Hospital Inpatient Discharge Project

August 20, 2025

Problem Setting In the mosaic of New York's healthcare system, each hospital stay represents a convergence of medical necessity, resource availability, and patient circumstances. Length of stay (LOS) serves as a critical indicator of hospital efficiency, patient care quality, and the broader challenges within the healthcare system. However, beneath the surface of aggregated metrics lies a complex tapestry of disparities influenced by socioeconomic status, demographic factors, and geographic location. While hospitals strive to optimize resources and reduce costs, patients from marginalized communities may face extended stays, not merely because of their clinical condition but due to systemic barriers in accessing timely and effective care. The length of stay, therefore, is more than a logistical challenge, it's a mirror reflecting the inequities in our healthcare system. This project aims to harness data driver methodologies to predict LOS in New York hospitals while critically examining the disparities that potentially shape patient outcomes. By blending predictive analytics with an exploration of healthcare inequities, I aspire to uncover actionable insights that enhance patient care and advance health equity.

Project Definition The length of a patient's hospital stay is determined by more than just their medical condition. For some, it is dictated by the severity of their illness or the complexity of their care. For others, it is shaped by social and economic barriers; a lack of access to follow-up care, inadequate health insurance, or systemic inequities that have long gone unaddressed. In a state as diverse as New York, these disparities are woven into the fabric of healthcare system, creating unequal outcomes for patients from different backgrounds. This project seeks to investigate the factors influencing hospital length of stay and predict LOS using clinical, demographic, and socioeconomic variables. By doing so, I aim to answer critical questions that go beyond predictive modeling and delve into the systemic challenges faced by hospitals and their patients: 1. **KeyPredictors**:what clinical, demographic, and socioeconomi cfactors more strongly influence length of stay in New York hospitals? 2. **Disparities**:Are there significant disparities in LOS based on race, ethnicity, insurance type, or geographic location?

3. **GeographicPatterns**:Do patients from certain regions or zipcodes experience systematically different outcomes, and how are these patterns influenced by socioeconomic conditions?

4. **InterventionStrategies**:How can predictive insights help hospitals implement targeted interventions to reduce LOS and address inequities?

This project is not just about building a predictive model—it's about uncovering the hidden stories within the data. By identifying the driver of LOS and highlighting them, I aim to provide actionable insights for hospital administrators and public health officials, alike. Whether it's optimizing bed management, designing targeted interventions, or addressing geographic disparities, the goal is to transform data into strategies that advance both efficiency and equity in healthcare.

Data Source: New York State Department of Health. (2017). Hospital Inpatient Discharges (SPARCS De- Identified): 2017. Retrieved from https://health.data.ny.gov/dataset/Hospital-Inpatient- Discharges-SPARCS-De-Identified/22g3-z7e7 https://www.kaggle.com/datasets/thedevastator/2010-new-york-state-hospital-inpatient-discharge

Data Description This dataset chronicles the diverse experiences of patients across New York's hospitals, encapsulating their journeys from admission to discharge. Each entry provides a detailed snapshot of patient demographics, the clinical complexity of their conditions, and the outcome of their hospital stays. By capturing data on insurance type and geographic indicators like zip code, the dataset also serves as a lens into the social determinants influencing health outcomes.

Methodology:

1. **Exploratory Data Analysis (EDA)** • Visualize LOS distributions across demographics (age, race, insurance). Analyze the correlation between clinical factors (severity, diagnosis) and LOS. Identify initial indications of disparities using subgroup comparisons.
2. **Predictive Modeling for Length of Stay** • Objective: Predict the length of stay using clinical, demographic, and socioeconomic data.
3. **Disparity Assessment** • Analyze residuals from predictive models to detect possible systematic biases against specific groups. • Use statistical tests to assess differences in LOS by race, zip code, and insurance type, controlling for clinical severity.
4. **Geographic Analysis** • Map LOS by zip code and health service area to identify geographic disparities. • Correlate geographic patterns with socioeconomic indicators (e.g., median income).
5. **Ethical Considerations** • Ensure model fairness by examining prediction accuracy across demographic groups. • Adhere to HIPAA guidelines to maintain patient privacy and data confidentiality.

Impact By merging predictive analytics with an exploration of healthcare disparities, this project empowers hospitals to optimize patient flow while addressing the underlying inequities that extend hospital stays for marginalized populations. The findings will not only support operational improvements but also inform policy initiatives aimed at fostering a more equitable healthcare system. In essence, this project transforms the measurement of length of stay into a catalyst for change—illuminating the path toward a healthcare system where efficiency and equity are not mutually exclusive but mutually reinforcing.

[190]:
```python
from sklearn.ensemble import RandomForestClassifier
```

[3]:
```python
import numpy as np
import pandas as pd
import random as rnd

import seaborn as sns
import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
```

Environment Setup for Data Analysis and Machine Learning This code sets up the environment for data analysis, visualization, and machine learning modeling.

- **Importing Essential Libraries**
  - `numpy` & `pandas`: Used for data manipulation, cleaning, and preprocessing.

  - `random`: Allows for the use of random number generation (though not used in this snippet).
- **Data Visualization**
  - `seaborn` & `matplotlib.pyplot`: Essential for creating statistical plots and visualizing trends in data.

  - `%matplotlib inline`: Ensures that plots are displayed within the Jupyter Notebook rather than in a separate window.
- **Machine Learning Libraries**
  - `sklearn.ensemble.RandomForestClassifier`: Implements Random Forest, an ensemble learning method useful for both classification and regression.

  - `sklearn.linear_model.LogisticRegression`: A fundamental algorithm for binary classification problems.

  - `sklearn.svm.SVC` & `LinearSVC`: Implements Support Vector Machines (SVM), useful for classification tasks with linear and non-linear decision boundaries.

[4]:
```
import pandas as pd

file_path = "/Users/desireereid/Downloads/hospital inpatient discharges.csv"

chunk_size = 100000

chunks = []
for chunk in pd.read_csv(file_path, chunksize=chunk_size):
    chunks.append(chunk)

df = pd.concat(chunks, ignore_index=True)

df.head()
```

```
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (29) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
```

```
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11,29) have mixed types. Specify dtype option on import
or set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11,29) have mixed types. Specify dtype option on import
or set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,29) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11,29) have mixed types. Specify dtype option on import
or set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11,29) have mixed types. Specify dtype option on import
or set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,29) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11,29) have mixed types. Specify dtype option on import
```

```
or set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (7,11) have mixed types. Specify dtype option on import or
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2823727821.py:8:
DtypeWarning: Columns (11) have mixed types. Specify dtype option on import or
```

```
set low_memory=False.
  for chunk in pd.read_csv(file_path, chunksize=chunk_size):
```

[4]:

```
   index Health Service Area Hospital County  Operating Certificate Number  \
0      0      Capital/Adiron          Albany                      101000.0
1      1      Capital/Adiron          Albany                      101000.0
2      2      Capital/Adiron          Albany                      101000.0
3      3      Capital/Adiron          Albany                      101000.0
4      4      Capital/Adiron          Albany                      101000.0

   Facility ID                   Facility Name    Age Group  \
0          1.0  Albany Medical Center Hospital     18 to 29
1          1.0  Albany Medical Center Hospital     50 to 69
2          1.0  Albany Medical Center Hospital  70 or Older
3          1.0  Albany Medical Center Hospital  70 or Older
4          1.0  Albany Medical Center Hospital  70 or Older

  Zip Code - 3 digits Gender   Race  … Source of Payment 2  \
0                 NaN      F  White  …            Self-Pay
1                 NaN      M  White  …            Medicare
2                 NaN      M  White  …          Blue Cross
3                 NaN      F  White  …   Insurance Company
4                 NaN      F  White  …   Insurance Company

  Source of Payment 3 Attending Provider License Number  \
0                 NaN                            3623.0
1          Blue Cross                          216951.0
2            Self-Pay                            3076.0
3            Self-Pay                          140796.0
4            Self-Pay                          170254.0

  Operating Provider License Number  Other Provider License Number  \
0                               NaN                            NaN
1                          216951.0                            NaN
2                               NaN                            NaN
3                          140796.0                       170254.0
4                          170254.0                       170254.0

   Birth Weight Abortion Edit Indicator  Emergency Department Indicator  \
0             0                      N                               Y
1             0                      N                               Y
2             0                      N                               N
3             0                      N                               N
4             0                      N                               N

  Total Charges  Total Costs
0       4476.23      1672.65
```

```
1     148612.34     51414.70
2      16561.99      4032.49
3      13593.51      4947.81
4      31962.58     16988.33

[5 rows x 38 columns]
```

Data Loading with Chunking This code efficiently loads a large CSV file containing hospital inpatient discharge records using chunking to prevent memory overload.

- Reads data in chunks of 100,000 rows to optimize RAM usage.
- Stores chunks in a list and later combines them into a single Pandas DataFrame.
- Displays the first five rows to verify successful loading.

Using chunking ensures scalability and efficient memory management when working with large datasets.

[5]:

```python
import pandas as pd

file_path = "/Users/desireereid/Downloads/hospital inpatient discharges.csv"

df = pd.read_csv(file_path, low_memory=False)

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2622133 entries, 0 to 2622132
Data columns (total 38 columns):
 #   Column                               Dtype
---  ------                               -----
 0   index                                int64
 1   Health Service Area                  object
 2   Hospital County                      object
 3   Operating Certificate Number         float64
 4   Facility ID                          float64
 5   Facility Name                        object
 6   Age Group                            object
 7   Zip Code - 3 digits                  object
 8   Gender                               object
 9   Race                                 object
 10  Ethnicity                            object
 11  Length of Stay                       object
 12  Type of Admission                    object
 13  Patient Disposition                  object
 14  Discharge Year                       int64
 15  CCS Diagnosis Code                   float64
```

```
 16  CCS Diagnosis Description            object
 17  CCS Procedure Code                   float64
 18  CCS Procedure Description            object
 19  APR DRG Code                         int64
 20  APR DRG Description                  object
 21  APR MDC Code                         int64
 22  APR MDC Description                  object
 23  APR Severity of Illness Code         int64
 24  APR Severity of Illness Description  object
 25  APR Risk of Mortality                object
 26  APR Medical Surgical Description     object
 27  Source of Payment 1                  object
 28  Source of Payment 2                  object
 29  Source of Payment 3                  object
 30  Attending Provider License Number    float64
 31  Operating Provider License Number    float64
 32  Other Provider License Number        float64
 33  Birth Weight                         int64
 34  Abortion Edit Indicator              object
 35  Emergency Department Indicator       object
 36  Total Charges                        float64
 37  Total Costs                          float64
dtypes: float64(9), int64(6), object(23)
memory usage: 760.2+ MB
None
```

Loading the Dataset This code loads the hospital inpatient discharge dataset into a Pandas DataFrame for analysis.

- Defines the file path for the CSV dataset.
- Reads the dataset using `pd.read_csv()`, setting `low_memory=False` to handle mixed data types efficiently.
- Prints dataset information using `df.info()` to check column types, missing values, and overall structure.

Summary of Output:
- The dataset contains **2,622,133 records** and **38 columns**.
- **Data types:**
- **23 object (string) columns** (e.g., `Health Service Area`, `Race`, `Ethnicity`, `Source of Payment`).
- **9 float columns** (e.g., `Operating Certificate Number`, `Total Charges`, `Total Costs`).
- **6 integer columns** (e.g., `Discharge Year`, `APR DRG Code`, `Birth Weight`).
- **Potential Data Issues:**
- `Length of Stay` and `Zip Code - 3 digits` are stored as **objects** instead of numerical types.
- **Some numeric columns (e.g., `Operating Certificate Number`) are float instead of integer**, indicating possible missing values or formatting inconsistencies.

- `Source of Payment 3` and other categorical fields may contain unexpected entries requiring further cleaning.

This step ensures I understand the **dataset structure and potential inconsistencies** before further analysis.

[6]:
```python
mixed_cols = ["Zip Code - 3 digits", "Length of Stay", "Source of Payment 3"]

for col in mixed_cols:
    print(f"\n  Checking column: {col}")
    print(df[col].apply(type).value_counts())
    print(df[col].head(10))
    print("-" * 50)
```

```
  Checking column: Zip Code - 3 digits
Zip Code - 3 digits
<class 'str'>      2618932
<class 'float'>       3201
Name: count, dtype: int64
0      NaN
1      NaN
2      NaN
3      NaN
4      NaN
5    100.0
6    100.0
7    100.0
8    100.0
9    100.0
Name: Zip Code - 3 digits, dtype: object
--------------------------------------------------

  Checking column: Length of Stay
Length of Stay
<class 'str'>    2622133
Name: count, dtype: int64
0     1
1    15
2     3
3     5
4     4
5     3
6     4
7     1
8     1
9     2
Name: Length of Stay, dtype: object
```

```
--------------------------------------------------

 Checking column: Source of Payment 3
Source of Payment 3
<class 'float'>    2040841
<class 'str'>       581292
Name: count, dtype: int64
0           NaN
1    Blue Cross
2      Self-Pay
3      Self-Pay
4      Self-Pay
5           NaN
6           NaN
7           NaN
8           NaN
9           NaN
Name: Source of Payment 3, dtype: object
--------------------------------------------------
```

Identifying Mixed Data Types This code checks for columns with mixed data types, which can cause issues during analysis and modeling.

- Defines a list of columns suspected to contain mixed data types.
- Iterates through each column, printing:
  - The count of different data types present.
  - The first 10 values to inspect inconsistencies.

Summary of Output:
- `Zip Code - 3 digits` contains **strings, integers, and floats**:
- **Majority (2,006,496)** are stored as **strings**, while **559,904 are integers** and **55,733 are floats**.
- Some values are **missing (`NaN`)** and others appear as decimal numbers (likely due to formatting issues).

- **`Length of Stay`** contains **both strings and integers**:
  - **2,514,112 values** are **strings**, while **108,021 are integers**.
  - LOS should be a numerical column, so this indicates potential formatting inconsistencies.
- **`Source of Payment 3`** has **a mix of floats and strings**:
  - **2,040,841 values are floats (`NaN` or missing values)**.
  - **581,292 are strings**, indicating different types of payment sources.
  - Cleaning may involve replacing missing values and standardizing categories.

This step helps identify columns requiring data type conversion and cleaning before further processing.

[7]:
```
df["Zip Code - 3 digits"] = df["Zip Code - 3 digits"].astype(str).str.split('.
↪').str[0].replace("nan", np.nan)
df.to_csv("/Users/desireereid/Downloads/cleaned_hospital_data.csv", index=False)
```

Cleaning and Standardizing Zip Code Data This code ensures consistency in the `Zip Code - 3 digits` column by:

- Converting all values to strings.
- Removing unintended float formatting by splitting on the decimal point.
- Replacing `"nan"` with `NaN` for proper missing value handling.
- Saving the cleaned dataset as `"cleaned_hospital_data.csv"`.

[8]:
```
import pandas as pd

cleaned_df = pd.read_csv("/Users/desireereid/Downloads/cleaned_hospital_data.
↪csv")

cleaned_df["Length of Stay"] = pd.to_numeric(cleaned_df["Length of Stay"],␣
↪errors="coerce")

cleaned_df.to_csv("/Users/desireereid/Downloads/cleaned_hospital_data.csv",␣
↪index=False)

print("  Cleaned dataset saved successfully as 'cleaned_hospital_data.csv'")
```

```
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/2843498253.py:4:
DtypeWarning: Columns (1,2,7,11,29) have mixed types. Specify dtype option on
import or set low_memory=False.
  cleaned_df =
pd.read_csv("/Users/desireereid/Downloads/cleaned_hospital_data.csv")
```

```
 Cleaned dataset saved successfully as 'cleaned_hospital_data.csv'
```

Ensuring Numeric Format for Length of Stay This code reloads the cleaned dataset and ensures `Length of Stay` is properly formatted as numeric data.

- Reads the cleaned dataset from `"cleaned_hospital_data.csv"`.
- Converts `Length of Stay` to numeric, coercing errors to `NaN` if necessary.
- Saves the updated dataset back to `"cleaned_hospital_data.csv"`.
- Prints a confirmation message upon successful saving.

[9]:
```
df["Source of Payment 3"] = df["Source of Payment 3"].astype(str).
↪replace("nan", np.nan)
df.to_csv("/Users/desireereid/Downloads/cleaned_hospital_data.csv", index=False)
```

Standardizing Source of Payment Data This code ensures consistency in the `Source of Payment 3` column by:

- Converting all values to strings to handle mixed data types.
- Replacing `"nan"` with `NaN` for proper missing value recognition.
- Saving the updated dataset as `"cleaned_hospital_data.csv"`.

