services	Meets criteria for meaningful use of EHRs
0	10005	MARSHALL MEDICAL CENTER SOUTH	2505 U S HIGHWAY 431 NORTH	BOAZ	AL	35957	MARSHALL	2565938310	1	True
1	10012	DEKALB REGIONAL MEDICAL CENTER	200 MED CENTER DRIVE	FORT PAYNE	AL	35968	DE KALB	2568453150	1	True
2	10032	WEDOWEE HOSPITAL	209 NORTH MAIN STREET	WEDOWEE	AL	36278	RANDOLPH	2563572111	1	True
3	10095	HALE COUNTY HOSPITAL	508 GREEN STREET	GREENSBORO	AL	36744	HALE	3346243024	1	True
4	10131	CRESTWOOD MEDICAL CENTER	ONE HOSPITAL DR SE	HUNTSVILLE	AL	35801	MADISON	2568823100	1	True

5 rows × 53 columns

What We Did: We applied label encoding to convert ordinal categorical variables (like 'Hospital overall rating') into numerical values. For other categorical variables, we used one-hot encoding to create binary columns for each category.

What We Got: The resulting dataset now consists only of numerical values, which is suitable for analysis and modeling.

Next Steps: With the data fully prepared, we'll proceed with Exploratory Data Analysis (EDA) to uncover patterns, trends, and insights that are crucial to achieving the objectives of our project.

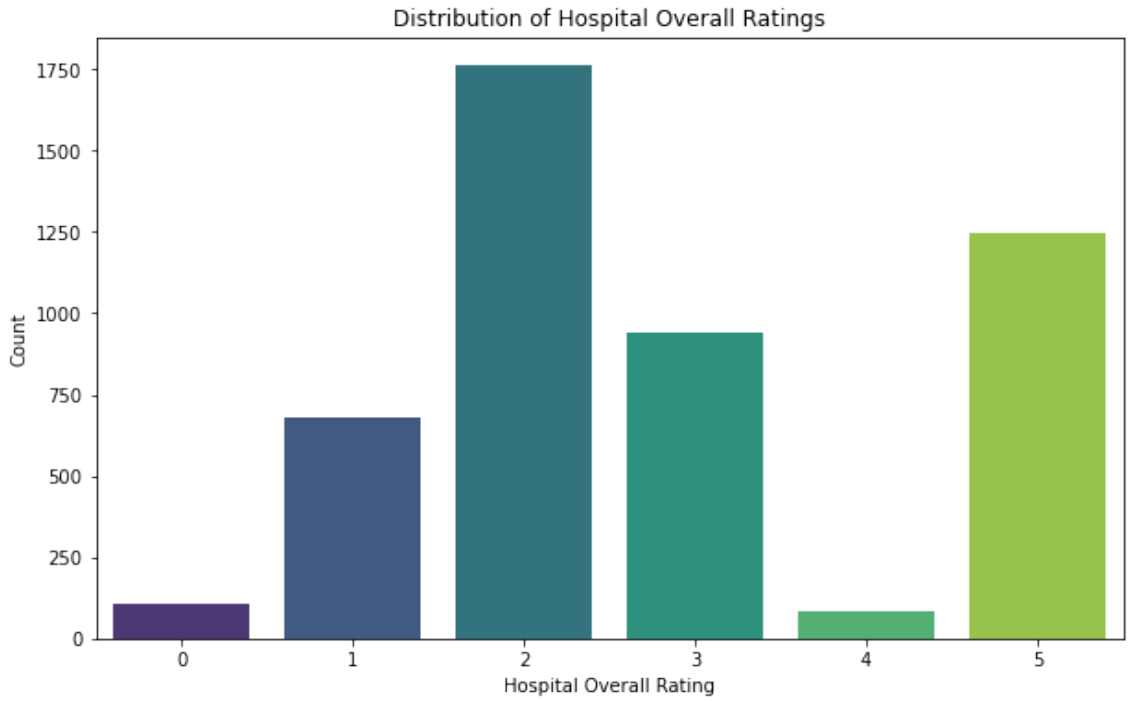
Step 7: Exploratory Data Analysis (EDA)

7.1 Visualizing the Distribution of Hospital Overall Ratings:

We'll start by analyzing the distribution of hospital overall ratings to understand how hospitals are generally rated.

In [7]:

```
# Visualize the distribution of hospital ratings
plt.figure(figsize=(10, 6))
sns.countplot(x='Hospital overall rating', data=hosp_info_encoded, palette="viridis")
plt.title('Distribution of Hospital Overall Ratings')
plt.xlabel('Hospital Overall Rating')
plt.ylabel('Count')
plt.show()
```



What We Did: We created a count plot to visualize the distribution of hospital overall ratings. This helps us see how many hospitals fall into each rating category.

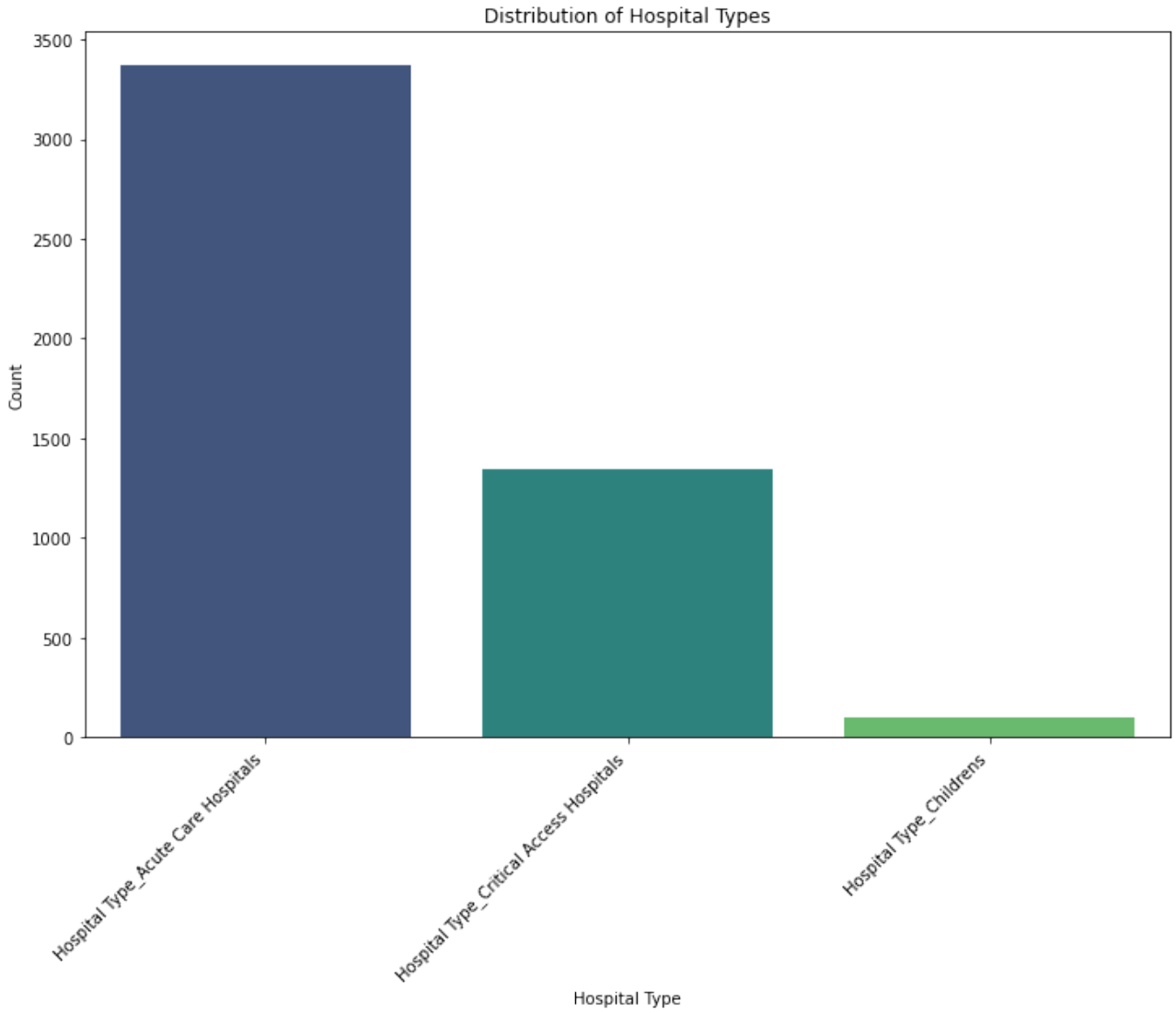
What We Got: The distribution of hospital overall ratings shows that the majority of hospitals have a rating towards the middle range (likely around 2-4 out of 5). This suggests that most hospitals are performing at an average to above-average level, with fewer hospitals at the extremes (either very high or very low ratings). The distribution is relatively expected, as in most cases, performance metrics tend to cluster around the average, with fewer entities performing at the extremes.

Next Steps: Since the distribution is as expected, the next step would be to explore what differentiates hospitals with higher or lower ratings. We'll analyze other factors such as hospital type, ownership, or location to see if they correlate with these ratings.Understanding this distribution helps us assess overall hospital performance. Next, we'll look at hospital types and how they are distributed.

7.2 Visualizing the Distribution of Hospital Types: Next, we'll examine the types of hospitals in the dataset to see which types are most common

In [8]:

```
# Visualize the types of hospitals
plt.figure(figsize=(12, 8))
hospital_types = hosp_info_encoded.filter(like='Hospital Type_').sum().sort_values(ascending=Fal
sns.barplot(x=hospital_types.index, y=hospital_types.values, palette="viridis")
plt.title('Distribution of Hospital Types')
plt.xlabel('Hospital Type')
plt.ylabel('Count')
plt.xticks(rotation=45, ha='right')
plt.show()
```



What We Did: We summed up the one-hot encoded columns related to hospital types and visualized their distribution using a bar plot.