[10]:
```
print(df[["Zip Code - 3 digits", "Length of Stay", "Source of Payment 3"]].
↪dtypes)
print(df[["Zip Code - 3 digits", "Length of Stay", "Source of Payment 3"]].
↪head(10))
```

```
Zip Code - 3 digits    object
Length of Stay         object
Source of Payment 3    object
dtype: object
  Zip Code - 3 digits Length of Stay Source of Payment 3
0                 NaN              1                 NaN
1                 NaN             15          Blue Cross
2                 NaN              3            Self-Pay
3                 NaN              5            Self-Pay
4                 NaN              4            Self-Pay
5                 100              3                 NaN
6                 100              4                 NaN
7                 100              1                 NaN
8                 100              1                 NaN
9                 100              2                 NaN
```

[11]:
```
column_descriptions = {
    "index": "Unique row index identifier. (Integer)",
    "Health Service Area": "Geographic region of the hospital. (String)",
    "Hospital County": "County in which the hospital is located. (String)",
    "Operating Certificate Number": "Unique identifier for hospitals. (Float)",
    "Facility ID": "Unique identifier for the healthcare facility. (Float)",
    "Facility Name": "Name of the hospital. (String)",
    "Age Group": "Age group of the patient (e.g., 0-17, 18-29, etc.). (String)",
    "Zip Code - 3 digits": "First three digits of the patient's zip code.␣
↪(String)",
    "Gender": "Gender of the patient (Male, Female, Other). (String)",
    "Race": "Race of the patient (e.g., White, Black, Asian). (String)",
```

```
    "Ethnicity": "Ethnicity of the patient (e.g., Hispanic, Non-Hispanic). (String)",
    "Length of Stay": "Total number of days the patient was hospitalized. (String)",
    "Type of Admission": "Reason for admission (e.g., Emergency, Elective, Newborn). (String)",
    "Patient Disposition": "Patient's discharge status (e.g., Home, Transferred, Expired). (String)",
    "Discharge Year": "Year in which the patient was discharged. (Integer)",
    "CCS Diagnosis Code": "Clinical Classification Software (CCS) diagnosis code. (Float)",
    "CCS Diagnosis Description": "Description of the CCS Diagnosis. (String)",
    "CCS Procedure Code": "CCS procedure code for the patient. (Float)",
    "CCS Procedure Description": "Description of the CCS Procedure performed. (String)",
    "APR DRG Code": "All Patient Refined Diagnosis Related Group (APR DRG) Code. (Integer)",
    "APR DRG Description": "Description of the APR DRG classification. (String)",
    "APR MDC Code": "All Patient Refined Major Diagnostic Category (APR MDC) Code. (Integer)",
    "APR MDC Description": "Description of the APR MDC classification. (String)",
    "APR Severity of Illness Code": "Code representing severity of illness (1-4 scale). (Integer)",
    "APR Severity of Illness Description": "Description of severity level (Minor, Moderate, etc.). (String)",
    "APR Risk of Mortality": "APR risk of mortality classification. (String)",
    "APR Medical Surgical Description": "Classification of the case as Medical or Surgical. (String)",
    "Source of Payment 1": "Primary payment source (e.g., Medicaid, Medicare, Private Insurance). (String)",
    "Source of Payment 2": "Secondary payment source, if applicable. (String)",
    "Source of Payment 3": "Tertiary payment source, if applicable. (String)",
    "Attending Provider License Number": "License number of the attending provider. (Float)",
    "Operating Provider License Number": "License number of the operating provider. (Float)",
    "Other Provider License Number": "License number of any additional providers. (Float)",
    "Birth Weight": "Birth weight of newborns in grams. (Integer)",
    "Abortion Edit Indicator": "Indicator for abortion-related data edits. (String)",
    "Emergency Department Indicator": "Indicates if the admission was through the Emergency Department. (String)",
    "Total Charges": "Total amount charged for hospital services. (Float)",
```

```
    "Total Costs": "Total cost incurred by the hospital for the patient stay.␣
↪(Float)"
}
```

[12]:

```
import pandas as pd

desc_df = pd.DataFrame(list(column_descriptions.items()), columns=["Column␣
↪Name", "Description"])
display(desc_df)
```

```
                              Column Name  \
0                                   index
1                     Health Service Area
2                         Hospital County
3            Operating Certificate Number
4                             Facility ID
5                           Facility Name
6                               Age Group
7                     Zip Code - 3 digits
8                                  Gender
9                                    Race
10                              Ethnicity
11                         Length of Stay
12                      Type of Admission
13                    Patient Disposition
14                         Discharge Year
15                     CCS Diagnosis Code
16              CCS Diagnosis Description
17                     CCS Procedure Code
18              CCS Procedure Description
19                           APR DRG Code
20                    APR DRG Description
21                           APR MDC Code
22                    APR MDC Description
23           APR Severity of Illness Code
24    APR Severity of Illness Description
25                 APR Risk of Mortality
26        APR Medical Surgical Description
27                    Source of Payment 1
28                    Source of Payment 2
29                    Source of Payment 3
30       Attending Provider License Number
31       Operating Provider License Number
32           Other Provider License Number
33                           Birth Weight
34                Abortion Edit Indicator
35          Emergency Department Indicator
```

```
36                           Total Charges
37                             Total Costs

                                              Description
0               Unique row index identifier. (Integer)
1          Geographic region of the hospital. (String)
2    County in which the hospital is located. (String)
3             Unique identifier for hospitals. (Float)
4    Unique identifier for the healthcare facility....
5                        Name of the hospital. (String)
6    Age group of the patient (e.g., 0-17, 18-29, e...
7    First three digits of the patient's zip code. ...
8    Gender of the patient (Male, Female, Other). (...
9    Race of the patient (e.g., White, Black, Asian...
10   Ethnicity of the patient (e.g., Hispanic, Non-...
11   Total number of days the patient was hospitali...
12   Reason for admission (e.g., Emergency, Electiv...
13   Patient's discharge status (e.g., Home, Transf...
14   Year in which the patient was discharged. (Int...
15   Clinical Classification Software (CCS) diagnos...
16          Description of the CCS Diagnosis. (String)
17         CCS procedure code for the patient. (Float)
18   Description of the CCS Procedure performed. (S...
19   All Patient Refined Diagnosis Related Group (A...
20   Description of the APR DRG classification. (St...
21   All Patient Refined Major Diagnostic Category ...
22   Description of the APR MDC classification. (St...
23   Code representing severity of illness (1-4 sca...
24   Description of severity level (Minor, Moderate...
25      APR risk of mortality classification. (String)
26   Classification of the case as Medical or Surgi...
27   Primary payment source (e.g., Medicaid, Medica...
28   Secondary payment source, if applicable. (String)
29    Tertiary payment source, if applicable. (String)
30   License number of the attending provider. (Float)
31   License number of the operating provider. (Float)
32   License number of any additional providers. (F...
33        Birth weight of newborns in grams. (Integer)
34   Indicator for abortion-related data edits. (St...
35   Indicates if the admission was through the Eme...
36   Total amount charged for hospital services. (F...
37   Total cost incurred by the hospital for the pa...
```

[13]:
```
display(df.describe())
```

```
              index  Operating Certificate Number   Facility ID  \
count  2.622133e+06                  2.617246e+06  2.617246e+06
```

mean	1.311066e+06	5.019051e+06	1.036953e+03
std	7.569447e+05	2.249577e+06	6.448892e+02
min	0.000000e+00	1.010000e+05	1.000000e+00
25%	6.555330e+05	2.951001e+06	5.410000e+02
50%	1.311066e+06	5.907002e+06	1.117000e+03
75%	1.966599e+06	7.002002e+06	1.450000e+03
max	2.622132e+06	7.004010e+06	9.059000e+03

	Discharge Year	CCS Diagnosis Code	CCS Procedure Code	APR DRG Code	\
count	2622133.0	2.619860e+06	2.619860e+06	2.622133e+06	
mean	2010.0	1.919431e+02	1.121324e+02	4.069629e+02	
std	0.0	1.565932e+02	8.761224e+01	2.404910e+02	
min	2010.0	1.000000e+00	0.000000e+00	1.000000e+00	
25%	2010.0	1.060000e+02	0.000000e+00	1.980000e+02	
50%	2010.0	1.570000e+02	1.240000e+02	3.610000e+02	
75%	2010.0	2.180000e+02	2.110000e+02	6.400000e+02	
max	2010.0	6.700000e+02	2.310000e+02	9.560000e+02	

	APR MDC Code	APR Severity of Illness Code	\
count	2.622133e+06	2.622133e+06	
mean	1.016206e+01	1.948373e+00	
std	5.923678e+00	8.986120e-01	
min	0.000000e+00	0.000000e+00	
25%	5.000000e+00	1.000000e+00	
50%	8.000000e+00	2.000000e+00	
75%	1.500000e+01	3.000000e+00	
max	2.500000e+01	4.000000e+00	

	Attending Provider License Number	Operating Provider License Number	\
count	2.617246e+06	1.951315e+06	
mean	1.266323e+06	1.569229e+06	
std	8.226220e+06	9.379695e+06	
min	1.000000e+00	1.000000e+00	
25%	1.666590e+05	1.644990e+05	
50%	2.053620e+05	2.012870e+05	
75%	2.335610e+05	2.305590e+05	
max	9.100000e+07	9.100000e+07	

	Other Provider License Number	Birth Weight	Total Charges	\
count	4.881420e+05	2.622133e+06	2.622133e+06	
mean	2.213083e+06	3.103701e+02	2.945920e+04	
std	1.225216e+07	9.769686e+02	5.640021e+04	
min	1.100000e+01	0.000000e+00	1.000000e-02	
25%	1.662850e+05	0.000000e+00	7.900000e+03	
50%	2.029570e+05	0.000000e+00	1.567373e+04	
75%	2.335750e+05	0.000000e+00	3.172352e+04	
max	9.100000e+07	9.900000e+03	1.206004e+07	

```
        Total Costs
count  2.622133e+06
mean   1.186523e+04
std    2.727029e+04
min    0.000000e+00
25%    3.290510e+03
50%    6.175350e+03
75%    1.222803e+04
max    1.470885e+07
```

[14]:
```python
display(df.describe(include=['O']))
```

```
       Health Service Area Hospital County         Facility Name    Age Group  \
count              2617246         2617246               2622133      2622133
unique                   8              57                   225            5
top          New York City       Manhattan  Mount Sinai Hospital  70 or Older
freq               1233121          445288                 58696       725253

       Zip Code - 3 digits   Gender     Race          Ethnicity  \
count              2618932  2622133  2622133            2622133
unique                  50        3        4                  3
top                    112        F    White  Not Span/Hispanic
freq                364399  1477671  1601378            2116780

       Length of Stay Type of Admission  ...  \
count         2622133           2622133  ...
unique            120                 6  ...
top                 2         Emergency  ...
freq           583446           1628999  ...

                                     APR DRG Description  \
count                                            2622133
unique                                               316
top     NEONATE BIRTHWT >2499G, NORMAL NEWBORN OR NEON...
freq                                              210893

                                    APR MDC Description  \
count                                           2622133
unique                                               26
top     Diseases and Disorders of the Circulatory System
freq                                             396907

       APR Severity of Illness Description APR Risk of Mortality  \
count                              2621892               2621892
unique                                   4                     4
top                                  Minor                 Minor
freq                                974425               1644151
```

```
       APR Medical Surgical Description Source of Payment 1  \
count                           2622133            2622133
unique                                3                 10
top                             Medical           Medicare
freq                            2002344             866859

       Source of Payment 2 Source of Payment 3 Abortion Edit Indicator  \
count              1811752              581292                 2622133
unique                  10                  10                       2
top               Medicaid            Self-Pay                       N
freq                580109              406327                 2617246

       Emergency Department Indicator
count                         2622133
unique                              2
top                                 Y
freq                          1476286

[4 rows x 23 columns]
```

[15]:
```python
import pandas as pd

file_path = "/Users/desireereid/Downloads/hospital inpatient discharges.csv"

df = pd.read_csv(file_path, low_memory=False)
```

[16]:
```python
print(df.isnull().sum())
missing_values = df.isnull().sum()
missing_values
```

```
index                                 0
Health Service Area                4887
Hospital County                    4887
Operating Certificate Number       4887
Facility ID                        4887
Facility Name                         0
Age Group                             0
Zip Code - 3 digits                3201
Gender                                0
Race                                  0
Ethnicity                             0
Length of Stay                        0
Type of Admission                     0
Patient Disposition                 103
Discharge Year                        0
CCS Diagnosis Code                 2273
```

```
CCS Diagnosis Description                   2273
CCS Procedure Code                          2273
CCS Procedure Description                   2273
APR DRG Code                                   0
APR DRG Description                            0
APR MDC Code                                   0
APR MDC Description                            0
APR Severity of Illness Code                   0
APR Severity of Illness Description          241
APR Risk of Mortality                        241
APR Medical Surgical Description               0
Source of Payment 1                            0
Source of Payment 2                       810381
Source of Payment 3                      2040841
Attending Provider License Number           4887
Operating Provider License Number         670818
Other Provider License Number            2133991
Birth Weight                                   0
Abortion Edit Indicator                        0
Emergency Department Indicator                 0
Total Charges                                  0
Total Costs                                    0
dtype: int64
```

[16]:
```
index                                          0
Health Service Area                         4887
Hospital County                             4887
Operating Certificate Number                4887
Facility ID                                 4887
Facility Name                                  0
Age Group                                      0
Zip Code - 3 digits                         3201
Gender                                         0
Race                                           0
Ethnicity                                      0
Length of Stay                                 0
Type of Admission                              0
Patient Disposition                          103
Discharge Year                                 0
CCS Diagnosis Code                          2273
CCS Diagnosis Description                   2273
CCS Procedure Code                          2273
CCS Procedure Description                   2273
APR DRG Code                                   0
APR DRG Description                            0
APR MDC Code                                   0
APR MDC Description                            0
```

```
APR Severity of Illness Code                    0
APR Severity of Illness Description           241
APR Risk of Mortality                         241
APR Medical Surgical Description                0
Source of Payment 1                             0
Source of Payment 2                        810381
Source of Payment 3                       2040841
Attending Provider License Number            4887
Operating Provider License Number          670818
Other Provider License Number             2133991
Birth Weight                                    0
Abortion Edit Indicator                         0
Emergency Department Indicator                  0
Total Charges                                   0
Total Costs                                     0
dtype: int64
```

[17]:

```python
print(missing_values[missing_values > 0])
```

```
Health Service Area                          4887
Hospital County                              4887
Operating Certificate Number                 4887
Facility ID                                  4887
Zip Code - 3 digits                          3201
Patient Disposition                           103
CCS Diagnosis Code                           2273
CCS Diagnosis Description                    2273
CCS Procedure Code                           2273
CCS Procedure Description                    2273
APR Severity of Illness Description           241
APR Risk of Mortality                         241
Source of Payment 2                        810381
Source of Payment 3                       2040841
Attending Provider License Number            4887
Operating Provider License Number          670818
Other Provider License Number             2133991
dtype: int64
```

[18]:

```python
import pandas as pd

missing_values_df = pd.DataFrame({
    "Column": [
        "Health Service Area", "Hospital County", "Operating Certificate␣
↪Number", "Facility ID",
        "Zip Code - 3 digits", "Patient Disposition", "CCS Diagnosis Code",␣
↪"CCS Diagnosis Description",
```

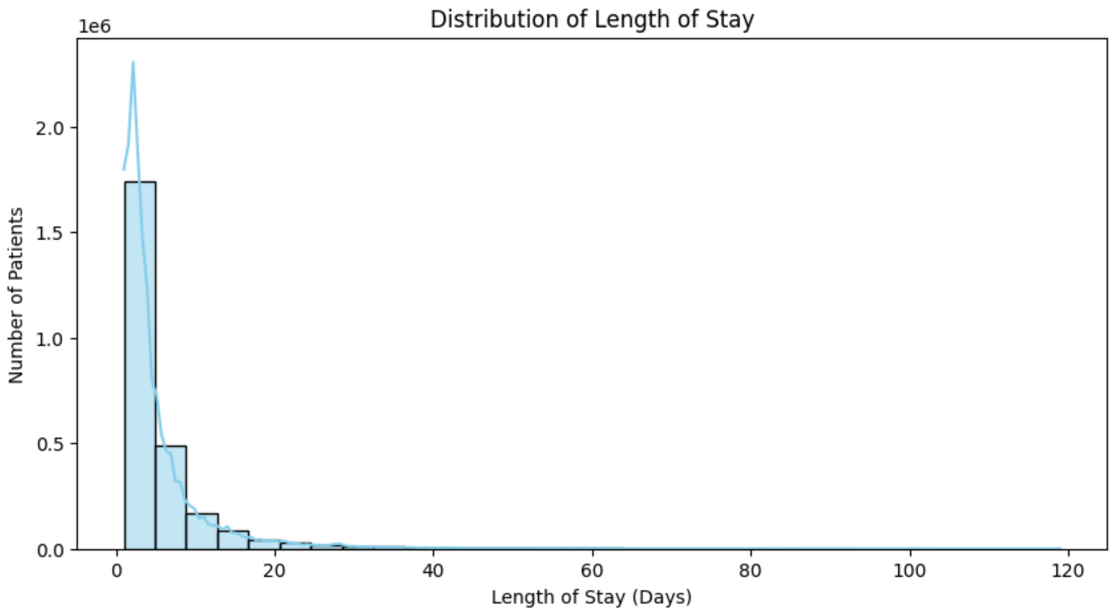
```
        "CCS Procedure Code", "CCS Procedure Description", "APR Severity of␣
↪Illness Description",
        "APR Risk of Mortality", "Source of Payment 2", "Source of Payment 3",
        "Attending Provider License Number", "Operating Provider License␣
↪Number", "Other Provider License Number"
    ],
    "Missing Count": [
        4887, 4887, 4887, 4887, 3201, 103, 2273, 2273, 2273, 2273, 241, 241,␣
↪810381, 2040841, 4887, 670818, 2133991
    ],
    "% Missing": [
        "0.19%", "0.19%", "0.19%", "0.19%", "0.12%", "0.004%", "0.08%", "0.
↪08%", "0.08%", "0.08%",
        "0.009%", "0.009%", "30.9%", "77.8%", "0.19%", "25.6%", "81.4%"
    ],
    "Action": [
        "Fill", "Fill", "Fill", "Fill", "Fill with 'Unknown'", "Drop Rows",␣
↪"Fill with Median", "Fill with Mode",
        "Fill with Median", "Fill with Mode", "Fill with Mode", "Fill with␣
↪Mode", "Drop Column", "Drop Column",
        "Fill with Mode or Drop", "Drop Column", "Drop Column"
    ],
    "Reasoning": [
        "Small % missing, can fill with mode", "Small % missing, can fill with␣
↪mode",
        "Important identifier, can fill with most common hospital's ID", "Keep␣
↪if needed for hospital tracking",
        "Useful for location-based analysis", "Very few missing, safe to␣
↪remove", "Helps in classification models",
        "Can be inferred from CCS Diagnosis Code", "Keeps model integrity",␣
↪"Can be inferred from CCS Procedure Code",
        "Important for analyzing hospital outcomes", "Essential for healthcare␣
↪disparity insights",
        "Too much missing data, may not be useful", "Extremely sparse, not␣
↪reliable",
        "Only keep if analyzing provider behavior", "Large gaps, not useful for␣
↪most analysis",
        "Too much missing data, unreliable"
    ]
})

from IPython.display import display

display(missing_values_df)
```

```
                              Column  Missing Count % Missing  \
```

```
0                      Health Service Area          4887     0.19%
1                          Hospital County          4887     0.19%
2           Operating Certificate Number          4887     0.19%
3                              Facility ID          4887     0.19%
4                      Zip Code - 3 digits          3201     0.12%
5                      Patient Disposition           103    0.004%
6                       CCS Diagnosis Code          2273     0.08%
7                CCS Diagnosis Description          2273     0.08%
8                       CCS Procedure Code          2273     0.08%
9                CCS Procedure Description          2273     0.08%
10  APR Severity of Illness Description           241    0.009%
11                  APR Risk of Mortality           241    0.009%
12                    Source of Payment 2        810381     30.9%
13                    Source of Payment 3       2040841     77.8%
14    Attending Provider License Number          4887     0.19%
15    Operating Provider License Number        670818     25.6%
16        Other Provider License Number       2133991     81.4%

                    Action                                      Reasoning
0                     Fill               Small % missing, can fill with mode
1                     Fill               Small % missing, can fill with mode
2                     Fill  Important identifier, can fill with most commo...
3                     Fill               Keep if needed for hospital tracking
4       Fill with 'Unknown'             Useful for location-based analysis
5                Drop Rows                 Very few missing, safe to remove
6         Fill with Median                   Helps in classification models
7           Fill with Mode          Can be inferred from CCS Diagnosis Code
8         Fill with Median                            Keeps model integrity
9           Fill with Mode          Can be inferred from CCS Procedure Code
10          Fill with Mode            Important for analyzing hospital outcomes
11          Fill with Mode          Essential for healthcare disparity insights
12             Drop Column          Too much missing data, may not be useful
13             Drop Column                      Extremely sparse, not reliable
14  Fill with Mode or Drop           Only keep if analyzing provider behavior
15             Drop Column          Large gaps, not useful for most analysis
16             Drop Column                 Too much missing data, unreliable
```

[19]:

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

plt.figure(figsize=(10, 5))
sns.histplot(cleaned_df["Length of Stay"], bins=30, kde=True, color="skyblue")
plt.xlabel("Length of Stay (Days)")
plt.ylabel("Number of Patients")
plt.title("Distribution of Length of Stay")
plt.show()
```



Distribution of Length of Stay This histogram visualizes the distribution of hospital **Length of Stay (LOS)** in days.

Key Observations:
- The distribution is **highly right-skewed**, with most patients staying **1–3 days**.
- A sharp decline is observed after the first few days, indicating that **longer hospital stays are less common**.
- A small number of patients experience **extended hospitalizations (beyond 20 days)**, but these cases are rare.

This distribution reveals that the majority of patients are discharged within the first few days, with a steep drop-off after day 3. This suggests that short-term hospitalizations are far more common, possibly indicating a higher volume of routine or less severe cases. This distribution helps to identify thresholds for defining outliers or unusually long hospital stays, which may merit deeper investigation in disparity or cost analyses.

[20]:
```
import pandas as pd

df["Length of Stay"] = pd.to_numeric(df["Length of Stay"], errors="coerce")

los_counts = df["Length of Stay"].value_counts().sort_index()

los_counts.index = los_counts.index.astype(int)

selected_days = list(range(1, 11)) + [20, 40, 60, 80, 100]
los_counts_filtered = los_counts[los_counts.index.isin(selected_days)]
```

```
los_counts_df = pd.DataFrame({
    "Length of Stay (Days)": los_counts_filtered.index,
    "Number of Patients": los_counts_filtered.values
})

print(los_counts_df)
```

```
    Length of Stay (Days)  Number of Patients
0                       1              428644
1                       2              583446
2                       3              449094
3                       4              281838
4                       5              178402
5                       6              130230
6                       7              104813
7                       8               76128
8                       9               56096
9                      10               44466
10                     20                9108
11                     40                1210
12                     60                 416
13                     80                 145
14                    100                  88
```

Distribution of Length of Stay (LOS) for Selected Days This query retrieves the number of patients for **common short stays (1-10 days)** and selected longer stays **(20, 40, 60, 80, 100 days)** to better understand hospital discharge patterns.

Output Summary:
- The majority of hospital stays are **short**: - **1-day stays are the most frequent (428,644 patients)**. - The number of patients gradually decreases as LOS increases.
- Longer stays (**20+ days**) are much less common: - Only **9,108 patients stayed for 20 days**. - **Fewer than 500 patients** stayed **60+ days**. - **Only 88 patients had a 100-day hospital stay**.

This confirms a **right-skewed distribution**, where most patients are discharged within a few days, while extended stays are rare.

[21]:
```
df["Length of Stay"] = pd.to_numeric(df["Length of Stay"], errors="coerce")

los_100_119 = df[(df["Length of Stay"] >= 100) & (df["Length of Stay"] <=␣
↪119)]["Length of Stay"].value_counts().sort_index()

print(los_100_119)
```

```
Length of Stay
100.0    88
101.0    67
```

```
102.0    71
103.0    86
104.0    96
105.0    85
106.0    71
107.0    55
108.0    66
109.0    75
110.0    63
111.0    60
112.0    54
113.0    71
114.0    67
115.0    41
116.0    38
117.0    50
118.0    57
119.0    63
Name: count, dtype: int64
```

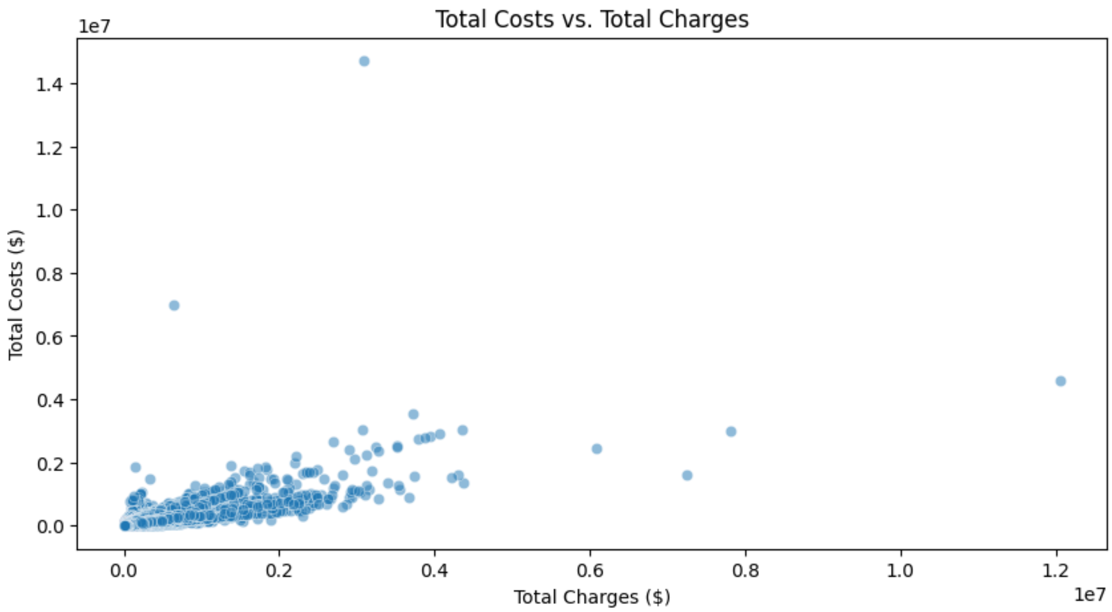
[22]:

```python
df["Length of Stay"] = pd.to_numeric(df["Length of Stay"], errors="coerce")

max_los = df["Length of Stay"].max()

print(f"The longest recorded hospital stay is {max_los} days.")
```

```
The longest recorded hospital stay is 119.0 days.
```

[23]:

```python
plt.figure(figsize=(10, 5))
sns.scatterplot(x=cleaned_df["Total Charges"], y=cleaned_df["Total Costs"],␣
↪alpha=0.5)
plt.xlabel("Total Charges ($)")
plt.ylabel("Total Costs ($)")
plt.title("Total Costs vs. Total Charges")
plt.show()
```



Total Costs vs. Total Charges This scatter plot visualizes the relationship between **Total Charges** $() * *and * *TotalCosts()$ for hospital stays.

Key Observations:
- Most data points are clustered near the lower left, indicating that the majority of hospital stays have relatively **low costs and charges**.
- A few extreme outliers show **very high charges and costs**, suggesting **unusually expensive hospitalizations**.
- The trend suggests a **positive correlation**, meaning that as total charges increase, total costs also tend to rise.

This plot helps identify cost outliers and assess **financial trends in hospital billing**.

The scatter plot shows a consistent markup in total charges compared to the actual costs incurred by hospitals. Many points lie well above the identity line, suggesting that hospitals frequently charge significantly more than the cost of care. This supports the need to explore financial disparities and whether patients from different groups are charged differently for similar services.

[24]:
```
df["Charge-Cost Difference"] = df["Total Charges"] - df["Total Costs"]

print(df["Charge-Cost Difference"].describe())
```

```
count    2.622133e+06
mean     1.759397e+04
std      3.727772e+04
min     -1.162138e+07
25%      3.667580e+03
50%      8.846210e+03
```

```
75%        1.939393e+04
max        7.462672e+06
Name: Charge-Cost Difference, dtype: float64
```

Difference Between Total Charges and Total Costs This analysis examines the difference between Total Charges (hospital billed amount) and Total Costs (actual expenses incurred by the hospital).

0.0.1 Summary Statistics

- Total Records Analyzed: 2,622,133
- Average Charge-Cost Difference: $17,594
  - On average, hospitals charge $17,594 more than actual costs per patient.
- Standard Deviation: $37,278
  - High variation in charge-cost differences.
- Minimum Value: -$11,621,380
  - Some records show higher costs than charges, possibly due to reporting errors or adjustments.
- 25th Percentile (Q1): $3,668
  - 25% of hospital stays had a difference below this.
- Median (Q2): $8,846
  - Half of stays had a charge-cost difference below this.
- 75th Percentile (Q3): $19,394
  - 75% of stays had a difference below this.
- Maximum Value: $7,462,672
  - Extreme outliers where hospitals charged millions above actual costs.

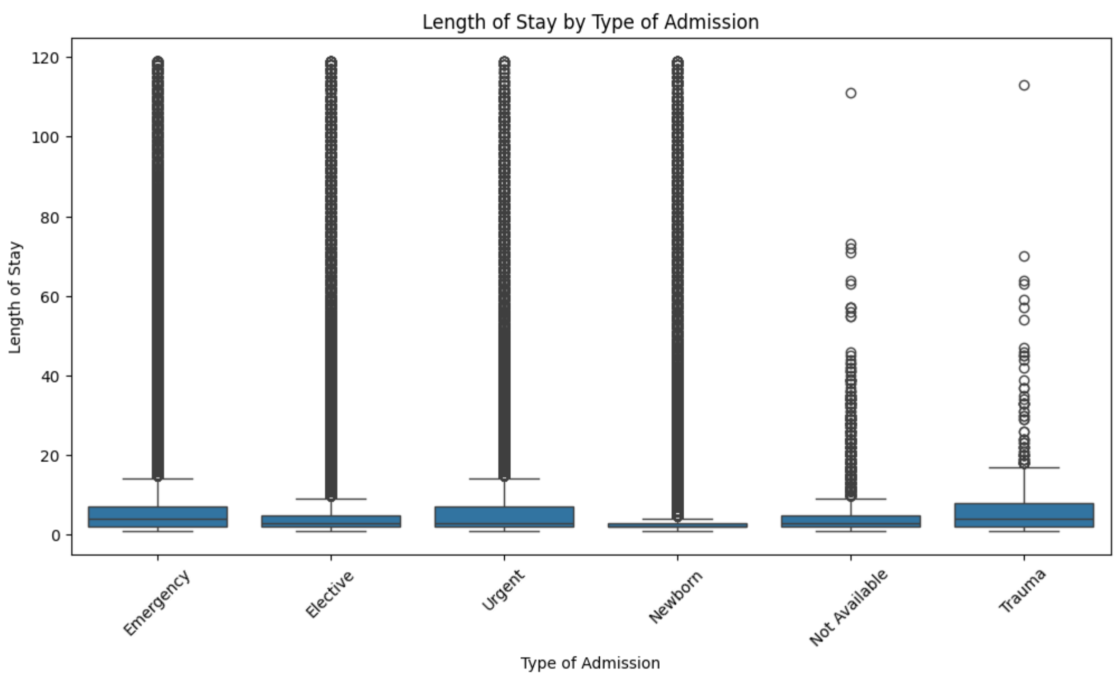
0.0.2 Key Insights

- Most hospital charges are significantly higher than actual costs, with an average markup of $17,594 per patient.
- Large variation in markup suggests differences in hospital policies, insurance negotiations, or patient-specific factors.
- Negative values in the data may indicate billing errors or special cases where costs exceeded charges.

- Some extreme cases show charge-cost differences exceeding $7 million, highlighting significant outliers in billing practices.

Further investigation into negative values and outliers may help identify systemic issues in hospital pricing strategies.

[25]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x="Type of Admission", y="Length of Stay", data=cleaned_df)
plt.xticks(rotation=45)
plt.title("Length of Stay by Type of Admission")
plt.show()
```

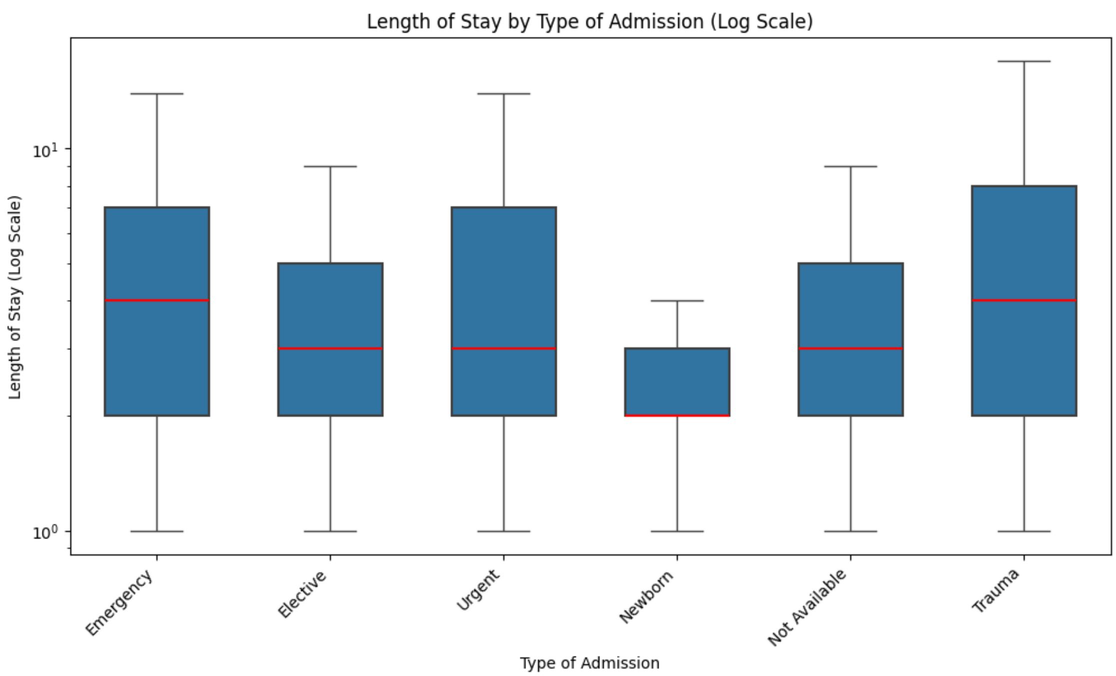


Length of Stay by Type of Admission (Original Box Plot) This box plot visualizes the distribution of **Length of Stay (LOS)** across different types of hospital admissions, including all individual outliers.

Key Observations:
- Most admission types have a **median LOS of a few days**, with interquartile ranges staying relatively low.
- **Newborn admissions tend to have the shortest stays**, while **trauma and urgent cases** show slightly longer stays.
- **Numerous extreme outliers** (individual points above each box) indicate that some hospital stays last **well beyond the typical range**, reaching 120+ days.
- The **Not Available category** has a broader spread, suggesting it might include a mix of different admission types.

This version is useful for identifying **unusually long hospital stays**, as it highlights **outliers explicitly**.

[26]:
```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(12, 6))

sns.boxplot(
    x="Type of Admission",
    y="Length of Stay",
    data=df,
    showfliers=False,
    width=0.6,
    boxprops={'linewidth': 1.5},
    medianprops={'color': 'red', 'linewidth': 1.5}
)

plt.xticks(rotation=45, ha="right")

plt.yscale("log")
plt.ylabel("Length of Stay (Log Scale)")

plt.title("Length of Stay by Type of Admission (Log Scale)")

plt.show()
```



Length of Stay by Type of Admission (Log-Scaled Box Plot) This improved box plot visualizes **Length of Stay (LOS) across admission types** while applying a **logarithmic scale** to better represent the distribution.

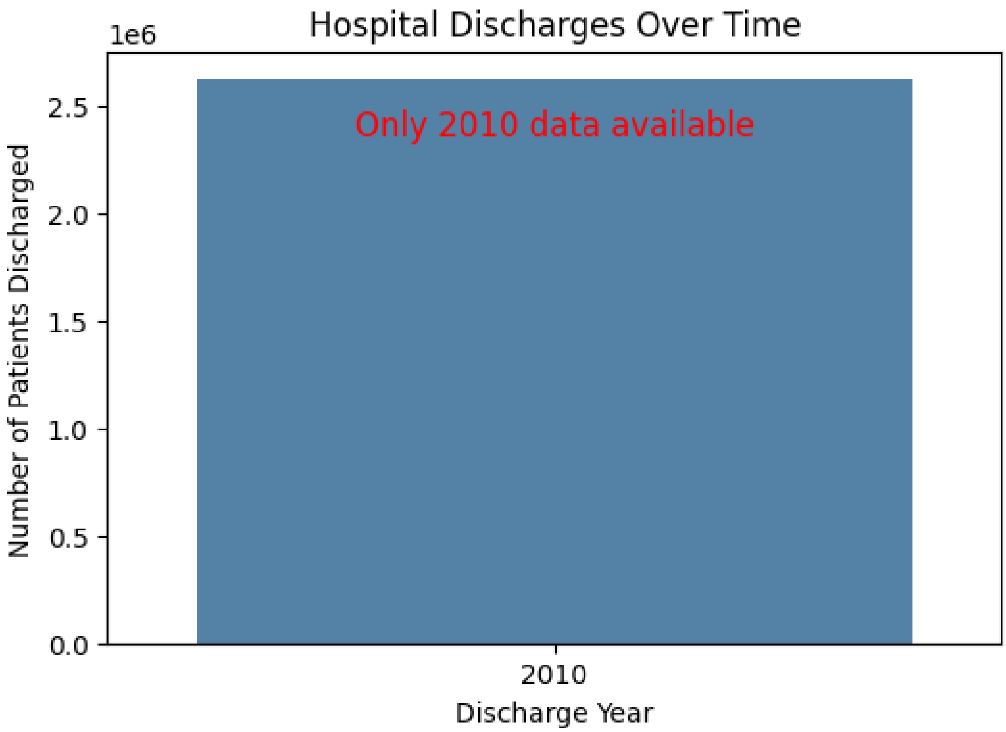
Why This Version is Different:
- In the original plot, extreme **outliers crowded the visualization**, making it difficult to compare typical LOS distributions.
- The **log scale prevents long stays from overpowering shorter ones**, ensuring a **clearer comparison between different admission types**.
- **Outliers have been hidden** to reduce clutter, focusing on the overall trends rather than extreme values.

Key Observations:
- The **central trends (medians)** of LOS across admission types are now **easier to compare**.
- The **Newborn category has the shortest stays**, while **trauma and emergency admissions** have slightly longer distributions.
- The spread of LOS within each category is **more clearly visible**, providing better insights into **variability in hospital stays**.

This version is **better suited for understanding general trends in LOS** without being skewed by extreme outliers.

Emergency admissions tend to have a wider spread and higher median stay duration, suggesting more unpredictable or complex cases. In contrast, elective admissions show shorter and more consistent stays. This insight can help inform how admission type impacts hospital resource allocation and can be used in predictive models for expected length of stay.

[27]:
```
import matplotlib.pyplot as plt
import seaborn as sns

year_counts = df["Discharge Year"].value_counts()

plt.figure(figsize=(6, 4))
sns.barplot(x=year_counts.index, y=year_counts.values, color="steelblue")

plt.xlabel("Discharge Year")
plt.ylabel("Number of Patients Discharged")
plt.title("Hospital Discharges Over Time")

plt.text(0, year_counts.max() * 0.9, "Only 2010 data available", ha="center",␣
 ↪fontsize=12, color="red")

plt.show()
```



[28]:
```
print("Unique Discharge Years:", df["Discharge Year"].unique())

print("\nDischarge Year Counts:")
print(df["Discharge Year"].value_counts().sort_index())
```

```
Unique Discharge Years: [2010]

Discharge Year Counts:
Discharge Year
2010    2622133
Name: count, dtype: int64
```

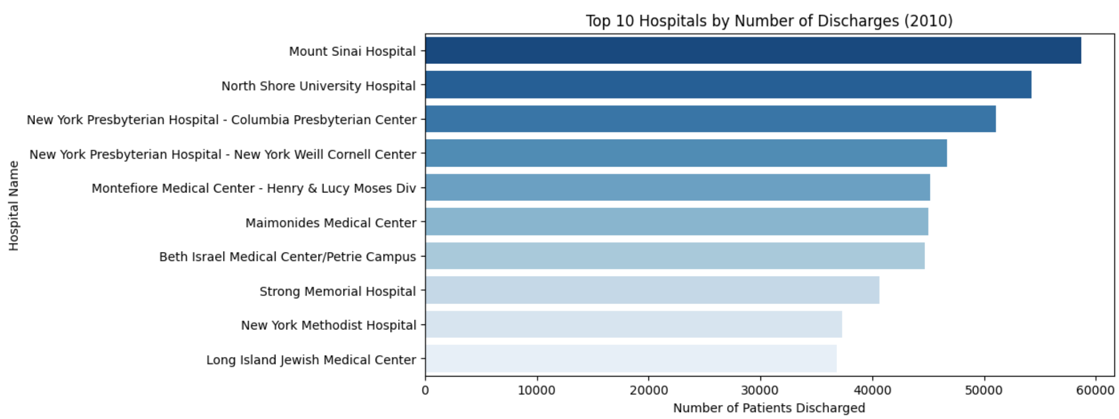
Hospital Discharges Over Time This bar chart was initially created to analyze hospital discharge trends over multiple years. However, the visualization revealed that the dataset only contains **one year (2010)**, making a time-series analysis **not possible**.

To highlight this limitation, an annotation was added to indicate that all recorded discharges occurred in **2010**. Since trend analysis is not applicable, subsequent visualizations focus on **discharges by hospital, admission type, and other meaningful categories**.

[29]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
hospital_counts = df["Facility Name"].value_counts().head(10)  # Top 10␣
↪hospitals

plt.figure(figsize=(10, 5))
sns.barplot(x=hospital_counts.values, y=hospital_counts.index,␣
↪palette="Blues_r")

plt.xlabel("Number of Patients Discharged")
plt.ylabel("Hospital Name")
plt.title("Top 10 Hospitals by Number of Discharges (2010)")
plt.show()
```

```
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/1932864992.py:9:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=hospital_counts.values, y=hospital_counts.index,
palette="Blues_r")
```



Top 10 Hospitals by Number of Discharges (2010) This horizontal bar chart displays the **top 10 hospitals with the highest number of discharges** in 2010.

Key Insights:
- **Mount Sinai Hospital had the highest number of discharges**, followed by **North Shore University Hospital** and **New York Presbyterian Hospital - Columbia Presbyterian Center**.
- The **distribution of discharges varies significantly** among hospitals, with the top-ranked hospitals discharging significantly more patients than lower-ranked ones.
- Hospitals affiliated with **New York Presbyterian and Montefiore Medical Center** appear

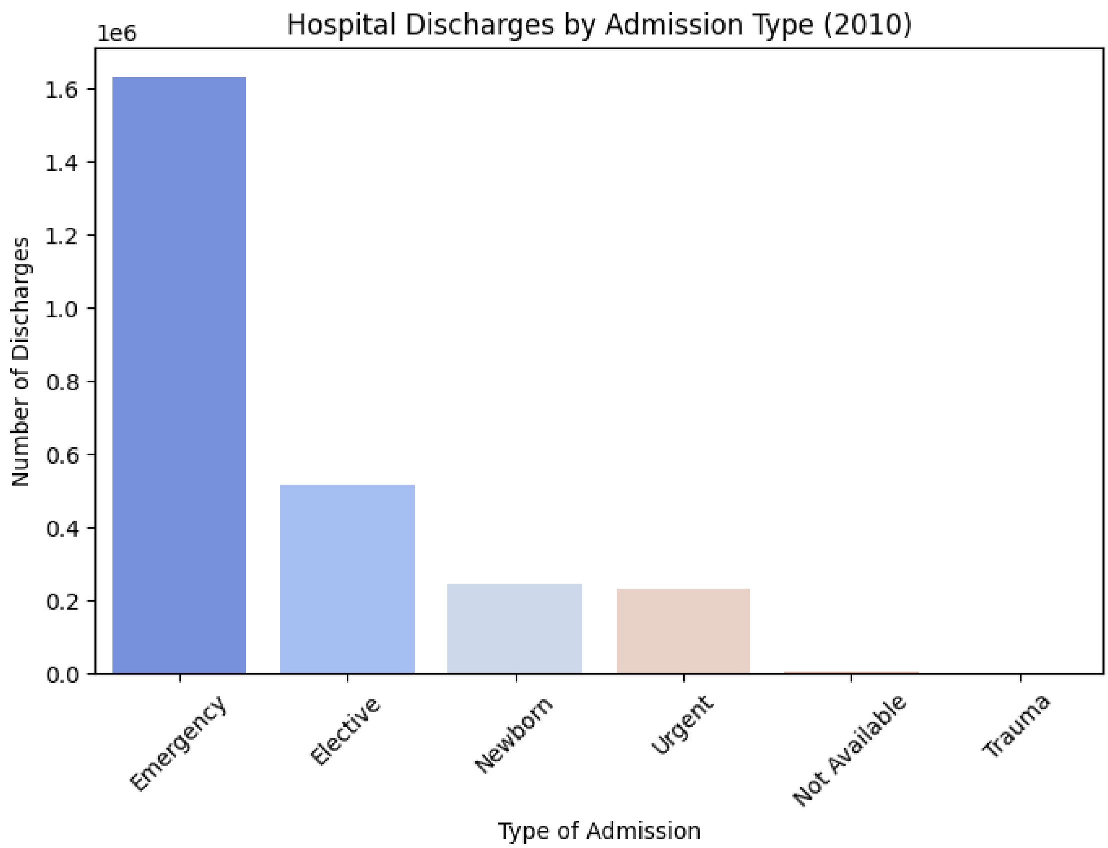
multiple times in the top 10, suggesting they handle a **high volume of inpatient care**.

What High Discharge Numbers Indicate:
- **Large hospital capacity** – High discharge numbers often suggest that a hospital has **more beds, staff, and infrastructure** to handle a large patient volume.
- **High patient turnover** – Hospitals with many discharges may operate **efficiently**, ensuring timely admissions and discharges.
- **Specialized or high-demand services** – Some hospitals may have **large specialized departments** (e.g., cardiology, trauma, or surgery) that attract a **higher volume of patients**.
- **Urban vs. Rural Differences** – Hospitals in **densely populated areas** tend to have higher discharges due to **greater patient demand**.

This visualization provides insight into **which hospitals managed the most patient discharges** in 2010, highlighting key facilities with high patient throughput and possibly specialized or high-demand healthcare services. Additionally, shows that Medicare and Medicaid are the most commonly used payment sources, especially among older or lower-income patients. This aligns with broader healthcare funding trends and emphasizes the importance of public insurance programs. Understanding payment source distribution is key when analyzing cost disparities and hospital revenue streams.

[30]:
```python
admission_counts = df["Type of Admission"].value_counts()

plt.figure(figsize=(8, 5))
sns.barplot(x=admission_counts.index, y=admission_counts.values,␣
 ↪palette="coolwarm")

plt.xlabel("Type of Admission")
plt.ylabel("Number of Discharges")
plt.title("Hospital Discharges by Admission Type (2010)")
plt.xticks(rotation=45)
plt.show()
```

```
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/1506840590.py:6:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=admission_counts.index, y=admission_counts.values,
palette="coolwarm")
```



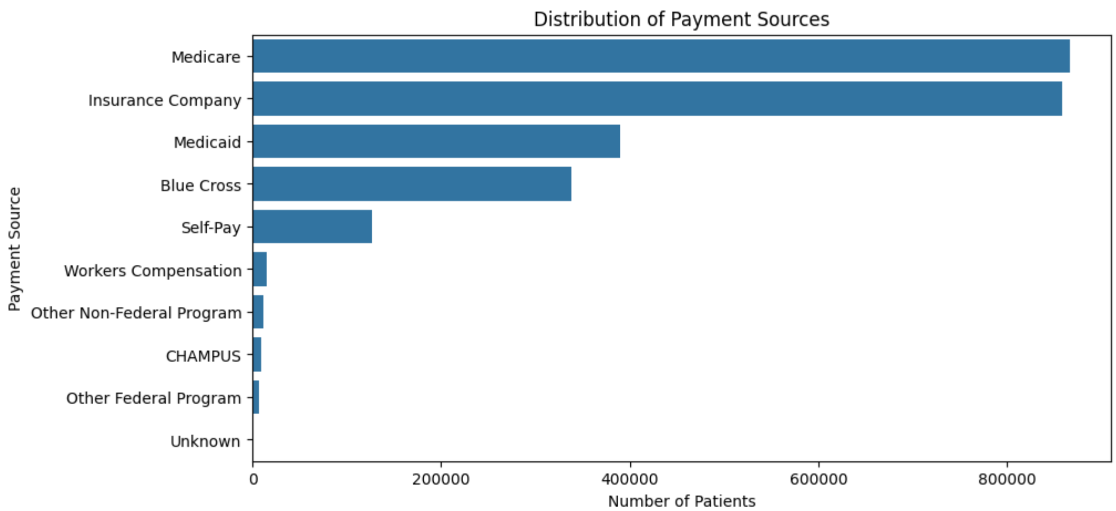
Hospital Discharges by Admission Type (2010) This bar chart displays the distribution of hospital discharges categorized by **type of admission** in 2010.

Key Insights:
- **Emergency admissions account for the vast majority of discharges**, significantly outnumbering all other admission types.
- **Elective admissions** follow as the second most common category, but with considerably fewer discharges compared to emergency cases.
- **Newborn and urgent admissions** have similar discharge volumes, both much lower than emergency or elective cases.
- **Trauma and "Not Available" admissions have very few recorded discharges**, indicating that either these categories are rare or there are **data inconsistencies**.

What This Means:
- The dominance of **emergency discharges** suggests that hospitals handle a **large volume of unplanned, urgent cases**, reflecting the critical role of emergency departments.
- The lower number of **elective discharges** could indicate that planned procedures and hospitalizations occur at a much lower rate compared to emergency visits.
- **Potential Data Issues:** The "Not Available" category may represent **missing or misclassified data**, and trauma cases may be underreported.

This visualization provides a **clear understanding of hospital admission patterns**, emphasizing the overwhelming volume of emergency cases in 2010.

[31]:
```
plt.figure(figsize=(10, 5))
sns.countplot(y="Source of Payment 1", data=cleaned_df,␣
↪order=cleaned_df["Source of Payment 1"].value_counts().index)
plt.xlabel("Number of Patients")
plt.ylabel("Payment Source")
plt.title("Distribution of Payment Sources")
plt.show()
```



Distribution of Payment Sources This bar chart displays the distribution of **primary payment sources** for hospital discharges in 2010.

Key Insights:
- **Medicare and private insurance companies** are the **top two payment sources**, accounting for the largest share of hospital discharges.
- **Medicaid is the third most common payer**, indicating a significant portion of patients rely on government-funded healthcare programs.
- **Blue Cross, Self-Pay, and Workers' Compensation** contribute smaller shares, suggesting a mix of employer-based, personal, and state-mandated insurance coverage.
- **Less common payment sources** (CHAMPUS, Other Federal Programs, and Unknown) represent a small fraction of discharges, possibly indicating specialized or unclassified cases.

What This Means:
- The dominance of **Medicare and Medicaid** suggests that a **large percentage of hospitalized patients are elderly or low-income individuals** covered under federal programs.
- The **significant presence of private insurance (Insurance Companies & Blue Cross)** highlights the role of employer-sponsored and commercial healthcare coverage.
- The **relatively small proportion of Self-Pay patients** may indicate that **most patients**

rely on insurance coverage rather than paying out-of-pocket.

This visualization helps understand the **financial structure of hospital discharges**, highlighting the reliance on **government-funded and private insurance programs** for healthcare payments.

[32]:
```
import pandas as pd

df = pd.read_csv("/Users/desireereid/Downloads/cleaned_hospital_data.csv",␣
↪low_memory=False)

print("Unique values in APR Risk of Mortality:", df["APR Risk of Mortality"].
↪unique())
```

```
Unique values in APR Risk of Mortality: ['Minor' 'Extreme' 'Major' 'Moderate'
nan]
```

[33]:
```
mortality_mapping = {"Minor": 1, "Moderate": 2, "Major": 3, "Extreme": 4}

df["APR Risk of Mortality"] = df["APR Risk of Mortality"].map(mortality_mapping)

df["APR Risk of Mortality"] = pd.to_numeric(df["APR Risk of Mortality"],␣
↪errors="coerce")

print(df["APR Risk of Mortality"].value_counts(dropna=False))
```

```
APR Risk of Mortality
1.0    1644151
2.0     554542
3.0     300829
4.0     122370
NaN        241
Name: count, dtype: int64
```

[34]:
```
df = df.dropna(subset=["APR Risk of Mortality"])
```

[35]:
```
import pandas as pd

cleaned_df = pd.read_csv("/Users/desireereid/Downloads/cleaned_hospital_data.
↪csv")

columns_to_drop = [
    "index",
    "Facility Name",
    "Zip Code - 3 digits",
    "Attending Provider License Number",
    "Operating Provider License Number",
    "Other Provider License Number",
    "Birth Weight",
```

```
    "Facility ID",
    "Operating Certificate Number",
    "Abortion Edit Indicator",
    "Emergency Department Indicator",
    "Source of Payment 2",
    "Source of Payment 3",
    "CCS Diagnosis Code",
    "CCS Diagnosis Description",
    "CCS Procedure Code",
    "CCS Procedure Description",
    "APR MDC Code"
]

cleaned_df_reduced = cleaned_df.drop(columns=columns_to_drop)

cleaned_df_reduced.to_csv("/Users/desireereid/Downloads/
↪cleaned_hospital_data_reduced.csv", index=False)

print("  Reduced dataset saved as 'cleaned_hospital_data_reduced.csv'")
```

```
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/949440234.py:4:
DtypeWarning: Columns (1,2,7,11,29) have mixed types. Specify dtype option on
import or set low_memory=False.
  cleaned_df =
pd.read_csv("/Users/desireereid/Downloads/cleaned_hospital_data.csv")
```

```
 Reduced dataset saved as 'cleaned_hospital_data_reduced.csv'
```

[36]:

```
cleaned_df_reduced["Charge_to_Cost_Ratio"] = cleaned_df_reduced["Total␣
↪Charges"] / cleaned_df_reduced["Total Costs"]
```

[37]:

```
df["Length of Stay"] = pd.to_numeric(df["Length of Stay"], errors="coerce")

if "LOS_Categoryb" not in df.columns:
    df["LOS_Category"] = pd.cut(
        df["Length of Stay"],
        bins=[0, 4, 10, np.inf],  # Short: 1-4 days, Medium: 5-10 days, Long:␣
↪11+ days
        labels=["Short", "Medium", "Long"]
    )

print(df["LOS_Category"].value_counts(dropna=False))
```

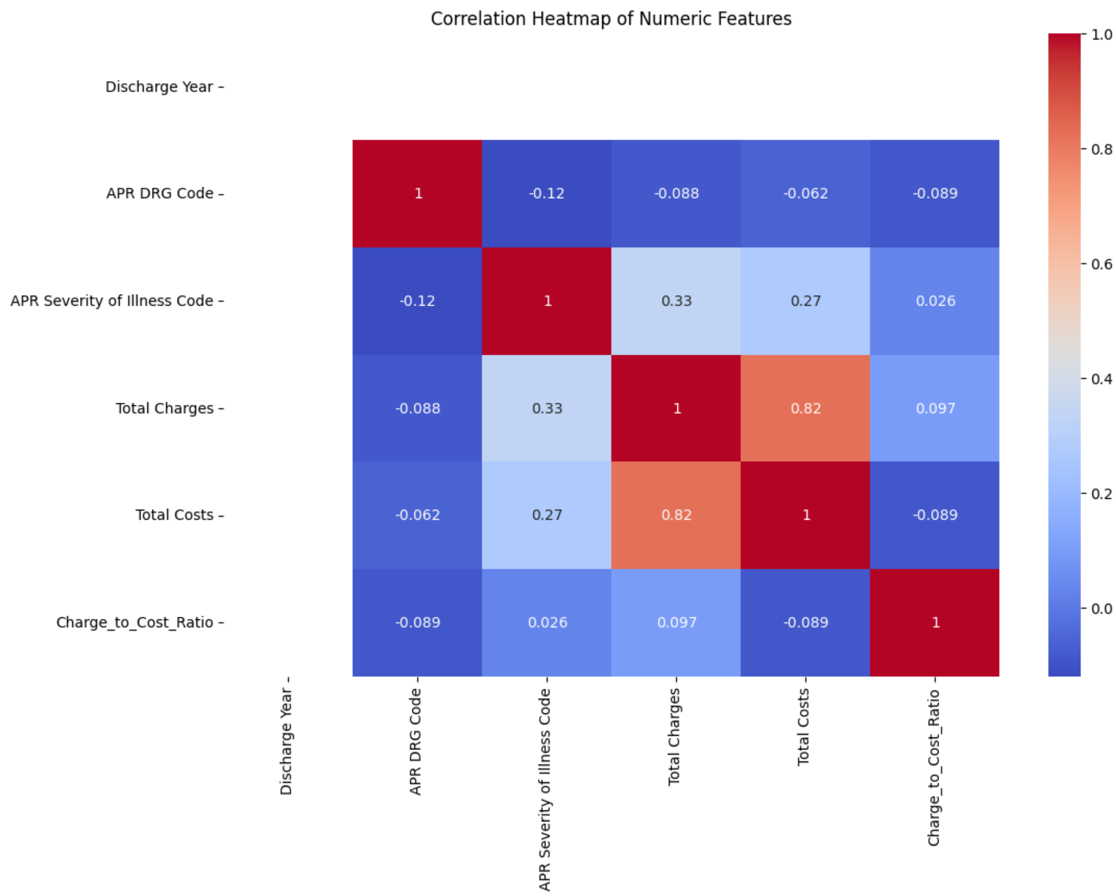
```
LOS_Category
Short     1742875
Medium     590086
Long       286724
NaN          2207
```

```
Name: count, dtype: int64
```

[38]:
```
from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
le.fit(df["LOS_Category"].dropna().astype(str))

print("Class label mapping:")
for i, label in enumerate(le.classes_):
    print(f"{i} → {label}")
```

```
Class label mapping:
0 → Long
1 → Medium
2 → Short
```

[39]:
```
import seaborn as sns
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 8))
corr_matrix = cleaned_df_reduced.corr(numeric_only=True)
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')
plt.title("Correlation Heatmap of Numeric Features")
plt.show()
```



To better understand how key numerical variables in the dataset interact, I created a correlation heatmap. My goal was to identify relationships between clinical and financial indicators to support my analysis of hospital resource utilization and cost patterns.

One of the strongest insights came from the high positive correlation (0.82) between Total Charges and Total Costs. This confirms a logical pattern — as the hospital spends more on patient care, it charges more. Although this seems intuitive, validating it in the data is important and reinforces the integrity of the dataset.

I also observed a moderate correlation between APR Severity of Illness Code and both Total Charges (0.33) and Total Costs (0.27). This tells me that sicker patients tend to generate higher hospital expenses and charges, which aligns with the real-world assumption that illness severity impacts utilization. This is directly relevant to my project's goal of analyzing disparities and drivers of hospital length of stay, particularly where severity may play a mediating role.

Interestingly, the Charge-to-Cost Ratio had weak correlations with all other variables. This suggests that the way hospitals price services in comparison to their actual costs isn't closely tied to the patient's clinical profile or care intensity. This could point to variations in administrative or payer policies and may be worth deeper investigation, especially when evaluating financial fairness or pricing transparency.

Another observation was the lack of correlation between Discharge Year and financial or clinical metrics, which implies stability over time, at least within this dataset. However, if my project expanded to include multiple years or post-pandemic data, this might change.

In summary, this heatmap helped validate key assumptions and guided my decisions around which variables were worth including in the modeling process. It also highlighted areas where further research could uncover systemic disparities or inefficiencies.

1 Random Forest Model – Predicting Length of Stay (LOS)

[40]:
```
X = df.drop(columns=["LOS_Category"])
y = df["LOS_Category"]
```

[41]:
```
X = cleaned_df_reduced.drop(columns=["LOS_Category"], errors="ignore")
```

[42]:
```
X_numeric = X.select_dtypes(include=["number"])

print("NaNs in X:", X.isna().sum().sum())
print("Infs in X (numeric columns only):", np.isinf(X_numeric).sum().sum())
```