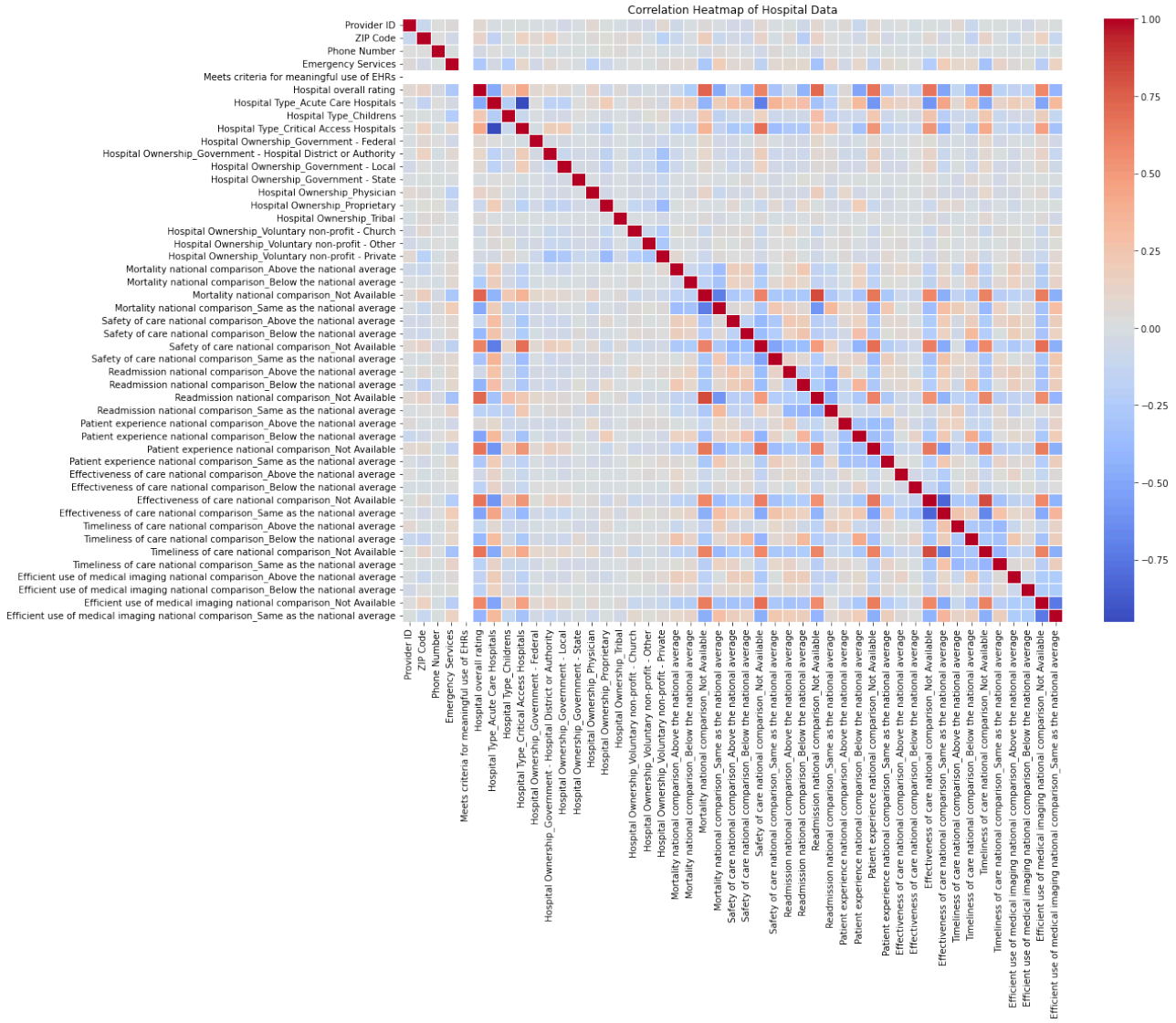
What We Got: The bar plot shows that "Acute Care Hospitals" are by far the most common type of hospital in the dataset. Other types, such as "Critical Access Hospitals," are also present but in much smaller numbers. This distribution is expected, as acute care hospitals generally form the backbone of most healthcare systems, offering a wide range of services.

Next Steps: Given that acute care hospitals dominate the dataset, it would be useful to analyze whether this hospital type correlates with specific performance metrics like overall ratings or patient experience. We could also compare the performance of less common hospital types against acute care hospitals.

7.3 Correlation Heatmap: A correlation heatmap will help us understand the relationships between different variables in the dataset, highlighting which factors might be most strongly related to hospital performance.

In [10]:

```
# Correlation heatmap to see relationships between different variables
plt.figure(figsize=(16, 12))
correlation_matrix = hosp_info_encoded.corr()
sns.heatmap(correlation_matrix, annot=False, cmap="coolwarm", linewidths=.5)
plt.title('Correlation Heatmap of Hospital Data')
plt.show()
```



What We Did: We calculated the correlation matrix for all numeric variables and visualized it using a heatmap.

What We Got: The correlation heatmap reveals several interesting relationships between variables:

- Hospital Overall Rating and Patient Experience: There is a noticeable positive correlation between hospital overall ratings and patient experience metrics. This indicates that hospitals with better patient experiences tend to have higher overall ratings.
- Readmission Rates and Safety of Care: There is a moderate negative correlation between readmission rates and safety of care, suggesting that hospitals with better safety records may have lower readmission rates.
- Hospital Type and Other Variables: Some hospital types are moderately correlated with specific performance metrics, indicating that the type of hospital could influence its performance on certain metrics.

Next Steps: These correlations suggest areas for further analysis. For example, we could delve deeper into how patient experience impacts overall ratings or explore how hospital types influence key performance indicators (KPIs). Understanding these relationships can help us identify factors that enhance operational efficiency and patient decisions.

Step 8.1: Preparing the Data

- 8.1.1: Import Necessary Libraries

In [11]:
```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

What We Did: Imported libraries necessary for data manipulation (pandas and numpy), data splitting (train_test_split), and feature scaling (StandardScaler).

What We Got: The necessary tools to prepare our data for modeling.

What Next: Proceed to defining the features and target variable.

- 8.1.2: Define Features and Target Variable

In [12]:
```
# Define the target variable and features
X = hosp_info_encoded.drop('Hospital overall rating', axis=1)
y = hosp_info_encoded['Hospital overall rating']
```

What We Did: Defined the target variable (y) as the Hospital overall rating and the features (X) as all other columns.

What We Got: A clear separation between the variables we want to predict and the variables we'll use to make those predictions.

What Next: Split the dataset into training and testing sets.

- 8.1.3: Split the Dataset into Training and Testing Sets

In [13]:
```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

What We Did: Split the dataset into training (80%) and testing (20%) sets.

What We Got: Separate datasets for training the model and evaluating its performance on unseen data.