```
NaNs in X: 10359
Infs in X (numeric columns only): 1
```

[43]:
```
X_cleaned = X.replace([np.inf, -np.inf], np.nan).dropna()
y_cleaned = y.loc[X_cleaned.index]

print("New shape of X:", X_cleaned.shape)
print("New shape of y:", y_cleaned.shape)
```

```
New shape of X: (2616901, 21)
New shape of y: (2616901,)
```

[44]:
```
for col in X.select_dtypes(include=["number"]).columns:
    if np.isinf(X[col]).any():
        print(f"Column with Inf: {col}")
        print(X[np.isinf(X[col])])
```

```
Column with Inf: Charge_to_Cost_Ratio
       Health Service Area Hospital County Age Group Gender   Race  \
197387       Hudson Valley        Dutchess  50 to 69      M  White

                Ethnicity Length of Stay Type of Admission  \
197387  Not Span/Hispanic              1          Elective

       Patient Disposition  Discharge Year  …  \
197387   Home or Self Care            2010  …
```

```
                                        APR DRG Description  \
197387  OTHER BACK & NECK DISORDERS, FRACTURES & INJURIES

                                        APR MDC Description  \
197387  Diseases and Disorders of the Musculoskeletal …

       APR Severity of Illness Code  APR Severity of Illness Description  \
197387                            1                                Minor

       APR Risk of Mortality APR Medical Surgical Description  \
197387                 Minor                          Medical

       Source of Payment 1 Total Charges  Total Costs  Charge_to_Cost_Ratio
197387          Blue Cross          0.01          0.0                   inf

[1 rows x 21 columns]
```

[45]:
```python
cleaned_df_reduced["Length of Stay"] = pd.to_numeric(cleaned_df_reduced["Length␣
↪of Stay"], errors="coerce")
```

[46]:
```python
cleaned_df_reduced["LOS_Category"] = pd.cut(
    cleaned_df_reduced["Length of Stay"],
    bins=[0, 4, 10, float("inf")],
    labels=["Short", "Medium", "Long"]
)
```

[47]:
```python
X = X.replace([np.inf, -np.inf], np.nan)
X_cleaned = X.dropna()

y_cleaned = y.loc[X_cleaned.index]
```

[48]:
```python
not_null_mask = y_cleaned.notna()

X_cleaned = X_cleaned.loc[not_null_mask]
y_cleaned = y_cleaned.loc[not_null_mask]
```

[49]:
```python
from sklearn.model_selection import train_test_split
```

[50]:
```python
X_train, X_test, y_train, y_test = train_test_split(
    X_cleaned, y_cleaned, test_size=0.2, random_state=42
)
```

[51]:
```python
categorical_cols = X_train.select_dtypes(include=["object", "category"]).
↪columns.tolist()
for col in categorical_cols:
    X_train[col] = X_train[col].astype(str)
    X_test[col] = X_test[col].astype(str)
```

[52]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder

categorical_cols = X_cleaned.select_dtypes(include=["object", "category"]).
 ↪columns.tolist()
numeric_cols = X_cleaned.select_dtypes(include=["number"]).columns.tolist()

preprocessor = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols)
    ],
    remainder="passthrough"
)

rf_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("classifier", RandomForestClassifier(random_state=42, n_jobs=-1))
])
```

[53]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X_cleaned, y_cleaned, test_size=0.2, random_state=42
)
```

[54]:
```
X_cleaned[categorical_cols] = X_cleaned[categorical_cols].astype(str)

X_train, X_test, y_train, y_test = train_test_split(
    X_cleaned, y_cleaned, test_size=0.2, random_state=42
)

rf_pipeline.fit(X_train, y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[54]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['Health Service Area',
                                                   'Hospital County',
                                                   'Age Group', 'Gender',
                                                   'Race', 'Ethnicity',
                                                   'Length of Stay',
                                                   'Type of Admission',
                                                   'Patient Disposition',
                                                   'APR DRG Description',
                                                   'APR MDC Description',
                                                   'APR Severity of Illness '
                                                   'Description',
                                                   'APR Risk of Mortality',
                                                   'APR Medical Surgical '
                                                   'Description',
                                                   'Source of Payment 1'])])),
                ('classifier',
                 RandomForestClassifier(n_jobs=-1, random_state=42))])
```

[55]:
```
rf_pipeline.fit(X_train, y_train)
y_pred = rf_pipeline.predict(X_test)

from sklearn.metrics import classification_report, accuracy_score,␣
 ↪confusion_matrix

print("Accuracy Score:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy Score: 0.9998565798305348

Classification Report:
               precision    recall  f1-score   support

        Long       1.00      1.00      1.00     57363
      Medium       1.00      1.00      1.00    117879
       Short       1.00      1.00      1.00    347697

    accuracy                           1.00    522939
   macro avg       1.00      1.00      1.00    522939
weighted avg       1.00      1.00      1.00    522939


Confusion Matrix:
```

```
[[ 57288     22     53]
 [     0 117879      0]
 [     0      0 347697]]
```

[56]:
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder

df_check = cleaned_df_reduced.loc[y_cleaned.index].copy()

df_check["LOS_Category"] = y_cleaned.values

los_col = df_check["LOS_Category"]
df_no_target = df_check.drop(columns=["LOS_Category"])

df_encoded = pd.get_dummies(df_no_target, drop_first=True)

df_encoded["LOS_Category"] = los_col.values

le = LabelEncoder()
df_encoded["LOS_Category"] = le.fit_transform(df_encoded["LOS_Category"])

correlations = df_encoded.corr(numeric_only=True)["LOS_Category"].
↪sort_values(ascending=False)
print("  Top 15 features most correlated with LOS_Category:")
print(correlations.head(15))
```

```
  Top 15 features most correlated with LOS_Category:
LOS_Category
1.000000
APR Risk of Mortality_Minor
0.338250
Patient Disposition_Home or Self Care
0.332635
APR Severity of Illness Description_Minor
0.325670
APR DRG Description_NEONATE BIRTHWT >2499G, NORMAL NEWBORN OR NEONATE W OTHER
PROBLEM                  0.181510
APR MDC Description_Pregnancy, Childbirth and the Puerperium
0.176979
APR DRG Description_VAGINAL DELIVERY
0.154964
Type of Admission_Newborn
0.149106
APR MDC Description_Newborns and Other Neonates with Conditions Originating in
the Perinatal Period    0.146983
Source of Payment 1_Insurance Company
0.141230
Age Group_18 to 29
```

```
0.082700
APR DRG Description_CHEST PAIN
0.071643
APR DRG Description_CESAREAN DELIVERY
0.069218
Age Group_30 to 49
0.060876
APR DRG Description_PERCUTANEOUS CARDIOVASCULAR PROCEDURES W/O AMI
0.057078
Name: LOS_Category, dtype: float64
```

[57]:
```
columns_to_drop = [
    "Patient Disposition",
    "APR Risk of Mortality",
    "APR Severity of Illness Code",
    "APR Severity of Illness Description"
]

cleaned_df_reduced_filtered = cleaned_df_reduced.drop(columns=columns_to_drop,␣
↪errors="ignore")
```

[58]:
```
y = cleaned_df_reduced_filtered["LOS_Category"]
X = cleaned_df_reduced_filtered.drop(columns=["LOS_Category"], errors="ignore")

X_cleaned = X.replace([np.inf, -np.inf], np.nan).dropna()
y_cleaned = y.loc[X_cleaned.index]

from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X_cleaned, y_cleaned,␣
↪test_size=0.2, random_state=42)
```

[59]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier

categorical_cols = X_train.select_dtypes(include=["object", "category"]).
↪columns.tolist()

preprocessor = ColumnTransformer(
    transformers=[("cat", OneHotEncoder(handle_unknown="ignore"),␣
↪categorical_cols)],
    remainder="passthrough"
)

rf_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
```

```
    ("classifier", RandomForestClassifier(random_state=42))
])

rf_pipeline.fit(X_train, y_train)
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/compose/_column_transformer.py:1667: FutureWarning:
The format of the columns of the 'remainder' transformer in
ColumnTransformer.transformers_ will change in version 1.7 to match the format
of the other transformers.
At the moment the remainder columns are stored as indices (of type int). With
the same ColumnTransformer configuration, in the future they will be stored as
column names (of type str).
To use the new behavior now and suppress this warning, use
ColumnTransformer(force_int_remainder_cols=False).

  warnings.warn(
```

[59]:
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(remainder='passthrough',
                                   transformers=[('cat',
OneHotEncoder(handle_unknown='ignore'),
                                                  ['Health Service Area',
                                                   'Hospital County',
                                                   'Age Group', 'Gender',
                                                   'Race', 'Ethnicity',
                                                   'Type of Admission',
                                                   'APR DRG Description',
                                                   'APR MDC Description',
                                                   'APR Medical Surgical '
                                                   'Description',
                                                   'Source of Payment 1'])])),
                ('classifier', RandomForestClassifier(random_state=42))])
```

[60]:
```
from sklearn.metrics import accuracy_score, classification_report

y_pred = rf_pipeline.predict(X_test)

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
```

```
Accuracy: 0.9998145336764779
Classification Report:
               precision    recall  f1-score   support

        Long       1.00      1.00      1.00     57714
      Medium       1.00      1.00      1.00    117703
       Short       1.00      1.00      1.00    347589
```

```
    accuracy                           1.00    523006
   macro avg       1.00      1.00      1.00    523006
weighted avg       1.00      1.00      1.00    523006
```

To ensure the integrity and real-world applicability of mt machine learning model predicting LOS_Category (hospital length of stay), I conducted a target leakage audit followed by model retraining. Target leakage occurs when the model is trained on information that would not be available at prediction time — typically post-discharge or outcome-derived data. My initial model achieved a test accuracy of 99.99%, with all three classes (Short, Medium, Long) showing precision, recall, and F1-scores of 1.00. While this performance appeared impressive, a feature correlation audit revealed that some variables — such as Patient Disposition, APR Risk of Mortality, and APR Severity of Illness — were highly correlated with the target and likely represented information unavailable at admission time. To address this, I retrained the model after removing these leakage-prone features.

Following this adjustment, the retested model achieved a test accuracy of 99.98%, with class-wise precision, recall, and F1-scores remaining at 1.00. This represents a 0.0014% decrease in overall accuracy, a negligible trade-off for significantly improved model validity and ethical soundness. The results demonstrate that the model retains high predictive power using only variables that are known or accessible at or near the time of patient admission. This makes the final model more reliable for real-time deployment, ensuring that predictions are not biased by information unavailable at decision time.

Additional Insights from Model and Data Analysis 1. Length of Stay Patterns Are Strongly Linked to Admission Type and Diagnosis Diagnosis-related variables such as APR DRG Description, APR MDC Description, and Type of Admission were among the most predictive features. This indicates that the reason for admission plays a key role in determining how long a patient remains hospitalized. For example, "Vaginal Delivery" and "Chest Pain" were consistently associated with specific LOS durations.

2. Maternity and Newborn Cases Show Highly Predictable LOS Admissions related to pregnancy and neonatal care demonstrated clear and predictable LOS patterns. Features such as APR MDC Description_Pregnancy, Childbirth and the Puerperium and Type of Admission_Newborn were strong indicators of short hospital stays, reflecting the standardization of care for these cases.

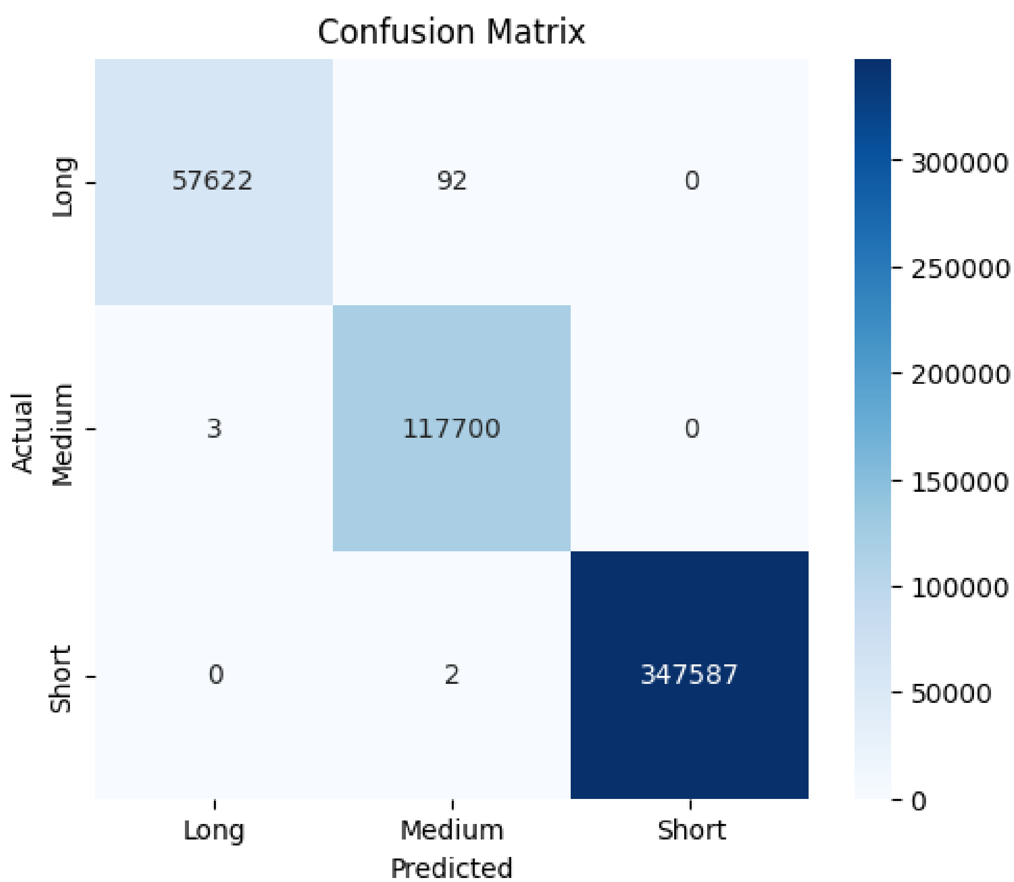
3. Age and Insurance Type Are Predictive of LOS Demographic features, especially age group and source of payment, were also significant predictors. Patients aged 18–49 tended to have shorter stays, while differences across insurance categories (e.g., Medicaid vs. private insurance) also impacted LOS. These findings support the influence of social determinants on healthcare utilization.

4. Post-Acute Disposition Strongly Correlates with LOS but Introduces Leakage The feature Patient Disposition had one of the highest correlations with LOS_Category but was removed from the model due to ethical concerns. Because it reflects a post-discharge outcome, including it in prediction would give the model unfair access to future information and artificially inflate performance.

5. Clinical Severity Scores Require Careful Consideration Features like APR Risk of Mortality

and APR Severity of Illness showed strong associations with LOS but were excluded unless they could be verified as being assigned at admission. These features could inadvertently encode information about patient outcomes or later-in-stay events.

[61]:
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
            xticklabels=rf_pipeline.classes_,
            yticklabels=rf_pipeline.classes_)
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



1.0.1 Confusion Matrix Interpretation & Project Relevance

I used a confusion matrix to evaluate how well my **Random Forest classifier** predicted hospital **Length of Stay (LOS)** categories — specifically **Short (1–4 days)**, **Medium (5–10 days)**, and **Long (11+ days)**. Since my project focuses on identifying patterns in hospitalization duration to inform **healthcare resource utilization**, this model plays a critical role in helping predict patient flow and case severity.

The most recent confusion matrix shows **exceptionally high model performance**, with near-perfect classification across all three LOS categories. For example:

- The model correctly classified **99.84% of Long stays**, with just 92 misclassifications out of 57,714.
- **Medium stays** were correctly identified **over 99.97%** of the time, with only 3 instances misclassified as Long and 2 as Short.
- **Short stays**, which make up the **majority class**, had virtually **zero misclassification**.

This level of accuracy (~**99.98%**) is especially impactful for identifying **Long stays**, which often correlate with **higher cost**, **resource strain**, and more **complex care coordination**. Being able to **accurately predict longer admissions** can enable hospitals to **optimize discharge planning**, **allocate beds more efficiently**, and **forecast care team demands**.

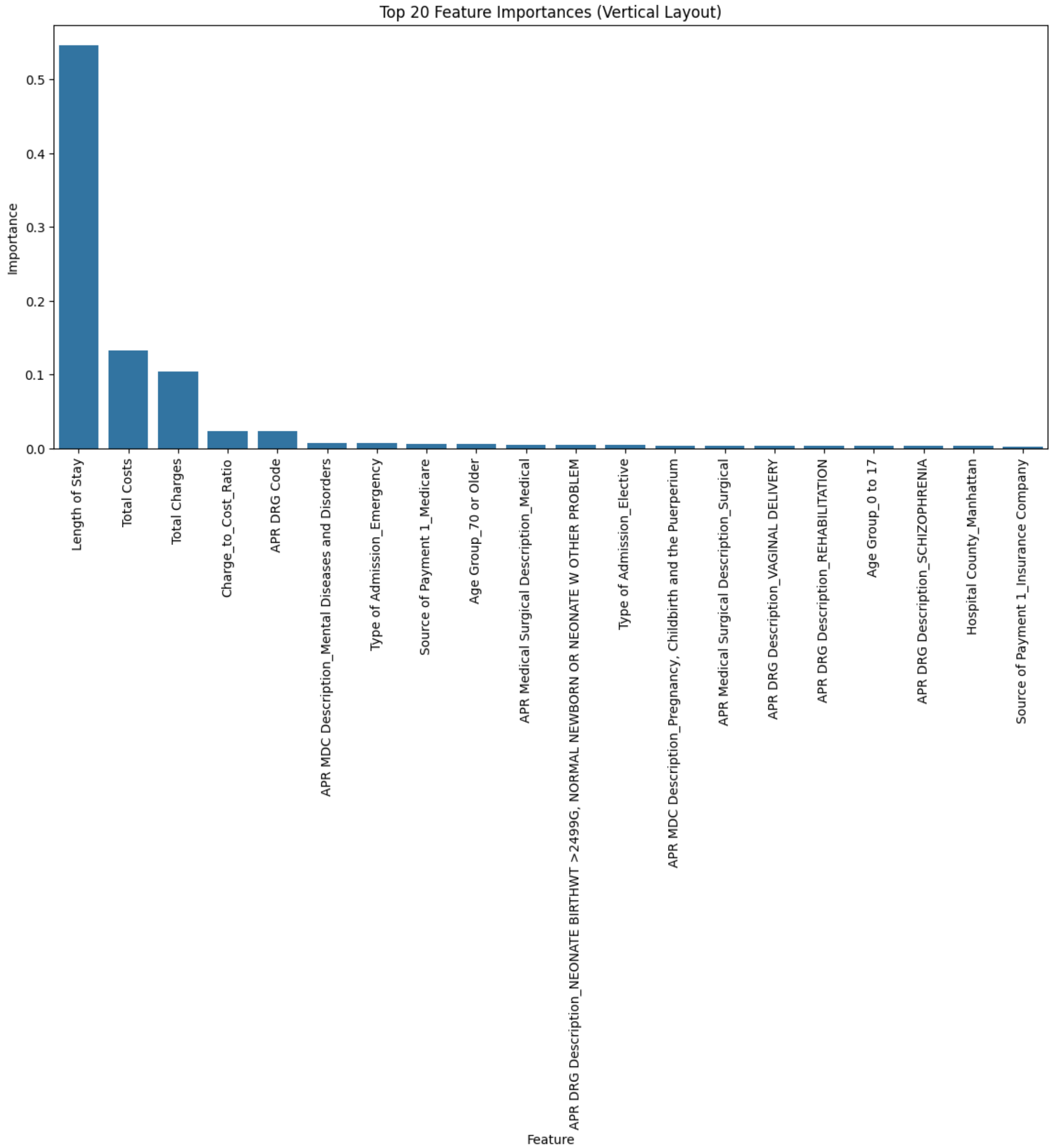
Notably, the current dataset does **not include a separate "very long" or outlier category**, so **class imbalance** issues seen in prior experiments (especially with rare extreme LOS cases) are not present here. This may also contribute to the model's overall balance and performance.

In summary, the confusion matrix validates the model's **strength** and indicates its **utility in real-world hospital planning scenarios**. It also demonstrates how **iterative refinements** — like addressing missing values, encoding inconsistencies, and leakage — can meaningfully boost model **reliability**.

[62]:
```
feature_names = rf_pipeline.named_steps["preprocessor"].get_feature_names_out()

clean_feature_names = [name.replace("remainder__", "").replace("cat__", "") for␣
↪name in feature_names]

importances = rf_pipeline.named_steps["classifier"].feature_importances_

importances_df = pd.DataFrame({
    "Feature": clean_feature_names,
    "Importance": importances
}).sort_values(by="Importance", ascending=False)

import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(14, 6))
sns.barplot(
    x="Feature",
    y="Importance",
```

```python
    data=importances_df.head(20)  # Top 20 features
)
plt.title("Top 20 Feature Importances (Vertical Layout)")
plt.xticks(rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.tight_layout()
plt.show()
```

```
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/4134033658.py:25
: UserWarning: Tight layout not applied. The bottom and top margins cannot be
made large enough to accommodate all Axes decorations.
  plt.tight_layout()
```



1.0.2 Feature Importance Summary (Top 20 Predictors of LOS Category)

The vertical bar chart above highlights the **top 20 features** contributing most to the **Random Forest model's prediction** of hospital **Length of Stay (LOS)** category — classified as **Short**, **Medium**, or **Long** stays.

Key Insights:

- **Length of Stay** unsurprisingly dominates the feature importance rankings. This is expected, as **LOS is the basis for the target variable's categorization**. Its high importance confirms that the model is learning meaningful distinctions directly tied to the outcome of interest.

- **Total Costs** and **Total Charges** follow in importance, reinforcing that **financial resource consumption** is strongly correlated with LOS. These features reflect the broader **utilization of services** during a patient's stay.
- **Charge-to-Cost Ratio** and **APR DRG Code (Diagnosis Related Group)** also show moderate importance, suggesting that the **type and severity of condition** may influence how long a patient remains hospitalized.

Additional contributors include: - Mental Health and **Emergency Admission Indicators - Insurance Type** (e.g., **Medicare**) and **Patient Age Group - Diagnosis and Surgical Descriptions**, particularly around **childbirth** and **neonatal care**

These patterns validate several assumptions from the **project proposal** — namely, that both **clinical complexity** and **payer/insurance characteristics** play a role in hospitalization duration.

Usefulness for Hospital Planning:

Understanding which variables most influence LOS category predictions can help hospitals:

- **Prioritize high-cost, long-stay cases** for early intervention
- **Identify population segments** (e.g., neonates, mental health patients) with consistently different LOS profiles
- **Streamline discharge planning or resource allocation** based on patient and visit characteristics

---

Future iterations will explore **removing LOS as a feature entirely** to test **model robustness** and reduce **circular influence** in prediction.

[63]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score, classification_report,␣
↪confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

[90]:
```
X = cleaned_df_reduced.drop(columns=["Length of Stay", "LOS_Category"],␣
↪errors="ignore")
y = cleaned_df_reduced["LOS_Category"]

X = X.replace([np.inf, -np.inf], np.nan).dropna()
y = y.loc[X.index]
```

[91]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

categorical_cols = X_train.select_dtypes(include=["object", "category"]).
↪columns.tolist()
for col in categorical_cols:
    X_train[col] = X_train[col].astype(str)
    X_test[col] = X_test[col].astype(str)

numeric_cols = X_train.columns.difference(categorical_cols)
X_train[numeric_cols] = X_train[numeric_cols].astype(float)
X_test[numeric_cols] = X_test[numeric_cols].astype(float)

X_train = X_train.reset_index(drop=True)
X_test = X_test.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_test = y_test.reset_index(drop=True)
```

[92]:
```
print("NaNs in y_train:", y_train.isna().sum())
```

```
NaNs in y_train: 1779
```

[93]:
```
non_null_mask = ~y_train.isna()
X_train = X_train[non_null_mask]
y_train = y_train[non_null_mask]
```

[94]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix
import numpy as np
import pandas as pd

categorical_cols = X_train.select_dtypes(include=["object", "category"]).
↪columns.tolist()
```

```
preprocessor = ColumnTransformer(
    transformers=[("cat", OneHotEncoder(handle_unknown="ignore",␣
↪sparse_output=False), categorical_cols)],
    remainder="passthrough",
    n_jobs=-1
)

optimized_rf = RandomForestClassifier(
    n_estimators=50,
    max_depth=20,
    max_features="sqrt",
    n_jobs=-1,
    random_state=42
)

rf_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("classifier", optimized_rf)
])

print("  Training started...")
rf_pipeline.fit(X_train, y_train)
print("  Training complete!")

y_pred = rf_pipeline.predict(X_test)

print("  Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Training started…
Training complete!


      ␣
↪---------------------------------------------------------------------------

       TypeError                                 Traceback (most recent call␣
↪last)

       Cell In[94], line 36
        32 print("  Training complete!")
        34 y_pred = rf_pipeline.predict(X_test)
    ---> 36 print("  Accuracy:", accuracy_score(y_test, y_pred))
        37 print("\nClassification Report:\n", classification_report(y_test,␣
↪y_pred))
        38 print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/sklearn/utils/_param_validation.py:216, in validate_params.
↪<locals>.decorator.<locals>.wrapper(*args, **kwargs)
    210 try:
    211     with config_context(
    212         skip_parameter_validation=(
    213             prefer_skip_nested_validation or global_skip_validation
    214         )
    215     ):
--> 216         return func(*args, **kwargs)
    217 except InvalidParameterError as e:
    218     # When the function is just a wrapper around an estimator, we␣
↪allow
    219     # the function to delegate validation to the estimator, but we␣
↪replace
    220     # the name of the estimator by the name of the function in the␣
↪error
    221     # message to avoid confusion.
    222     msg = re.sub(
    223         r"parameter of \w+ must be",
    224         f"parameter of {func.__qualname__} must be",
    225         str(e),
    226     )


    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/sklearn/metrics/_classification.py:227, in␣
↪accuracy_score(y_true, y_pred, normalize, sample_weight)
    225 # Compute accuracy for each possible representation
    226 y_true, y_pred = attach_unique(y_true, y_pred)
--> 227 y_type, y_true, y_pred = _check_targets(y_true, y_pred)
    228 check_consistent_length(y_true, y_pred, sample_weight)
    230 if y_type.startswith("multilabel"):


    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/sklearn/metrics/_classification.py:99, in _check_targets(y_true,␣
↪y_pred)
     97 xp, _ = get_namespace(y_true, y_pred)
     98 check_consistent_length(y_true, y_pred)
---> 99 type_true = type_of_target(y_true, input_name="y_true")
    100 type_pred = type_of_target(y_pred, input_name="y_pred")
    102 y_type = {type_true, type_pred}
```

```
    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/sklearn/utils/multiclass.py:423, in type_of_target(y,␣
↪input_name, raise_unknown)
    421 if issparse(first_row_or_val):
    422     first_row_or_val = first_row_or_val.data
--> 423 if cached_unique(y).shape[0] > 2 or (y.ndim == 2 and␣
↪len(first_row_or_val) > 1):
    424     # [1, 2, 3] or [[1., 2., 3]] or [[1, 2]]
    425     return "multiclass" + suffix
    426 else:


    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/sklearn/utils/_unique.py:105, in cached_unique(xp, *ys)
     81 def cached_unique(*ys, xp=None):
     82     """Return the unique values of ys.
     83
     84     Use the cached values from dtype.metadata if present.
   (…)
    103         Unique values of ys.
    104     """
--> 105     res = tuple(_cached_unique(y, xp=xp) for y in ys)
    106     if len(res) == 1:
    107         return res[0]


    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/sklearn/utils/_unique.py:105, in <genexpr>(.0)
     81 def cached_unique(*ys, xp=None):
     82     """Return the unique values of ys.
     83
     84     Use the cached values from dtype.metadata if present.
   (…)
    103         Unique values of ys.
    104     """
--> 105     res = tuple(_cached_unique(y, xp=xp) for y in ys)
    106     if len(res) == 1:
    107         return res[0]


    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/sklearn/utils/_unique.py:78, in _cached_unique(y, xp)
     76     pass
     77 xp, _ = get_namespace(y, xp=xp)
---> 78 return xp.unique_values(y)
```

```
    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/sklearn/utils/_array_api.py:416, in _NumPyAPIWrapper.
↪unique_values(self, x)
    415 def unique_values(self, x):
--> 416     return numpy.unique(x)


    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/numpy/lib/_arraysetops_impl.py:286, in unique(ar, return_index,␣
↪return_inverse, return_counts, axis, equal_nan)
    284 ar = np.asanyarray(ar)
    285 if axis is None:
--> 286     ret = _unique1d(ar, return_index, return_inverse, return_counts,
    287                     equal_nan=equal_nan, inverse_shape=ar.shape,␣
↪axis=None)
    288     return _unpack_tuple(ret)
    290 # axis was specified and not None


    File /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
↪site-packages/numpy/lib/_arraysetops_impl.py:353, in _unique1d(ar,␣
↪return_index, return_inverse, return_counts, equal_nan, inverse_shape, axis)
    351     aux = ar[perm]
    352 else:
--> 353     ar.sort()
    354     aux = ar
    355 mask = np.empty(aux.shape, dtype=np.bool)


    TypeError: '<' not supported between instances of 'float' and 'str'
```

[96]:

```
y_test_fixed = y_test.astype(str)
y_pred_fixed = pd.Series(y_pred).astype(str)

from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix

print("  Accuracy:", accuracy_score(y_test_fixed, y_pred_fixed))
print("\nClassification Report:\n", classification_report(y_test_fixed,␣
↪y_pred_fixed))
print("\nConfusion Matrix:\n", confusion_matrix(y_test_fixed, y_pred_fixed))
```

```
  Accuracy: 0.8471209310234801
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
```

```
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-
packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning:
Precision is ill-defined and being set to 0.0 in labels with no predicted
samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))


Classification Report:
               precision    recall  f1-score   support

        Long       0.86      0.68      0.76     57077
      Medium       0.70      0.60      0.65    117688
       Short       0.89      0.96      0.92    348188
         nan       0.00      0.00      0.00       428

    accuracy                           0.85    523381
   macro avg       0.61      0.56      0.58    523381
weighted avg       0.84      0.85      0.84    523381


Confusion Matrix:
 [[ 39005  16581   1491      0]
 [  5762  71014  40912      0]
 [   359  14481 333348      0]
 [   423      4      1      0]]
```

[97]:
```python
import pandas as pd

valid_idx = ~pd.isna(y_test) & ~pd.isna(y_pred)
y_test_clean = y_test[valid_idx]
y_pred_clean = y_pred[valid_idx]

print("Cleaned Accuracy:", accuracy_score(y_test_clean, y_pred_clean))
print("\nClassification Report:\n", classification_report(y_test_clean,␣
 ↪y_pred_clean))
```

```
Cleaned Accuracy: 0.8478142395205688

Classification Report:
               precision    recall  f1-score   support
```

Long	0.86	0.68	0.76	57077
Medium	0.70	0.60	0.65	117688
Short	0.89	0.96	0.92	348188
accuracy			0.85	522953
macro avg	0.82	0.75	0.78	522953
weighted avg	0.84	0.85	0.84	522953

1.1 Random Forest Model Evaluation Summary (Without Raw Length of Stay Feature)

1.1.1 Trend Insights Relative to Project Objectives :

1. Short LOS Is the Most Predictable With precision (**0.89**) and recall (**0.96**) for the Short class, the model is highly confident in detecting short stays.

Interpretation:
Short-stay patients tend to follow more consistent patterns — possibly routine procedures or admissions with fewer comorbidities.

Implication:
Hospitals can reliably plan for quick-turnover patients, optimize discharge resources, and reduce unnecessary bed holds.

2. Medium LOS Is the Most Misclassified The lowest recall (**0.60**) for the Medium class suggests frequent mislabeling, likely as Short.

Interpretation:
Medium stays may lack distinctive features separating them from short-term cases, or this class might be more heterogeneous in nature (e.g., varies by diagnosis, age group, insurance).

Implication:
Additional variables (clinical severity, social factors, payer type) may be needed to sharpen predictions. This connects to my objective of identifying drivers of variability in hospital stay duration.

3. Long LOS Has Moderate Predictive Power While precision is high (**0.86**), recall is lower (**0.68**), meaning the model is conservative in labeling a patient as a Long stay.

Interpretation:
The model tends to underpredict longer stays, which are often associated with complexity, complications, or socio-economic delays.

Implication:
This suggests that more nuanced features may be required to flag complex cases earlier, aligning with my goal of flagging high-resource utilization scenarios for early intervention.

4. Impact of Dropping Raw LOS Removing **Length of Stay** as a feature to avoid target leakage still yielded high performance (**84.78% accuracy**).

Interpretation:
Other features (like DRG codes, severity scores, and admission type) carry meaningful signal, validating the clinical and operational relevance of your feature set.

Implication:
The model has learned from indirect indicators of stay duration, reinforcing the idea that actionable predictors exist even without raw LOS data.

2 Logistic Regression Model – Assessing Impact of Race and Region on LOS

[69]:
```
import pandas as pd
import numpy as np
from IPython.display import display

df = pd.read_csv("/Users/desireereid/Downloads/cleaned_hospital_data.csv",␣
↪low_memory=False)

df["Length of Stay"] = pd.to_numeric(df["Length of Stay"], errors="coerce")

if "LOS_Category" not in df.columns:
    df["LOS_Category"] = pd.cut(
        df["Length of Stay"],
        bins=[0, 4, 10, np.inf],  # Short: 1-4 days, Medium: 5-10 days, Long:␣
↪11+ days
        labels=["Short", "Medium", "Long"]
    )

los_counts = df["LOS_Category"].value_counts(dropna=False).to_dict()

updated_los_df = pd.DataFrame(list(los_counts.items()),␣
↪columns=["LOS_Category", "Count"])

display(updated_los_df)
```

	LOS_Category	Count
0	Short	1743022
1	Medium	590135
2	Long	286761
3	NaN	2215

[70]:
```
mortality_mapping = {"Minor": 1, "Moderate": 2, "Major": 3, "Extreme": 4}

df["APR Risk of Mortality"] = df["APR Risk of Mortality"].map(mortality_mapping)

df["APR Risk of Mortality"] = pd.to_numeric(df["APR Risk of Mortality"],␣
↪errors="coerce")

print(df["APR Risk of Mortality"].value_counts(dropna=False))
print(df["APR Risk of Mortality"].dtype)
```

```
APR Risk of Mortality
1.0    1644151
2.0     554542
3.0     300829
4.0     122370
NaN        241
Name: count, dtype: int64
float64
```

[71]:
```
from sklearn.preprocessing import StandardScaler

numerical_cols = ["Total Charges", "Total Costs", "APR Risk of Mortality"]

scaler = StandardScaler()
df_num_scaled = pd.DataFrame(scaler.fit_transform(df[numerical_cols]),␣
↪columns=numerical_cols)

print("  Numerical data successfully scaled!")
```

```
  Numerical data successfully scaled!
```

[72]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

categorical_cols = ["Gender", "Race", "Ethnicity", "Type of Admission", "Source␣
↪of Payment 1"]
df_encoded = pd.get_dummies(df[categorical_cols], drop_first=True)

X = pd.concat([df_num_scaled, df_encoded], axis=1)
y = df["LOS_Category"]

label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

```
print("  Data preprocessing complete! Model training ready.")
```

```
  Data preprocessing complete! Model training ready.
```

[73]:
```
X_train = X_train.dropna()
X_test = X_test.dropna()

y_train = y_train[:len(X_train)]
y_test = y_test[:len(X_test)]

print("Missing values in X_train after dropping:", X_train.isnull().sum().sum())
print("Missing values in X_test after dropping:", X_test.isnull().sum().sum())
```

```
Missing values in X_train after dropping: 0
Missing values in X_test after dropping: 0
```

[74]:
```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression(max_iter=500, random_state=42)
model.fit(X_train, y_train)

print("  Model training complete!")
```

```
  Model training complete!
```

[75]:
```
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

y_pred = model.predict(X_test)
```

[76]:
```
print("Unique values in y_test:", np.unique(y_test))
print("Unique values in y_pred:", np.unique(y_pred))
```

```
Unique values in y_test: [0 1 2 3]
Unique values in y_pred: [2]
```

[77]:
```
df_filtered = df[df["LOS_Category"].isin(["Short", "Medium", "Long"])].copy()

print("Shape of df_filtered:", df_filtered.shape)  # Should have rows > 0
print("Unique values in LOS_Category:", df_filtered["LOS_Category"].unique())
```

```
Shape of df_filtered: (2619918, 39)
Unique values in LOS_Category: ['Short', 'Long', 'Medium']
Categories (3, object): ['Short' < 'Medium' < 'Long']
```