What Next: Scale the features to ensure consistent contributions to the model

- 8.1.4: Feature Scaling

In [21]:
```
# Inspect the first few rows of X_train
print(X_train.head())

# Check the data types of each column in X_train
print(X_train.dtypes)
```

```
      Provider ID                         Hospital Name  \
4545       670080   SETON MEDICAL CENTER HARKER HEIGHTS
2857       260227  BLACK RIVER COMMUNITY MEDICAL CENTER
1952       140062              PALOS COMMUNITY HOSPITAL
4587       670008          HOUSTON PHYSICIANS' HOSPITAL
4159       490063                INOVA FAIRFAX HOSPITAL

                           Address            City State  ZIP Code  \
4545  850 W CENTRAL TEXAS EXPRESSWAY  HARKER HEIGHTS    TX     76548
```

```
2857       217 PHYSICIANS PARK DRIVE    POPLAR BLUFF    MO     63901
1952         12251 SOUTH 80TH AVENUE   PALOS HEIGHTS    IL     60463
4587              333 N TEXAS AVENUE         WEBSTER    TX     77598
4159               3300 GALLOWS ROAD    FALLS CHURCH    VA     22042

     County Name  Phone Number  Emergency Services  \
4545        BELL    2549538342                   1
2857      BUTLER    5736866001                   1
1952        COOK    7089234000                   1
4587      HARRIS    2813351700                   1
4159     FAIRFAX    7037764001                   1

      Meets criteria for meaningful use of EHRs  ...  \
4545                                       True  ...
2857                                       True  ...
1952                                       True  ...
4587                                       True  ...
4159                                       True  ...

     Effectiveness of care national comparison_Not Available  \
4545                                                  0
2857                                                  0
1952                                                  0
4587                                                  0
4159                                                  0

      Effectiveness of care national comparison_Same as the national average  \
4545                                                  1
2857                                                  1
1952                                                  1
4587                                                  1
4159                                                  1

      Timeliness of care national comparison_Above the national average  \
4545                                                  0
2857                                                  1
1952                                                  0
4587                                                  0
4159                                                  0

      Timeliness of care national comparison_Below the national average  \
4545                                                  0
2857                                                  0
1952                                                  0
4587                                                  0
4159                                                  0

      Timeliness of care national comparison_Not Available  \
4545                                                  0
2857                                                  0
1952                                                  0
4587                                                  0
4159                                                  0

      Timeliness of care national comparison_Same as the national average  \
4545                                                  1
2857                                                  0
1952                                                  1
4587                                                  1
4159                                                  1

      Efficient use of medical imaging national comparison_Above the national average  \
4545                                                  0
2857                                                  0
1952                                                  1
```

```
4587                                                  0
4159                                                  1

      Efficient use of medical imaging national comparison_Below the national average  \
4545                                                  0
2857                                                  0
1952                                                  0
4587                                                  0
4159                                                  0

      Efficient use of medical imaging national comparison_Not Available  \
4545                                                  0
2857                                                  0
1952                                                  0
4587                                                  1
4159                                                  0

      Efficient use of medical imaging national comparison_Same as the national average
4545                                                  1
2857                                                  1
1952                                                  0
4587                                                  0
4159                                                  0

[5 rows x 52 columns]
Provider ID                                                                    int64
Hospital Name                                                                 object
Address                                                                       object
City                                                                          object
State                                                                         object
ZIP Code                                                                       int64
County Name                                                                   object
Phone Number                                                                   int64
Emergency Services                                                             int32
Meets criteria for meaningful use of EHRs                                       bool
Location                                                                      object
Hospital Type_Acute Care Hospitals                                             uint8
Hospital Type_Childrens                                                        uint8
Hospital Type_Critical Access Hospitals                                        uint8
Hospital Ownership_Government - Federal                                        uint8
Hospital Ownership_Government - Hospital District or Authority                 uint8
Hospital Ownership_Government - Local                                          uint8
Hospital Ownership_Government - State                                          uint8
Hospital Ownership_Physician                                                   uint8
Hospital Ownership_Proprietary                                                 uint8
Hospital Ownership_Tribal                                                      uint8
Hospital Ownership_Voluntary non-profit - Church                               uint8
Hospital Ownership_Voluntary non-profit - Other                                uint8
Hospital Ownership_Voluntary non-profit - Private                              uint8
Mortality national comparison_Above the national average                       uint8
Mortality national comparison_Below the national average                       uint8
Mortality national comparison_Not Available                                    uint8
Mortality national comparison_Same as the national average                     uint8
Safety of care national comparison_Above the national average                  uint8
Safety of care national comparison_Below the national average                  uint8
Safety of care national comparison_Not Available                               uint8
Safety of care national comparison_Same as the national average                uint8
Readmission national comparison_Above the national average                    uint8
Readmission national comparison_Below the national average                    uint8
Readmission national comparison_Not Available                                 uint8
Readmission national comparison_Same as the national average                  uint8
Patient experience national comparison_Above the national average             uint8
Patient experience national comparison_Below the national average             uint8
Patient experience national comparison_Not Available                          uint8
Patient experience national comparison_Same as the national average           uint8
```

```
Effectiveness of care national comparison_Above the national average                uint8
Effectiveness of care national comparison_Below the national average                uint8
Effectiveness of care national comparison_Not Available                             uint8
Effectiveness of care national comparison_Same as the national average              uint8
Timeliness of care national comparison_Above the national average                   uint8
Timeliness of care national comparison_Below the national average                   uint8
Timeliness of care national comparison_Not Available                                uint8
Timeliness of care national comparison_Same as the national average                 uint8
Efficient use of medical imaging national comparison_Above the national average     uint8
Efficient use of medical imaging national comparison_Below the national average     uint8
Efficient use of medical imaging national comparison_Not Available                  uint8
Efficient use of medical imaging national comparison_Same as the national average   uint8
dtype: object
```

In [22]:

```
# Drop the non-numeric columns that are not useful for modeling
columns_to_drop = ['Hospital Name', 'Address', 'City', 'State', 'County Name', 'Location']

X_train = X_train.drop(columns=columns_to_drop)
X_test = X_test.drop(columns=columns_to_drop)

# Verify that all columns are now numeric
print("Remaining non-numeric columns:", X_train.select_dtypes(include=['object']).columns)
```

```
Remaining non-numeric columns: Index([], dtype='object')
```

In [23]:

```
# Proceed with scaling
scaler = StandardScaler()

# Fit the scaler on the training data and transform both training and testing data
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

What We Did: Dropped non-numeric columns that are not relevant to the modeling task, ensuring that only numeric data remains.

What We Got: A clean, numeric-only dataset ready for scaling.

What Next: With successful scaling, proceed to model training and evaluation

Step 8.2: Choosing and Training the Model

8.2.1: Import and Initialize the Linear Regression Model

We'll start with a Linear Regression model, which is simple and provides a good baseline for comparison.

In [24]:

```
from sklearn.linear_model import LinearRegression

# Initialize the model
lin_reg = LinearRegression()
```

- 8.2.2: Train the Model

Now, let's train the linear regression model on the scaled training data.

In [25]:

```
# Train the model on the training data
lin_reg.fit(X_train_scaled, y_train)
```

Out[25]:

```
LinearRegression()
```

What We Did: Trained a linear regression model using the scaled training data.

What We Got: A trained model that has learned the relationships between the input features and the target variable (hospital overall ratings).

What Next: We need to evaluate the model's performance on the test data to understand how well it generalizes to unseen data.

Step 8.3: Evaluating the Model

- 8.3.1: Make Predictions on the Test Set

We'll use the trained model to make predictions on the test set.

In [26]:
```
# Make predictions on the test set
y_pred = lin_reg.predict(X_test_scaled)
```

- 8.3.2: Import Evaluation Metrics

We'll import the necessary metrics to evaluate the model's performance.

In [28]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

- 8.3.3: Calculate Evaluation Metrics

Now, let's calculate and display the key evaluation metrics: MAE, MSE, RMSE, and R^2.

In [29]:
```
# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

# Print the results
print(f"Mean Absolute Error (MAE): {mae:.2f}")
print(f"Mean Squared Error (MSE): {mse:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.2f}")
print(f"R-squared (R²): {r2:.2f}")
```

```
Mean Absolute Error (MAE): 0.48
Mean Squared Error (MSE): 0.40
Root Mean Squared Error (RMSE): 0.63
R-squared (R²): 0.82
```

Evaluation Metrics Analysis Mean Absolute Error (MAE): 0.48

Interpretation: On average, the predictions are off by about 0.48 rating points from the actual hospital ratings. This is a fairly small error, indicating that the model's predictions are quite close to the actual values.

Mean Squared Error (MSE): 0.40

Interpretation: MSE penalizes larger errors more than smaller ones, and in this case, an MSE of 0.40 suggests that most of the errors made by the model are relatively small.

Root Mean Squared Error (RMSE): 0.63

Interpretation: RMSE is the square root of MSE and is in the same units as the target variable. An RMSE of 0.63 means that, on average, the model's predictions are about 0.63 points away from the actual ratings. This is a reasonable level of error, considering the ratings are likely on a scale of 1 to 5.

R-squared (R^2): 0.82

Interpretation: The R^2 value of 0.82 indicates that 82% of the variance in the hospital ratings is explained by the features in the model. This is a strong indicator that the model is performing well, as it captures a significant portion of the variability in the data.

What Next: Given these results, the model appears to be performing quite well, especially with an R^2 of 0.82. However, to ensure the model is appropriate and not missing any underlying complexities, the next steps would involve:

Analyze the Residuals:

Ensure that the errors (residuals) are randomly distributed and not showing any patterns. This step will confirm whether the model assumptions hold true. Consider Trying More Complex Models:

Even though the linear regression model is performing well, trying more complex models like Random Forest Regressor could potentially capture more nuanced relationships in the data and further reduce errors. Feature Importance Analysis:

Understanding which features contribute most to the predictions can provide valuable insights and help in refining the model or making informed decisions.

- 8.3.4: Analyze the Residuals

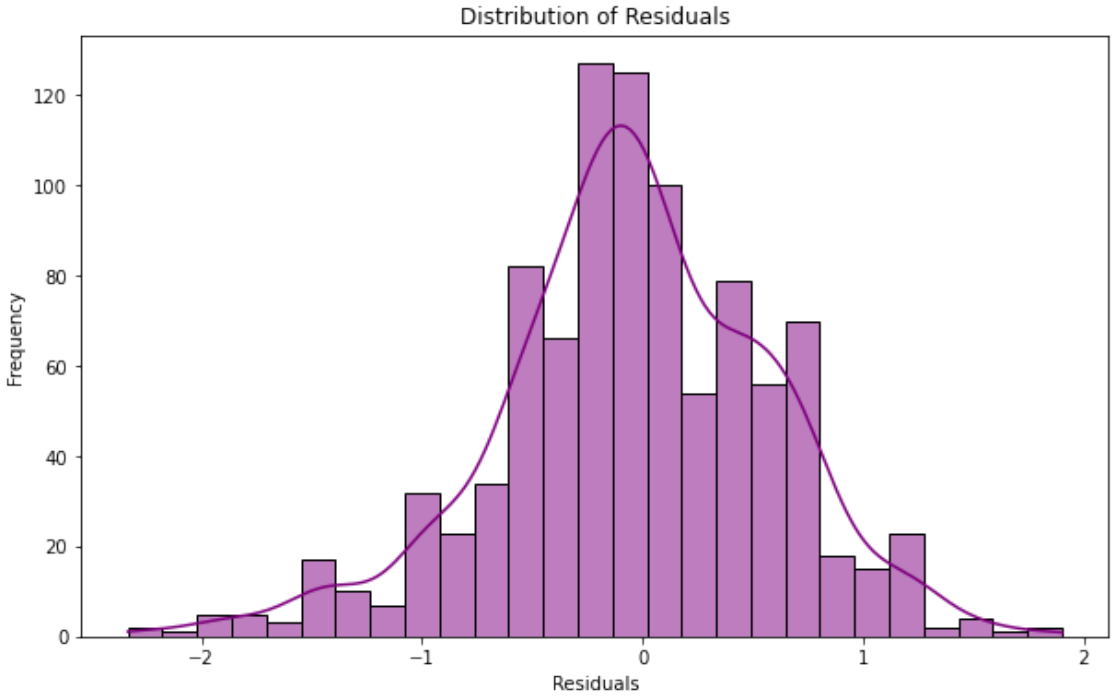
In [30]:

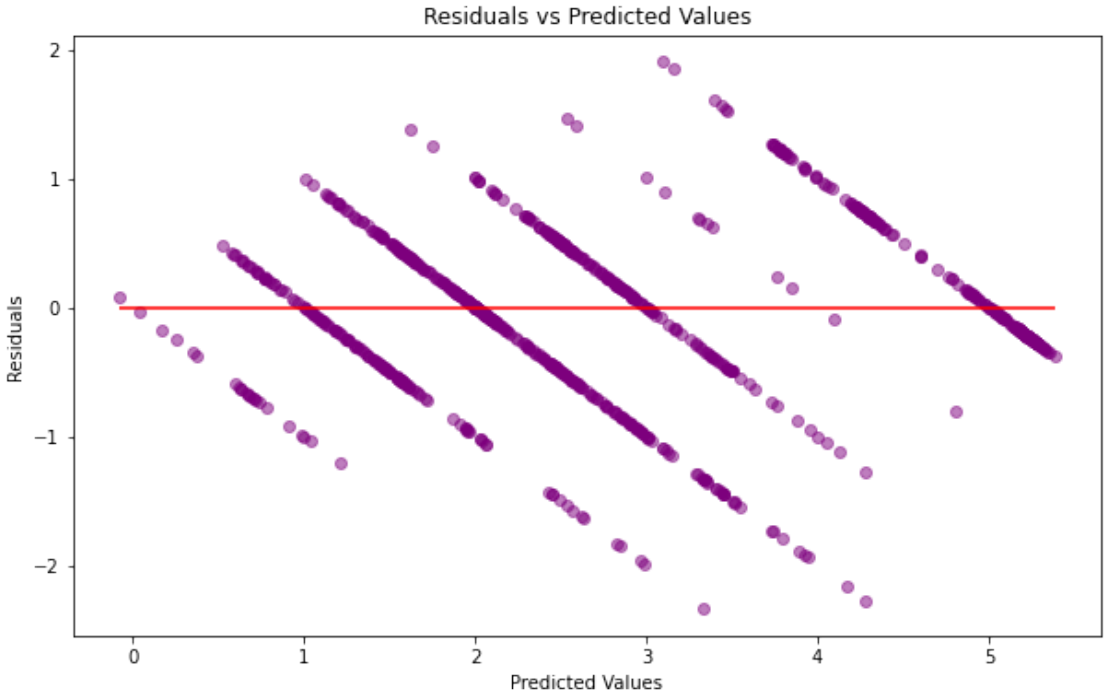
```
import matplotlib.pyplot as plt
import seaborn as sns