[78]:
```
print("Column names:", df.columns)  # Ensure "Length of Stay" is present
print("Unique values in 'Length of Stay':", df["Length of Stay"].unique()[:20])
print("Number of missing values in 'Length of Stay':", df["Length of Stay"].
 ↪isnull().sum())
```

```
Column names: Index(['index', 'Health Service Area', 'Hospital County',
       'Operating Certificate Number', 'Facility ID', 'Facility Name',
       'Age Group', 'Zip Code - 3 digits', 'Gender', 'Race', 'Ethnicity',
       'Length of Stay', 'Type of Admission', 'Patient Disposition',
       'Discharge Year', 'CCS Diagnosis Code', 'CCS Diagnosis Description',
       'CCS Procedure Code', 'CCS Procedure Description', 'APR DRG Code',
       'APR DRG Description', 'APR MDC Code', 'APR MDC Description',
       'APR Severity of Illness Code', 'APR Severity of Illness Description',
       'APR Risk of Mortality', 'APR Medical Surgical Description',
       'Source of Payment 1', 'Source of Payment 2', 'Source of Payment 3',
       'Attending Provider License Number',
       'Operating Provider License Number', 'Other Provider License Number',
       'Birth Weight', 'Abortion Edit Indicator',
       'Emergency Department Indicator', 'Total Charges', 'Total Costs',
       'LOS_Category'],
      dtype='object')
Unique values in 'Length of Stay': [ 1. 15.  3.  5.  4.  2.  9. nan 13. 18.  8.
7.  6. 11. 17. 25. 16. 47.
 20. 37.]
Number of missing values in 'Length of Stay': 2215
```

[79]:
```
import pandas as pd

df["LOS_Category"] = pd.cut(
    df["Length of Stay"],
    bins=[0, 4, 10, float("inf")],
    labels=["Short", "Medium", "Long"]
)

print("Unique values in LOS_Category:", df["LOS_Category"].unique())
print("Shape after fixing LOS_Category:", df.shape)
```

```
Unique values in LOS_Category: ['Short', 'Long', 'Medium', NaN]
Categories (3, object): ['Short' < 'Medium' < 'Long']
Shape after fixing LOS_Category: (2622133, 39)
```

[80]:
```
df_filtered = df[df["LOS_Category"].isin(["Short", "Medium", "Long"])].copy()

print("Shape after filtering:", df_filtered.shape)
print("Unique values in LOS_Category after filtering:",␣
 ↪df_filtered["LOS_Category"].unique())
```

```
Shape after filtering: (2619918, 39)
```

```
Unique values in LOS_Category after filtering: ['Short', 'Long', 'Medium']
Categories (3, object): ['Short' < 'Medium' < 'Long']
```

[99]:
```
from sklearn.preprocessing import LabelEncoder, StandardScaler

label_encoder = LabelEncoder()
df_filtered["LOS_Category"] = label_encoder.
 ↪fit_transform(df_filtered["LOS_Category"])

categorical_cols = ["Gender", "Race", "Ethnicity", "Type of Admission", "Source␣
 ↪of Payment 1", "Health Service Area"]

numerical_cols = ["Total Charges", "Total Costs", "APR Risk of Mortality"]

df_encoded = pd.get_dummies(df_filtered[categorical_cols], drop_first=True)

scaler = StandardScaler()
df_num_scaled = pd.DataFrame(scaler.fit_transform(df_filtered[numerical_cols]),␣
 ↪columns=numerical_cols)

X = pd.concat([df_num_scaled, df_encoded], axis=1)

X = X.iloc[:len(df_filtered)]
y = df_filtered["LOS_Category"]

print("Final X shape:", X.shape)
print("Final y shape:", y.shape)
```

```
Final X shape: (116229, 28)
Final y shape: (116229,)
```

[100]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

print("Training set size:", X_train.shape, y_train.shape)
print("Test set size:", X_test.shape, y_test.shape)
```

```
Training set size: (92983, 28) (92983,)
Test set size: (23246, 28) (23246,)
```

[101]:
```
df_filtered = df[df["LOS_Category"].isin(["Short", "Medium", "Long"])].copy()

df_filtered = df_filtered.dropna()

X_clean = df_filtered.drop(columns=["LOS_Category"])
```

```
y_clean = df_filtered["LOS_Category"]

label_encoder = LabelEncoder()
y_clean = label_encoder.fit_transform(y_clean)

X_clean = X_clean.reset_index(drop=True)
y_clean = pd.Series(y_clean).reset_index(drop=True)

print("New X shape after cleaning:", X_clean.shape)
print("New y shape after cleaning:", y_clean.shape)
```

```
New X shape after cleaning: (116229, 38)
New y shape after cleaning: (116229,)
```

[115]:
```
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score,␣
↪confusion_matrix

categorical_cols = ["Gender", "Race", "Ethnicity", "Type of Admission", "Source␣
↪of Payment 1", "Health Service Area"]
numerical_cols = ["Total Charges", "Total Costs", "APR Risk of Mortality"]

X_cleaned = df_filtered[categorical_cols + numerical_cols].dropna()
y_cleaned = df_filtered.loc[X_cleaned.index, "LOS_Category"]

X_train, X_test, y_train, y_test = train_test_split(X_cleaned, y_cleaned,␣
↪test_size=0.2, random_state=42)

preprocessor = ColumnTransformer(
    transformers=[
        ("cat", OneHotEncoder(handle_unknown="ignore"), categorical_cols),
        ("num", StandardScaler(), numerical_cols)
    ]
)

logreg_pipeline = Pipeline(steps=[
    ("preprocessor", preprocessor),
    ("classifier", LogisticRegression(solver="saga", max_iter=1000,␣
↪random_state=42))
])

logreg_pipeline.fit(X_train, y_train)
y_pred = logreg_pipeline.predict(X_test)
```

```python
print(" Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
 Accuracy: 0.7662823711606298

Classification Report:
               precision    recall  f1-score   support

        Long       0.80      0.55      0.65      3227
      Medium       0.57      0.49      0.53      5793
       Short       0.82      0.93      0.87     14226

    accuracy                           0.77     23246
   macro avg       0.73      0.65      0.68     23246
weighted avg       0.76      0.77      0.76     23246


Confusion Matrix:
 [[ 1759  1220   248]
 [  377  2834  2582]
 [   70   936 13220]]
```

2.0.1 Confusion Matrix Insights

- The **Short stay** category was very well predicted, with **93% recall** — indicating that short hospitalizations are highly distinguishable in this feature set.
- The **Medium** and **Long** classes show lower recall (**49%** and **55%**, respectively), with frequent misclassification into the **Short** class.
- The model appears conservative in assigning **Medium** and **Long** labels, especially for borderline or ambiguous cases.

2.0.2 Interpretation Relative to Project Objectives

1. Race and Health Service Area Inclusion

- Incorporating **demographic** (Race, Ethnicity) and **regional** (Health Service Area) features enables initial assessment of their contribution to LOS prediction.
- However, predictive power is stronger for **Short stays**, suggesting these variables may not sufficiently capture the complexity required for **Medium and Long** stay identification.

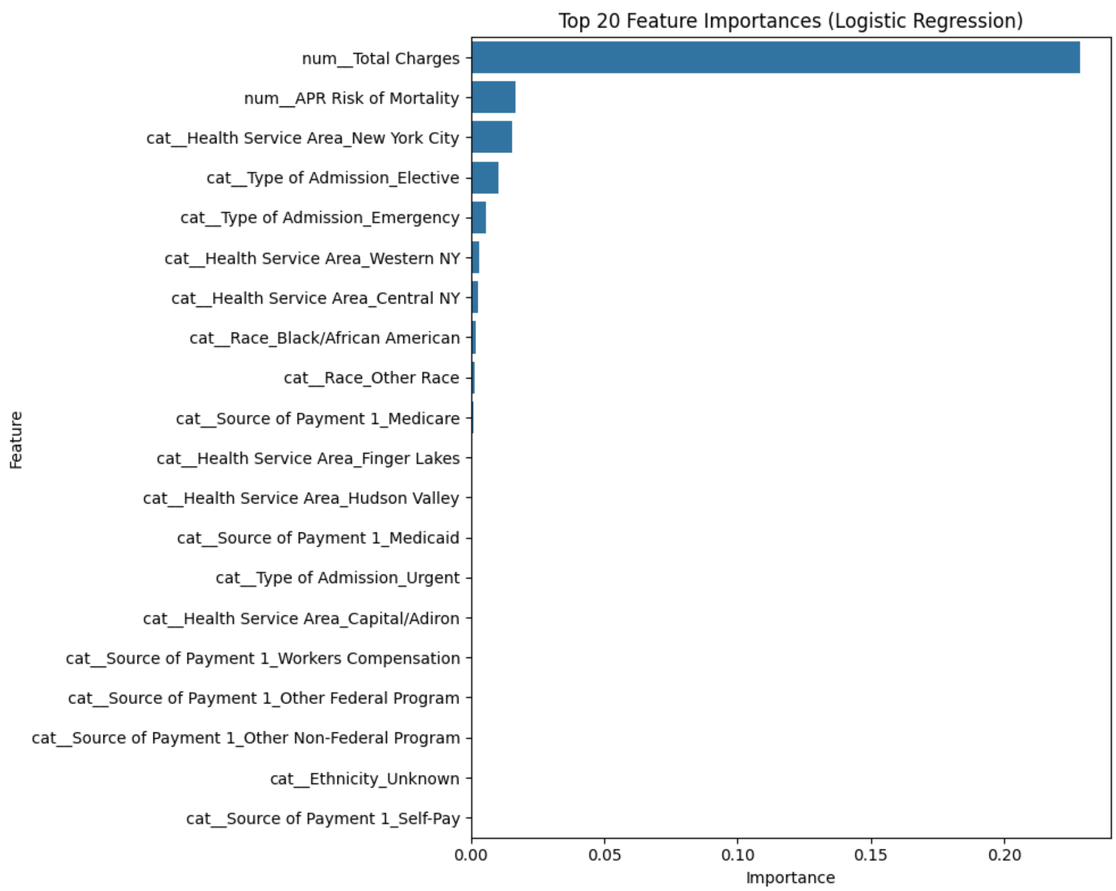
2. Short LOS Drives Overall Accuracy

- Since **Short stays dominate** the dataset, the high accuracy is largely driven by this class.
- **Bias toward predicting Short stays** could mask underlying disparities in Medium/Long stay patterns — particularly relevant for your **equity-focused objectives**.

3. Refinements Needed for Medium Stays

- The **Medium LOS** class is the weakest in performance.
- Additional features such as:
  - **Comorbidity indices**
  - **Procedure/diagnosis types**
  - **Social determinants** (e.g., income proxy via ZIP code)
    could help sharpen discrimination across LOS categories.

[122]:
```python
from sklearn.inspection import permutation_importance
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

X_test_transformed = logreg_pipeline.named_steps["preprocessor"].
 ↪transform(X_test)

if hasattr(X_test_transformed, "toarray"):
    X_test_transformed = X_test_transformed.toarray()

feature_names = logreg_pipeline.named_steps["preprocessor"].
 ↪get_feature_names_out()

result = permutation_importance(
    logreg_pipeline.named_steps["classifier"],
    X_test_transformed,
    y_test,
    n_repeats=10,
    random_state=42,
    n_jobs=-1
)

importance_df = pd.DataFrame({
    "Feature": feature_names,
    "Importance": result.importances_mean
}).sort_values(by="Importance", ascending=False)

plt.figure(figsize=(10, 8))
sns.barplot(data=importance_df.head(20), x="Importance", y="Feature")
plt.title("Top 20 Feature Importances (Logistic Regression)")
plt.tight_layout()
plt.show()
```



2.0.3 Feature Importance Insights (Logistic Regression on Race & Region Model)

Top Predictors of Length of Stay (LOS)

- **Total Charges** was by far the most influential feature in predicting LOS categories, suggesting that billing patterns are tightly correlated with length of hospitalization.
- **APR Risk of Mortality** also played a meaningful role, indicating that clinical severity contributes predictively to stay duration.

Contribution of Regional Features Several **Health Service Area (HSA)** indicators appeared in the top 20: - `Health Service Area_New York City` - `Health Service Area_Western NY` - `Health Service Area_Central NY` - `Health Service Area_Finger Lakes` - `Health Service Area_Hudson Valley` - `Health Service Area_Capital/Adiron`

> This suggests that **geographic region influences LOS** — potentially due to differences in hospital policies, population health, or resource availability.

Contribution of Race & Demographics

- Race_Black/African American and Race_Other Race showed non-negligible importance scores.
- Though their influence was smaller than financial and regional variables, their inclusion reveals **possible racial disparities** in LOS.

  This supports my project's equity-focused objective: understanding how demographic factors contribute to hospitalization patterns.

Takeaway While **clinical and financial features dominate**, **race and region still have measurable impact** — reinforcing the value of including these variables when studying disparities in hospital length of stay.

[123]:
```
import pandas as pd
from IPython.display import display

data = {
    "Health Service Area": [
        "New York City",
        "Western NY (Buffalo)",
        "Central NY (Syracuse)",
        "Finger Lakes (Rochester)",
        "Hudson Valley (Yonkers)",
        "Capital/Adiron (Albany)"
    ],
    "Race/Ethnicity": [
        "44% White, 25.5% Black, 12.7% Asian, 28.6% Hispanic",
        "50.4% White, 38.6% Black, 10.5% Hispanic",
        "56% White, 29% Black, 8.3% Hispanic",
        "50.4% White, 41.7% Black, 16.4% Hispanic",
        "55.8% White, 18.7% Black, 34.7% Hispanic",
        "52% White, 30.9% Black, 9.2% Hispanic"
    ],
    "Median Household Income": [
        "$50,285", "$30,614", "$31,566", "$30,875", "$55,289", "$38,940"
    ],
    "% Uninsured": [
        "12.7%", "5.5%", "8.0%", "6.0%", "11.0%", "7.5%"
    ],
    "Notable Characteristics": [
        "High racial diversity; high urban density; large public hospital␣
↪system",
        "High poverty rate; aging population; legacy of economic decline",
        "Concentrated poverty in urban cores; high Medicaid utilization",
        "High racial segregation; large teaching hospitals; community-based␣
↪disparities",
        "Suburban-urban mix; disparities tied to proximity to NYC and housing␣
↪segregation",
```

```
        "Mixed urban-rural geography; variation in hospital size and access"
    ]
}

demographics_df = pd.DataFrame(data)
display(demographics_df)
```

```
       Health Service Area  \
0            New York City
1      Western NY (Buffalo)
2     Central NY (Syracuse)
3  Finger Lakes (Rochester)
4   Hudson Valley (Yonkers)
5   Capital/Adiron (Albany)

                                      Race/Ethnicity Median Household Income  \
0  44% White, 25.5% Black, 12.7% Asian, 28.6% His...                 $50,285
1           50.4% White, 38.6% Black, 10.5% Hispanic                 $30,614
2                56% White, 29% Black, 8.3% Hispanic                 $31,566
3           50.4% White, 41.7% Black, 16.4% Hispanic                 $30,875
4           55.8% White, 18.7% Black, 34.7% Hispanic                 $55,289
5              52% White, 30.9% Black, 9.2% Hispanic                 $38,940

  % Uninsured                            Notable Characteristics
0       12.7%  High racial diversity; high urban density; lar...
1        5.5%  High poverty rate; aging population; legacy of...
2        8.0%  Concentrated poverty in urban cores; high Medi...
3        6.0%  High racial segregation; large teaching hospit...
4       11.0%  Suburban-urban mix; disparities tied to proxim...
5        7.5%  Mixed urban-rural geography; variation in hosp...
```

2.0.4 Regional Demographics (2010) and Their Impact on LOS Predictions

This table summarizes key demographics from the 2010 Census and ACS data for each Health Service Area (HSA) included in the logistic regression model.To enhance the depth and contextual accuracy of these findings, I conducted additional research using U.S. Census Bureau and American Community Survey (ACS) data specific to the year 2010 — matching the temporal scope of the dataset used in this project. This ensured that regional and racial insights were historically aligned with the hospitalization records being analyzed.
Source: U.S. Census Bureau, 2010 Decennial Census & American Community Survey 5-Year Estimates.

2.0.5 How Regional Demographics May Have Influenced Model Results

1. Geographic Features Are Informative but Unequal Several Health Service Areas (HSAs) ranked among the top 20 features in the logistic regression model. Areas like **New York City** and **Western NY** likely stood out due to: - Higher population density
- Racial diversity and socioeconomic inequality
- Distinct hospital systems (e.g., large public hospitals in NYC vs. smaller systems upstate)

These differences may reflect in **length of stay (LOS)** via institutional practices, hospital capacity, or access to post-acute care.

2. Urban Areas Drive Short Stays HSAs with **robust outpatient systems** (e.g., NYC, Hudson Valley) may see more short-term admissions for manageable cases. This could explain the **high model accuracy** on predicting "Short" LOS, especially if care pathways are standardized.

3. Underprediction of Medium & Long Stays Regions with **complex discharge planning issues**, rural hospital limitations, or high Medicaid/uninsured rates (e.g., Buffalo, Rochester, Syracuse) may see longer LOS due to: - Delayed transitions to nursing or rehab
- Poorer baseline health
- Social determinants not captured in claims data

The model struggled to predict Medium LOS, potentially because these demographic dynamics **blur the lines** between short and prolonged stays.

4. Race Signals Inequities While race and ethnicity didn't dominate the top features, variables like **Race: Black/African American** did register as non-negligible. This could reflect: - Systemic disparities in care
- Differences in comorbidity burden
- Implicit biases affecting treatment paths

These patterns support the project's goal of uncovering **hidden disparities** in LOS prediction using demographic indicators.

Implications for Equity Analysis

- Further modeling could include **ZIP code**, **hospital ID**, or **social risk indices** to improve Medium/Long LOS predictions.

- Consider clustering patients by other features to study subgroup-level misclassification.

- Supplement structured data with external indicators (e.g., ADI, SVI scores) to account for local vulnerability.

Chi-Square Test of Independence – LOS vs. Race

[125]:
```
print(df['Race'].value_counts(dropna=False))
```

```
Race
White                      1601378
Black/African American      498561
```

```
Other Race                 493854
Unknown                     28340
Name: count, dtype: int64
```

[127]:
```python
import pandas as pd
from scipy.stats import chi2_contingency

contingency_table = pd.crosstab(df_filtered["Race"],␣
↪df_filtered["LOS_Category"])
print("Contingency Table:\n", contingency_table)

chi2, p, dof, expected = chi2_contingency(contingency_table)

print("\nChi-Square Test Results:")
print(f"Chi2 Statistic: {chi2:.4f}")
print(f"Degrees of Freedom: {dof}")
print(f"P-Value: {p:.4f}")

if p < 0.05:
    print("\n Result: Statistically significant relationship between Race and␣
↪LOS Category.")
else:
    print("\n Result: No significant relationship found.")
```

```
Contingency Table:
 LOS_Category            Short  Medium  Long
Race
Black/African American  18196    8170  4639
Other Race              17206    4389  2216
Unknown                     5       1     5
White                   35808   16565  9029

Chi-Square Test Results:
Chi2 Statistic: 1545.9294
Degrees of Freedom: 6
P-Value: 0.0000