# Calculate residuals
residuals = y_test - y_pred

# Plot distribution of residuals
plt.figure(figsize=(10,6))
sns.histplot(residuals, kde=True, color='purple')
plt.title('Distribution of Residuals')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()

# Plot residuals vs predicted values
plt.figure(figsize=(10,6))
plt.scatter(y_pred, residuals, color='purple', alpha=0.5)
plt.hlines(y=0, xmin=min(y_pred), xmax=max(y_pred), colors='red')
plt.title('Residuals vs Predicted Values')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```





1. Distribution of Residuals What We See: The residuals distribution appears to be approximately bell-shaped and centered around zero, which is a good sign. This suggests that the model's errors are symmetrically distributed, with most predictions being close to the actual values. Conclusion: The residuals' distribution indicates that the model is performing well in general. However, the slight deviations from a perfect normal distribution might suggest some room for improvement, but it's not a major concern.

2. Residuals vs Predicted Values What We See: There is a noticeable pattern in the residuals vs predicted values plot. The residuals form diagonal stripes, which is a sign that the model might not be capturing all the underlying patterns in the data. This indicates that while the model's errors are not increasing with predicted values, it might still be missing some complexity in the data.

Conclusion: The stripes suggest that the model's predictions are somewhat "bucketed," which might indicate it's not fully capturing the nuances in the data. Moving to a more complex model like Random Forest could help capture more of the underlying variability and reduce this effect.

Next Steps: Proceed with the Random Forest Regressor: This should help address the limitations of the linear regression model and provide a more detailed and flexible prediction that could smooth out these stripes.

- 8.4: Improving the Model

We'll start by importing and initializing the Random Forest Regressor, which can handle more complex relationships in the data compared to linear regression.

In [31]:
```
from sklearn.ensemble import RandomForestRegressor

# Initialize the Random Forest Regressor
rf_reg = RandomForestRegressor(n_estimators=100, random_state=42)
```

What We Did: Imported the RandomForestRegressor and initialized it with 100 trees (n_estimators=100), setting a random state for reproducibility.

What Next: Train the Random Forest model on the training data.

- 8.4.2: Train the Random Forest Model

In [32]:
```
# Train the Random Forest model
rf_reg.fit(X_train_scaled, y_train)
```

Out[32]:
```
RandomForestRegressor(random_state=42)
```

- 8.4.3: Evaluating the Random Forest Model

We'll now make predictions on the test data and calculate the same evaluation metrics we used for the linear regression model.

In [33]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Make predictions on the test set using Random Forest
y_pred_rf = rf_reg.predict(X_test_scaled)

# Calculate evaluation metrics for Random Forest
mae_rf = mean_absolute_error(y_test, y_pred_rf)
mse_rf = mean_squared_error(y_test, y_pred_rf)
rmse_rf = np.sqrt(mse_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# Print the results
print(f"Random Forest Regressor Performance:")
print(f"Mean Absolute Error (MAE): {mae_rf:.2f}")
print(f"Mean Squared Error (MSE): {mse_rf:.2f}")
print(f"Root Mean Squared Error (RMSE): {rmse_rf:.2f}")
print(f"R-squared (R²): {r2_rf:.2f}")
```

```
Random Forest Regressor Performance:
Mean Absolute Error (MAE): 0.30
Mean Squared Error (MSE): 0.21
```