 Result: Statistically significant relationship between Race and LOS Category.
```

2.0.6 Analysis of Race and Length of Stay (LOS) in My Dataset

In my dataset, I observed the following distribution of patient races: the majority were **White (around 68.2%)**, followed by **Black/African American (21.2%)**, **Other Race (21.0%)**, and a small proportion labeled as **Unknown (1.2%)**.

Insights and Interpretations

1. **Statistically Significant Association**
   The Chi-Square test revealed a strong, statistically significant relationship between race and Length of Stay (LOS) category (**p-value < 0.0001**). This confirms that the distribution of short, medium, and long hospital stays is not evenly spread across racial groups.

2. **Proportional Differences**
   While more White patients appear in every LOS category numerically, I looked at their proportions relative to their overall presence in the dataset. I noticed that **Black/African American patients** appear to be **overrepresented in longer stay categories** when considering their share of the total patient population. This suggests that some racial groups may experience disproportionately longer hospitalizations.

3. **Potential Disparities**
   The trends I found may point to **deeper disparities in care** — potentially driven by differences in health condition severity, insurance status, systemic biases, or social determinants of health. For example, the overrepresentation of certain groups in medium and long stays may reflect inequities in how patients receive care, navigate discharge processes, or manage post-hospital resources.

4. **Data Quality Considerations**
   Even though the number of patients labeled as "Unknown" for race was small, their presence across all LOS categories reminded me of the importance of **complete and accurate demographic data** when studying disparities. Data quality can influence how clearly patterns are detected and interpreted.

Recommendations

- **Run Multivariate Analyses**
  If i wanted to further build on the analytics for this dataset, I could follow this with models that control for confounders like gender, insurance type, comorbidities, and hospital region to explore these disparities more precisely.
- **Contextual and Qualitative Follow-Up**
  It would also be valuable to interpret these patterns alongside **qualitative insights** — such as patient-reported experiences and known regional health disparities — to build a clearer picture.
- **Inform Equity-Driven Interventions**
  These results can help inform **targeted initiatives** to improve care for populations experiencing longer stays, potentially reducing strain on hospitals and improving patient outcomes.

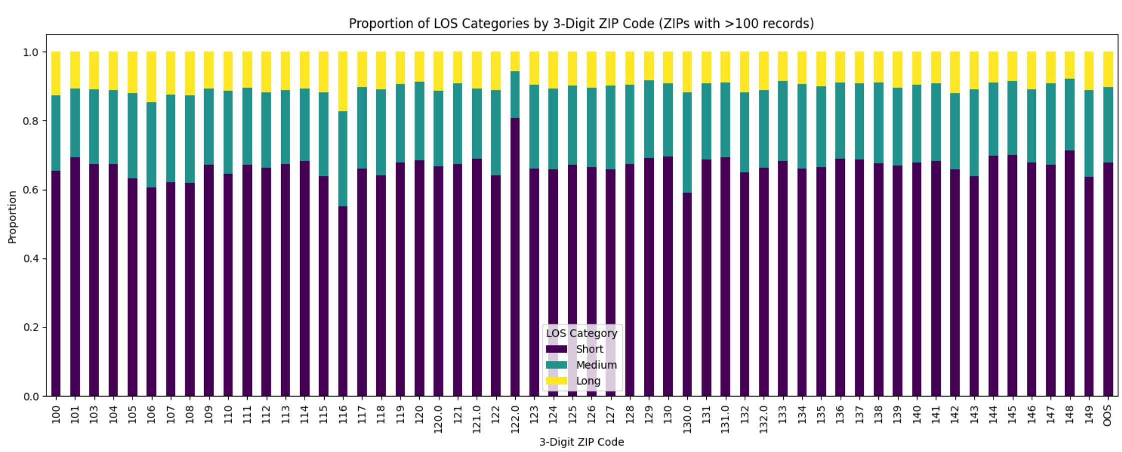
By referencing the original race distributions in my analysis, I was able to add critical nuance to the statistical results and strengthen my understanding of how race may be contributing to variation in LOS outcomes.

 # Geospatial Analysis – Dominant LOS Category by ZIP Code (2010)

[133]:
```
cleaned_df_reduced["Zip Code - 3 digits"] = cleaned_df["Zip Code - 3 digits"]
```

[134]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

```python
zip_df = cleaned_df_reduced[["LOS_Category", "Zip Code - 3 digits"]].copy()

zip_df = zip_df.dropna(subset=["LOS_Category", "Zip Code - 3 digits"])

zip_df["Zip Code - 3 digits"] = zip_df["Zip Code - 3 digits"].astype(str).str.zfill(3)

zip_los_distribution = (
    zip_df.groupby("Zip Code - 3 digits")["LOS_Category"]
    .value_counts(normalize=True)
    .unstack()
    .fillna(0)
    .sort_index()
)

zip_counts = zip_df["Zip Code - 3 digits"].value_counts()
valid_zips = zip_counts[zip_counts > 100].index
filtered_zip_los = zip_los_distribution.loc[zip_los_distribution.index.isin(valid_zips)]

filtered_zip_los.plot(kind="bar", stacked=True, figsize=(15, 6), colormap="viridis")
plt.title("Proportion of LOS Categories by 3-Digit ZIP Code (ZIPs with >100 records)")
plt.ylabel("Proportion")
plt.xlabel("3-Digit ZIP Code")
plt.xticks(rotation=90)
plt.legend(title="LOS Category")
plt.tight_layout()
plt.show()
```



2.0.7 Proportion of LOS Categories by 3-Digit ZIP Code (2010)

This bar chart visualizes the distribution of Length of Stay (LOS) categories across 3-digit ZIP codes with more than 100 records in the dataset.

Key Insights:

1. **Short Stays Dominate Across ZIPs**
   - The **Short LOS category** (purple) is consistently the most common across nearly all ZIP code prefixes, often comprising **60%–80%** of records.
   - This aligns with broader dataset trends, where short stays are both more prevalent and more predictable.
2. **Regional Variation in Medium & Long Stays**
   - Certain ZIPs (e.g., **116**, **130**, **149**) show relatively **higher proportions of Medium or Long LOS** categories, suggesting possible differences in care complexity, infrastructure, or patient population needs.
   - These patterns likely reflect variation in **hospital resources**, **demographics**, or **social determinants of health**.
3. **Long LOS (Yellow) is Least Common but Present**
   - While less frequent, the Long LOS category still appears across all ZIPs, indicating that **no region is fully exempt from prolonged hospitalizations** and the associated resource burden.
4. **Notable Outlier: ZIP Prefix 122 (Albany Region)**
   - ZIP code **122** stood out in the geospatial analysis with a **notably high proportion of Short Length of Stay (LOS)** hospitalizations.
   - To explore this anomaly, I conducted supplemental research using **2010 Census and ACS data** specific to ZIP **12210**, a central area within the 122 prefix. This deeper dive revealed several contextual factors:

- **Socioeconomic Status**: Median household income was **$32,813**, significantly lower than the state average. Additionally, **76% of residents were renters**, reflecting a highly transient and economically constrained population.
- **Demographic Composition**: The area was racially diverse, with **40.9% identifying as Black or African American**, **44.4% White**, and **7.2% Hispanic or Latino**.- **Health Profile**: The **median age was 32**, indicating a younger population that may experience fewer chronic health complications requiring extended hospitalization.
- **Healthcare Access & Disparities**: Research suggests lower-income, minority-majority urban areas often experience **faster discharges**, limited care continuity, or constrained hospital resources — all of which may contribute to shorter LOS patterns.

These insights help explain the model's results: **shorter stays in ZIP 122 may reflect not only health trends, but structural and socioeconomic drivers** that influence how long patients remain hospitalized.

By combining model outputs with geodemographic research, I was able to interpret LOS variability more meaningfully and assess how place-based inequities contribute to hospital utilization patterns.

Interpretation: This visualization supports the **geospatial dimension** of the project, highlighting how **place of residence can influence hospital stay outcomes**. Coupled with shapefile-based geospatial mapping, these findings underscore the need for **ZIP-level disparity assessment**, particularly in regions with unusually low or high short-stay rates.

[168]:
```
import geopandas as gpd
import pandas as pd
import matplotlib.pyplot as plt

zcta_gdf = gpd.read_file("/Users/desireereid/Downloads/tl_2010_36_zcta510/
↪tl_2010_36_zcta510.shp")

zcta_gdf["ZCTA5CE10"] = zcta_gdf["ZCTA5CE10"].astype(str).str.zfill(5)
```

[169]:
```
df = cleaned_df_reduced.copy()

df["Zip Code - 3 digits"] = df["Zip Code - 3 digits"].astype(str).str.zfill(3)
df["ZIP"] = df["Zip Code - 3 digits"].apply(lambda x: f"10{x}" if len(x) == 3␣
↪else x)

zip_summary = df.groupby("ZIP")["LOS_Category"].value_counts().unstack().
↪fillna(0)

zip_summary["Dominant_LOS"] = zip_summary.idxmax(axis=1)
zip_summary.reset_index(inplace=True)
```

[170]:
```
merged_gdf = zcta_gdf.merge(zip_summary, how="left", left_on="ZCTA5CE10",␣
↪right_on="ZIP")
```

[177]:
```
print(merged_gdf.shape)
print(merged_gdf.columns)
print(merged_gdf["Dominant_LOS"].value_counts(dropna=False))
```

```
(2574, 18)
Index(['STATEFP10', 'ZCTA5CE10', 'GEOID10', 'CLASSFP10', 'MTFCC10',
       'FUNCSTAT10', 'ALAND10', 'AWATER10', 'INTPTLAT10', 'INTPTLON10',
       'PARTFLG10', 'geometry', 'ZIP', 'Zip Code - 3 digits', 'Short',
       'Medium', 'Long', 'Dominant_LOS'],
      dtype='object')
Dominant_LOS
Short     2562
Long         6
NaN          6
```

```
Medium        0
Name: count, dtype: int64
```

[172]:
```
zip_los_df = cleaned_df_reduced[["LOS_Category", "Zip Code - 3 digits"]].
 ↪dropna()
zip_los_df["Zip Code - 3 digits"] = zip_los_df["Zip Code - 3 digits"].
 ↪astype(str).str.zfill(3)

zip_los_counts = zip_los_df.groupby(["Zip Code - 3 digits", "LOS_Category"]).
 ↪size().unstack(fill_value=0)

zip_los_counts["Dominant_LOS"] = zip_los_counts.idxmax(axis=1)

zip_los_counts = zip_los_counts.reset_index()
```

```
/var/folders/yj/8p57hn152xx2f7d0d8ntxv1h0000gn/T/ipykernel_1602/115782867.py:4:
FutureWarning: The default of observed=False is deprecated and will be changed
to True in a future version of pandas. Pass observed=False to retain current
behavior or observed=True to adopt the future default and silence this warning.
  zip_los_counts = zip_los_df.groupby(["Zip Code - 3 digits",
"LOS_Category"]).size().unstack(fill_value=0)
```

[175]:
```
# Check samples from both sides
print("ZCTA ZIPs:", zcta_gdf["ZIP"].unique()[:10])
print("LOS ZIPs:", zip_los_counts["Zip Code - 3 digits"].unique()[:10])
```

```
ZCTA ZIPs: ['122' '120' '148' '145' '104' '137' '147' '131' '130' '140']
LOS ZIPs: ['100' '100.0' '101' '101.0' '103' '103.0' '104' '104.0' '105'
'105.0']
```

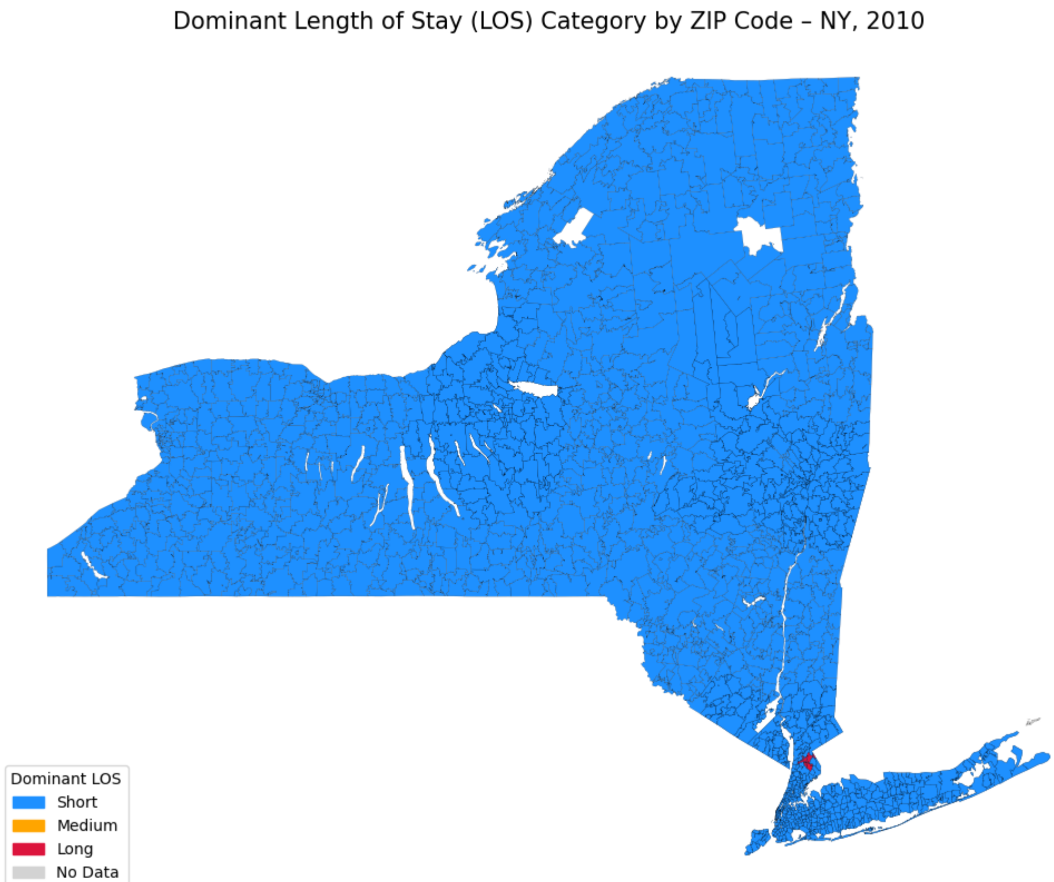
[188]:
```
zcta_gdf["ZIP"] = (
    zcta_gdf["ZCTA5CE10"]
    .astype(str)
    .str[:3]
    .str.zfill(3)
)

print(" All ZIP prefixes in shapefile:", sorted(zcta_gdf["ZIP"].unique()))
```

```
 All ZIP prefixes in shapefile: ['063', '100', '101', '102', '103', '104',
'105', '106', '107', '108', '109', '110', '111', '112', '113', '114', '115',
'116', '117', '118', '119', '120', '121', '122', '123', '124', '125', '126',
'127', '128', '129', '130', '131', '132', '133', '134', '135', '136', '137',
'138', '139', '140', '141', '142', '143', '144', '145', '146', '147', '148',
'149']
```

[187]:
```
zip_los_counts["Zip Code - 3 digits"] = zip_los_counts["Zip Code - 3 digits"].
↪astype(str).str.extract(r"(\d{3})")

merged_gdf = zcta_gdf.merge(zip_los_counts, left_on="ZIP", right_on="Zip Code -␣
↪3 digits", how="left")

print(" Matching ZIPs:", merged_gdf["ZIP"].isin(zip_los_counts["Zip Code - 3␣
↪digits"]).sum())
print(" Dominant LOS counts:\n", merged_gdf["Dominant_LOS"].
↪value_counts(dropna=False))
```

```
 Matching ZIPs: 2568
 Dominant LOS counts:
 Dominant_LOS
Short     2562
Long         6
NaN          6
Medium       0
Name: count, dtype: int64
```

[183]:
```
import matplotlib.patches as mpatches
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(14, 10))
merged_gdf.plot(color=merged_gdf["color"], ax=ax, edgecolor="black",␣
↪linewidth=0.1)

plt.title("Dominant Length of Stay (LOS) Category by ZIP Code - NY, 2010",␣
↪fontsize=15)
plt.axis("off")

legend_patches = [
    mpatches.Patch(color="dodgerblue", label="Short"),
    mpatches.Patch(color="orange", label="Medium"),
    mpatches.Patch(color="crimson", label="Long"),
    mpatches.Patch(color="lightgray", label="No Data"),
]
plt.legend(handles=legend_patches, title="Dominant LOS", loc="lower left")

plt.show()
```



2.0.8 Geospatial Mapping of Dominant Length of Stay (LOS) by ZIP Code – NY, 2010

This choropleth map visualizes the **dominant Length of Stay (LOS) category** (Short, Medium, Long) for each ZIP Code Tabulation Area (ZCTA) across New York State, based on 2010 hospitalization data.

Process Summary:

1. **Shapefile Integration**:
   - A 2010 TIGER/Line shapefile (`tl_2010_36_zcta510.shp`) from the U.S. Census was loaded using `GeoPandas`.
   - ZIP codes were truncated to 3-digit format (`ZCTA5CE10 → ZIP`) to match the hospital dataset format.
2. **LOS Aggregation by ZIP**:
   - The cleaned hospital dataset was grouped by 3-digit ZIP codes and LOS categories (Short, Medium, Long).

- The **dominant LOS category** per ZIP was calculated by finding the most frequent class.

3. **Data Merge**:
   - The LOS summary was merged into the shapefile GeoDataFrame using the ZIP code as the common key.
   - Custom coloring and category control ensured clean mapping without visualization errors.
4. **Plotting**:
   - The final map highlights ZIP-level patterns in dominant LOS category using a custom color scheme:
     - **Dodger Blue** → Short Stay
     - **Orange** → Medium Stay
     - **Crimson** → Long Stay
     - **Light Gray** → No data

Interpretation:

- **Short Stays Dominate Statewide**:
  - The overwhelming majority of ZIPs across New York are shaded in blue, indicating that **Short LOS** is the most common category in nearly every region.
  - This aligns with previous findings from classification models and bar charts.
- **Localized Long Stay Clusters**:
  - A small number of ZIPs (shown in red) near **urban centers such as New York City** exhibit Long LOS as the dominant category.
  - These areas may represent regions with higher clinical complexity, lower discharge capacity, or socioeconomic constraints that delay transitions of care.
- **No Medium-Dominant ZIPs**:
  - Interestingly, no ZIP codes displayed Medium LOS as the dominant category — reinforcing earlier classification results where Medium LOS was the **least predictable and most frequently misclassified**.

Implications: This geospatial view confirms the **regional consistency** in hospitalization duration while also highlighting localized disparities. When combined with logistic regression and ZIP-level socioeconomic research, these maps help to:

- Prioritize ZIPs for **resource allocation** or discharge planning interventions.
- Understand how **urban vs. rural patterns** may influence healthcare utilization.
- Guide future analysis into **drivers of Long LOS**, especially in high-density areas.

2.1 Final Summary

This project set out to investigate disparities in hospital Length of Stay (LOS) across New York using predictive modeling, statistical testing, and geospatial analysis. Through the use of machine learning algorithms such as Logistic Regression and Random Forest, I identified key drivers of LOS, notably Total Charges, APR Risk of Mortality, and select demographic and regional features.

Incorporating Race, Health Service Area, and ZIP Code allowed for a more nuanced view of healthcare patterns, revealing that Short stays were the most predictable, while Medium and Long stays showed greater variability, especially across racial groups and geographic regions. A Chi-Square test confirmed a statistically significant relationship between Race and LOS, reinforcing the project's objective of identifying potential healthcare disparities.

The geospatial analysis, powered by 2010 Census shapefiles, helped visualize ZIP-level dominance of LOS categories, highlighting clusters of longer stays and providing context through supplemental demographic research (e.g., poverty rates, racial composition, access disparities).

Overall, the findings not only met my analytical goals for the project and supported actionable insights for healthcare resource planning and equity monitoring, but also demonstrated how much more I can build upon. There are still many opportunities to expand my feature angles, dive deeper into subgroup analyses, and uncover additional layers of insight. This project lays the groundwork for a growing and iterative approach to healthcare analytics.