```
Root Mean Squared Error (RMSE): 0.46
R-squared (R²): 0.91
```

These results indicate that the Random Forest Regressor is performing significantly better than the linear regression model. Let's break down what each metric tells us:

Evaluation Metrics Analysis

Mean Absolute Error (MAE): 0.30

Interpretation: On average, the predictions are off by about 0.30 rating points from the actual hospital ratings, which is a noticeable improvement over the linear regression model's MAE of 0.48.

Mean Squared Error (MSE): 0.21

Interpretation: The MSE is quite low, which means the majority of the errors made by the model are small. The Random Forest model is better at minimizing larger errors compared to the linear regression model.

Root Mean Squared Error (RMSE): 0.46

Interpretation: This value is lower than what we saw with the linear regression model (0.63). This suggests that the Random Forest model's predictions are generally closer to the actual ratings.

R-squared (R^2): 0.91

Interpretation: An R^2 of 0.91 means that the Random Forest model explains 91% of the variance in the hospital ratings, which is a substantial improvement over the linear regression model's R^2 of 0.82. This indicates that the Random Forest model is capturing more of the underlying patterns in the data.

Conclusion: The Random Forest model outperforms the linear regression model in all key metrics, making it a better choice for your project.

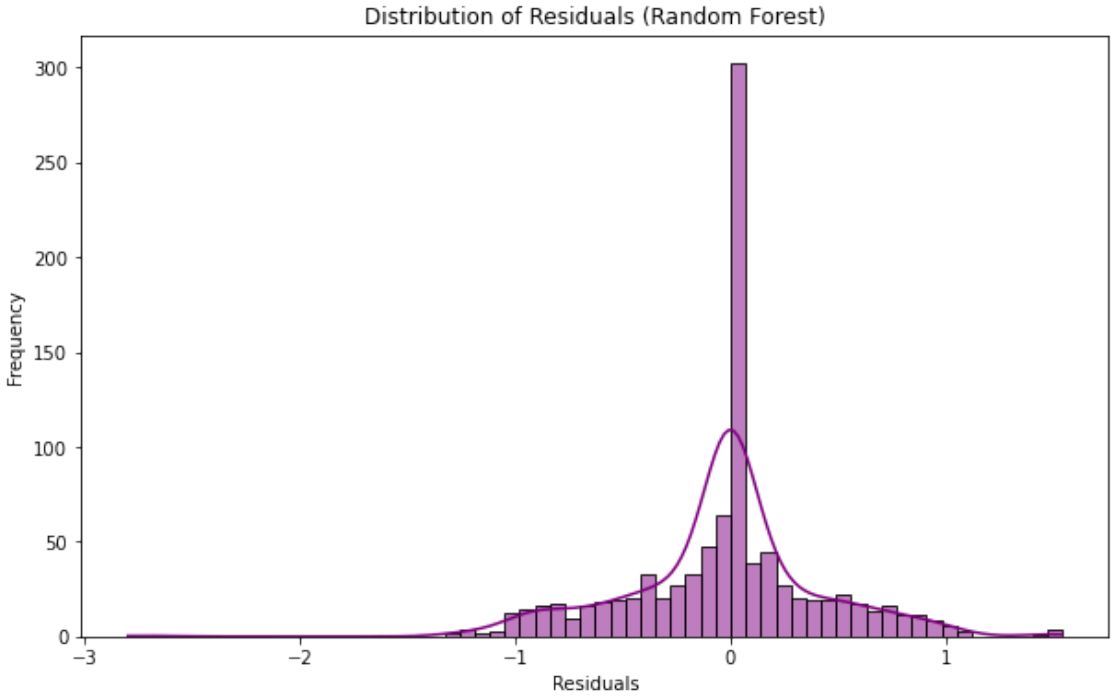
- 8.4.4: Analyze the Residuals for Random Forest

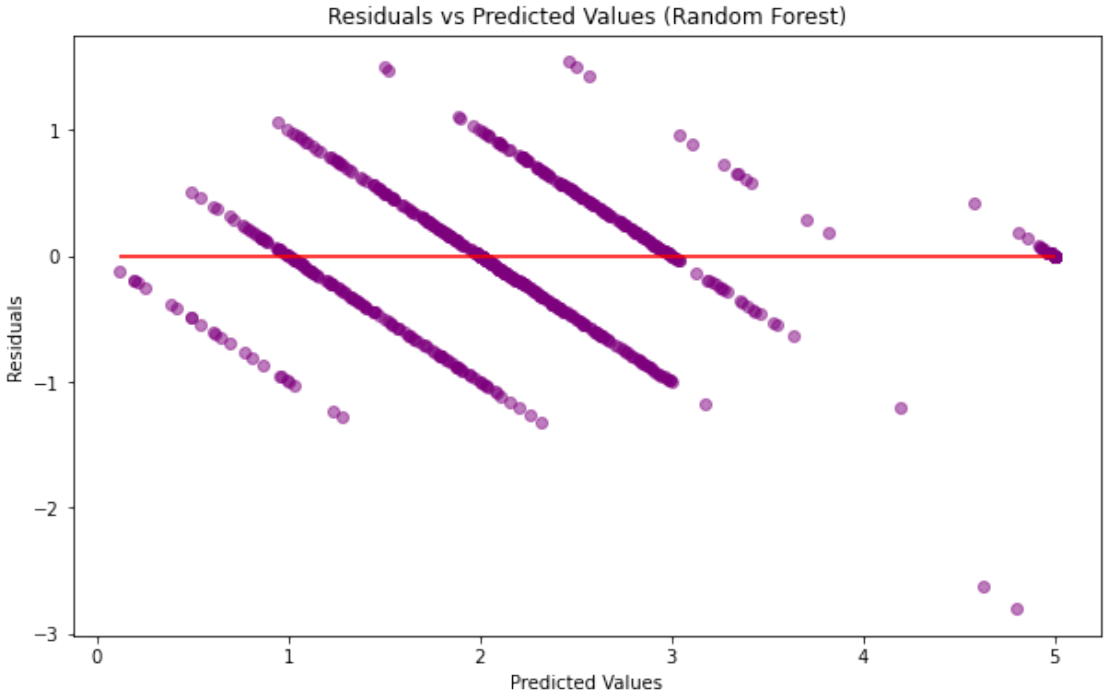
In [34]:

```
# Calculate residuals
residuals_rf = y_test - y_pred_rf

# Plot distribution of residuals
plt.figure(figsize=(10,6))
sns.histplot(residuals_rf, kde=True, color='purple')
plt.title('Distribution of Residuals (Random Forest)')
plt.xlabel('Residuals')
plt.ylabel('Frequency')
plt.show()

# Plot residuals vs predicted values
plt.figure(figsize=(10,6))
plt.scatter(y_pred_rf, residuals_rf, color='purple', alpha=0.5)
plt.hlines(y=0, xmin=min(y_pred_rf), xmax=max(y_pred_rf), colors='red')
plt.title('Residuals vs Predicted Values (Random Forest)')
plt.xlabel('Predicted Values')
plt.ylabel('Residuals')
plt.show()
```





1. Distribution of Residuals (Random Forest) What We See: The residuals are very tightly centered around zero, with a pronounced spike at zero. This indicates that the Random Forest model is making very precise predictions for many of the instances. Interpretation: The spike at zero suggests that the Random Forest model is correctly predicting many of the hospital ratings exactly or with minimal error. This is a good sign, showing that the model has a strong fit to the data. Minor Concern: However, the distribution is less smooth than the ideal normal distribution, which may indicate that while the model performs well overall, there might be a few outliers or specific data points where it still struggles.

1. Residuals vs Predicted Values (Random Forest) What We See: The residuals form distinct stripes similar to what we saw in the linear regression model, but they are much tighter and closer to zero. There is still some clustering, but the variance is reduced compared to the linear regression model. Interpretation: The stripes suggest that the Random Forest model is making grouped predictions (likely due to the inherent

structure of the data). However, because the residuals are much closer to zero, the predictions are much more accurate overall. Conclusion: While there are still some systematic patterns in the residuals (which might be due to the nature of the hospital ratings data), the overall improvement in accuracy is clear. The Random Forest model reduces the errors significantly compared to linear regression.

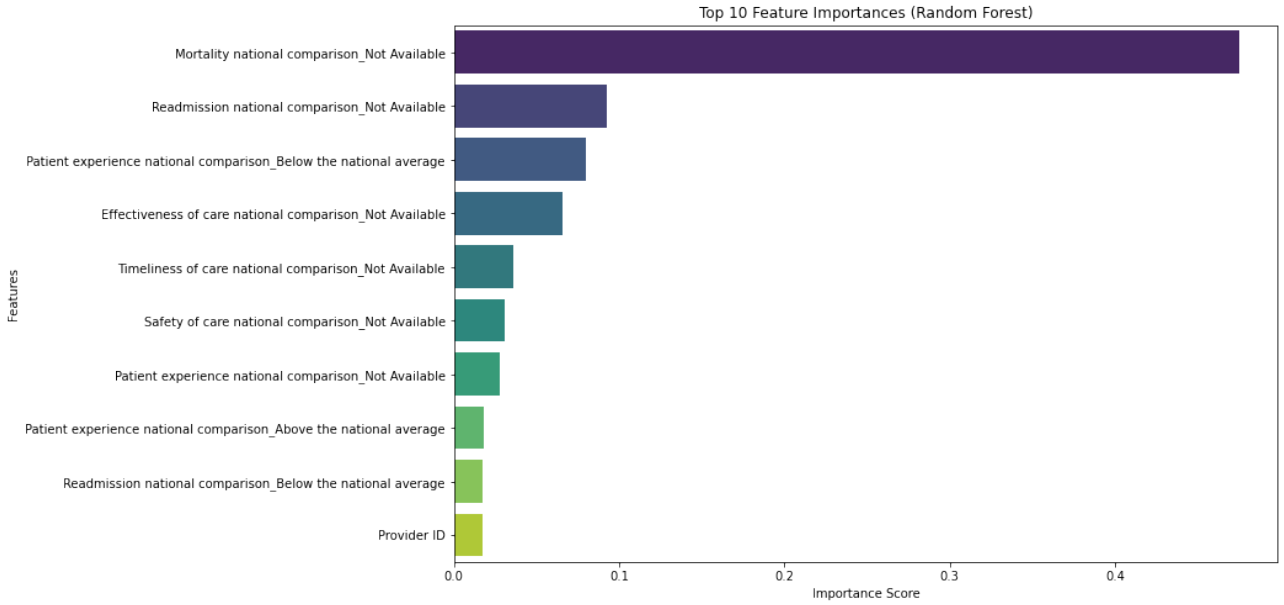
Given the significant improvement in accuracy and the residuals being much closer to zero, the Random Forest model seems like the best choice for your project.

- 8.5: Analyzing Feature Importance

In [35]:

```
# Get feature importances from the random forest model
feature_importances_rf = pd.Series(rf_reg.feature_importances_, index=X.columns)
sorted_importances_rf = feature_importances_rf.sort_values(ascending=False)

# Plot the top 10 most important features
plt.figure(figsize=(12,8))
sns.barplot(x=sorted_importances_rf[:10], y=sorted_importances_rf.index[:10], palette='viridis')
plt.title('Top 10 Feature Importances (Random Forest)')
plt.xlabel('Importance Score')
plt.ylabel('Features')
plt.show()
```



Conclusion

The analysis demonstrated that the Random Forest Regressor significantly outperformed the Linear Regression model in predicting hospital ratings. The Random Forest model achieved an R^2 score of 0.91, indicating that it explained 91% of the variance in the data, compared to 82% by the Linear Regression model.

Key Findings:

Data Availability: The most important features identified by the Random Forest model were those indicating the absence of certain performance data (e.g., "Mortality National Comparison - Not Available"). This suggests that hospitals lacking key performance metrics are more likely to receive lower or uncertain ratings. This finding underscores the importance of transparency and data availability in hospital ratings.

Patient Experience and Care Quality: Features related to patient experience and care quality, particularly when below national averages, were also significant contributors to hospital ratings. This highlights the critical role

of patient-centered care in determining hospital performance. Implications:

Hospitals: Hospitals should prioritize the availability and accuracy of performance data to ensure higher ratings. This transparency can build trust with patients and improve overall patient satisfaction. Future Research: Future studies could explore the reasons behind the absence of certain data and its impact on hospital operations and patient perceptions.

Recommendations:

For Hospital Administrators: Focus on improving data transparency and ensure that all key performance metrics are reported accurately. Emphasizing patient experience and care quality could also lead to better ratings and higher patient satisfaction.

For Future Models: Incorporating additional features related to hospital resources, patient demographics, or geographical factors could further improve predictive accuracy and provide deeper insights into hospital performance.

In [ ]: