In [2]:
```python
import pandas as pd

# Load dataset
df = pd.read_csv("data/diabetic_data.csv")

# Quick look
print("Shape:", df.shape)
df.head()
```

```
Shape: (101766, 50)
```

Out[2]:

	encounter_id	patient_nbr	race	gender	age	weight	admission_type_id	dis
0	2278392	8222157	Caucasian	Female	[0-10)	?	6	
1	149190	55629189	Caucasian	Female	[10-20)	?	1	
2	64410	86047875	AfricanAmerican	Female	[20-30)	?	1	
3	500364	82442376	Caucasian	Male	[30-40)	?	1	
4	16680	42519267	Caucasian	Male	[40-50)	?	1	

5 rows × 50 columns

In [3]:
```python
# See column names, data types, and non-null counts
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 101766 entries, 0 to 101765
Data columns (total 50 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   encounter_id              101766 non-null  int64
 1   patient_nbr               101766 non-null  int64
 2   race                      101766 non-null  object
 3   gender                    101766 non-null  object
 4   age                       101766 non-null  object
 5   weight                    101766 non-null  object
 6   admission_type_id         101766 non-null  int64
 7   discharge_disposition_id  101766 non-null  int64
 8   admission_source_id       101766 non-null  int64
 9   time_in_hospital          101766 non-null  int64
 10  payer_code                101766 non-null  object
 11  medical_specialty         101766 non-null  object
 12  num_lab_procedures        101766 non-null  int64
 13  num_procedures            101766 non-null  int64
 14  num_medications           101766 non-null  int64
 15  number_outpatient         101766 non-null  int64
 16  number_emergency          101766 non-null  int64
 17  number_inpatient          101766 non-null  int64
 18  diag_1                    101766 non-null  object
 19  diag_2                    101766 non-null  object
 20  diag_3                    101766 non-null  object
 21  number_diagnoses          101766 non-null  int64
 22  max_glu_serum             5346 non-null    object
 23  A1Cresult                 17018 non-null   object
 24  metformin                 101766 non-null  object
 25  repaglinide               101766 non-null  object
 26  nateglinide               101766 non-null  object
 27  chlorpropamide            101766 non-null  object
 28  glimepiride               101766 non-null  object
 29  acetohexamide             101766 non-null  object
 30  glipizide                 101766 non-null  object
 31  glyburide                 101766 non-null  object
 32  tolbutamide               101766 non-null  object
 33  pioglitazone              101766 non-null  object
 34  rosiglitazone             101766 non-null  object
 35  acarbose                  101766 non-null  object
 36  miglitol                  101766 non-null  object
 37  troglitazone              101766 non-null  object
 38  tolazamide                101766 non-null  object
 39  examide                   101766 non-null  object
 40  citoglipton               101766 non-null  object
 41  insulin                   101766 non-null  object
 42  glyburide-metformin       101766 non-null  object
 43  glipizide-metformin       101766 non-null  object
 44  glimepiride-pioglitazone  101766 non-null  object
 45  metformin-rosiglitazone   101766 non-null  object
 46  metformin-pioglitazone    101766 non-null  object
 47  change                    101766 non-null  object
 48  diabetesMed               101766 non-null  object
 49  readmitted                101766 non-null  object
```

```
dtypes: int64(13), object(37)
memory usage: 38.8+ MB
```

In [4]:
```python
# Statistics for numeric columns
df.describe()
```

Out[4]:

encounter_id	patient_nbr	admission_type_id	discharge_disposition_id	admission_source_i
1.017660e+05	1.017660e+05	101766.000000	101766.000000	101766.00000
1.652016e+08	5.433040e+07	2.024006	3.715642	5.75443
1.026403e+08	3.869636e+07	1.445403	5.280166	4.06408
1.252200e+04	1.350000e+02	1.000000	1.000000	1.00000
8.496119e+07	2.341322e+07	1.000000	1.000000	1.00000
1.523890e+08	4.550514e+07	1.000000	1.000000	7.00000
2.302709e+08	8.754595e+07	3.000000	4.000000	7.00000
4.438672e+08	1.895026e+08	8.000000	28.000000	25.00000

In [5]:
```python
# Count of missing values per column
df.isnull().sum()
```

Out[5]:

```
encounter_id                    0
patient_nbr                     0
race                            0
gender                          0
age                             0
weight                          0
admission_type_id               0
discharge_disposition_id        0
admission_source_id             0
time_in_hospital                0
payer_code                      0
medical_specialty               0
num_lab_procedures              0
num_procedures                  0
num_medications                 0
number_outpatient               0
number_emergency                0
number_inpatient                0
diag_1                          0
diag_2                          0
diag_3                          0
number_diagnoses                0
max_glu_serum               96420
A1Cresult                   84748
metformin                       0
repaglinide                     0
nateglinide                     0
chlorpropamide                  0
glimepiride                     0
acetohexamide                   0
glipizide                       0
glyburide                       0
tolbutamide                     0
pioglitazone                    0
rosiglitazone                   0
acarbose                        0
miglitol                        0
troglitazone                    0
tolazamide                      0
examide                         0
citoglipton                     0
insulin                         0
glyburide-metformin             0
glipizide-metformin             0
glimepiride-pioglitazone        0
metformin-rosiglitazone         0
metformin-pioglitazone          0
change                          0
diabetesMed                     0
readmitted                      0
dtype: int64
```

In [6]:

```python
# Check categorical columns
categorical_cols = df.select_dtypes(include='object').columns
print("Categorical columns:", categorical_cols)
```

```python
# Look at unique values in a few important columns
for col in ['race', 'gender', 'age', 'admission_type_id']:
    print(f"\n{col} unique values:", df[col].unique())
```

```
Categorical columns: Index(['race', 'gender', 'age', 'weight', 'payer_code', 'medica
l_specialty',
       'diag_1', 'diag_2', 'diag_3', 'max_glu_serum', 'A1Cresult', 'metformin',
       'repaglinide', 'nateglinide', 'chlorpropamide', 'glimepiride',
       'acetohexamide', 'glipizide', 'glyburide', 'tolbutamide',
       'pioglitazone', 'rosiglitazone', 'acarbose', 'miglitol', 'troglitazone',
       'tolazamide', 'examide', 'citoglipton', 'insulin',
       'glyburide-metformin', 'glipizide-metformin',
       'glimepiride-pioglitazone', 'metformin-rosiglitazone',
       'metformin-pioglitazone', 'change', 'diabetesMed', 'readmitted'],
      dtype='object')

race unique values: ['Caucasian' 'AfricanAmerican' '?' 'Other' 'Asian' 'Hispanic']

gender unique values: ['Female' 'Male' 'Unknown/Invalid']

age unique values: ['[0-10)' '[10-20)' '[20-30)' '[30-40)' '[40-50)' '[50-60)' '[60-
70)'
 '[70-80)' '[80-90)' '[90-100)']

admission_type_id unique values: [6 1 2 3 4 5 8 7]
```

In [7]:

```python
# Check distribution of the target variable
df['readmitted'].value_counts()
```

Out[7]:

```
readmitted
NO     54864
>30    35545
<30    11357
Name: count, dtype: int64
```

In [8]:

```python
import numpy as np

# Replace '?' with NaN
df.replace('?', np.nan, inplace=True)

# Check missing values again
df.isnull().sum()
```

Out[8]:

```
encounter_id                    0
patient_nbr                     0
race                         2273
gender                          0
age                             0
weight                      98569
admission_type_id               0
discharge_disposition_id        0
admission_source_id             0
time_in_hospital                0
payer_code                  40256
medical_specialty           49949
num_lab_procedures              0
num_procedures                  0
num_medications                 0
number_outpatient               0
number_emergency                0
number_inpatient                0
diag_1                         21
diag_2                        358
diag_3                       1423
number_diagnoses                0
max_glu_serum               96420
A1Cresult                   84748
metformin                       0
repaglinide                     0
nateglinide                     0
chlorpropamide                  0
glimepiride                     0
acetohexamide                   0
glipizide                       0
glyburide                       0
tolbutamide                     0
pioglitazone                    0
rosiglitazone                   0
acarbose                        0
miglitol                        0
troglitazone                    0
tolazamide                      0
examide                         0
citoglipton                     0
insulin                         0
glyburide-metformin             0
glipizide-metformin             0
glimepiride-pioglitazone        0
metformin-rosiglitazone         0
metformin-pioglitazone          0
change                          0
diabetesMed                     0
readmitted                      0
dtype: int64
```

In [10]:

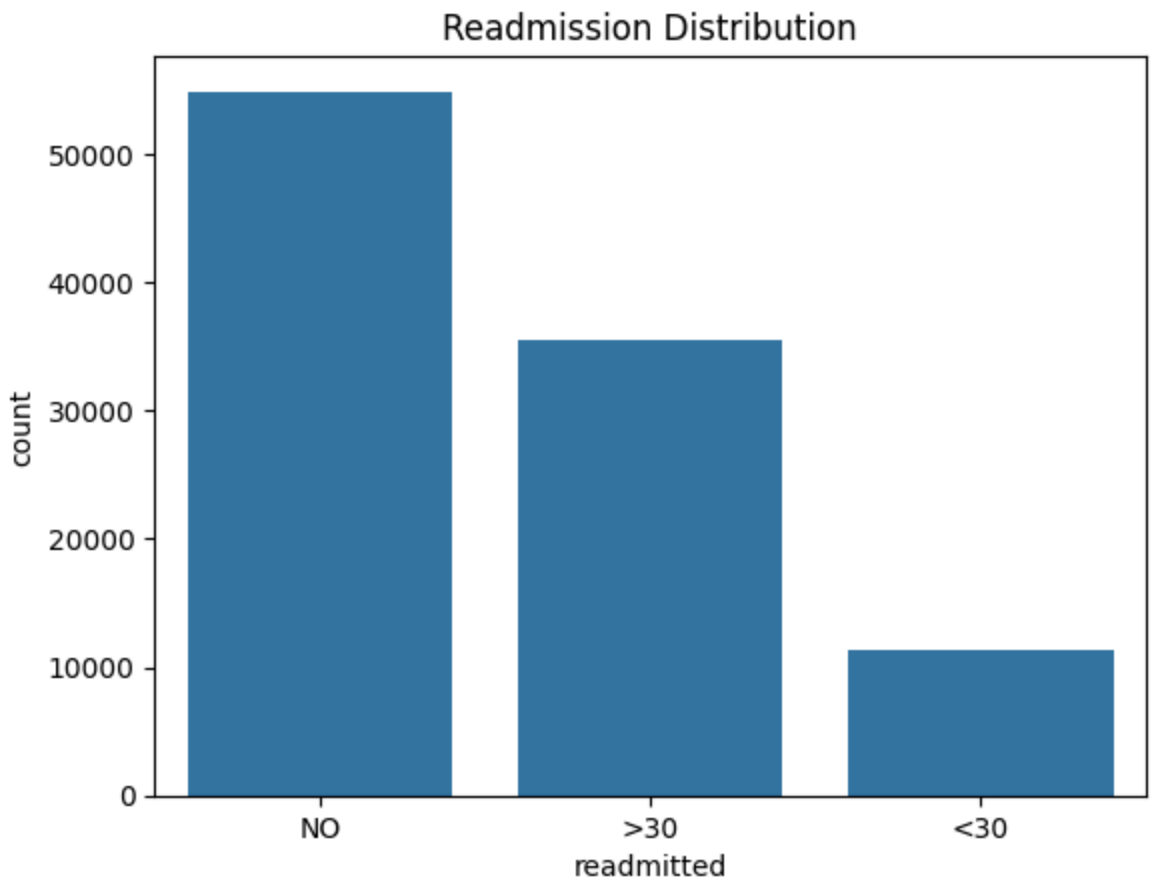
```python
# Fill missing 'race' with the most common race
df['race'] = df['race'].fillna(df['race'].mode()[0])
```

```
# Confirm
df['race'].isnull().sum()
```

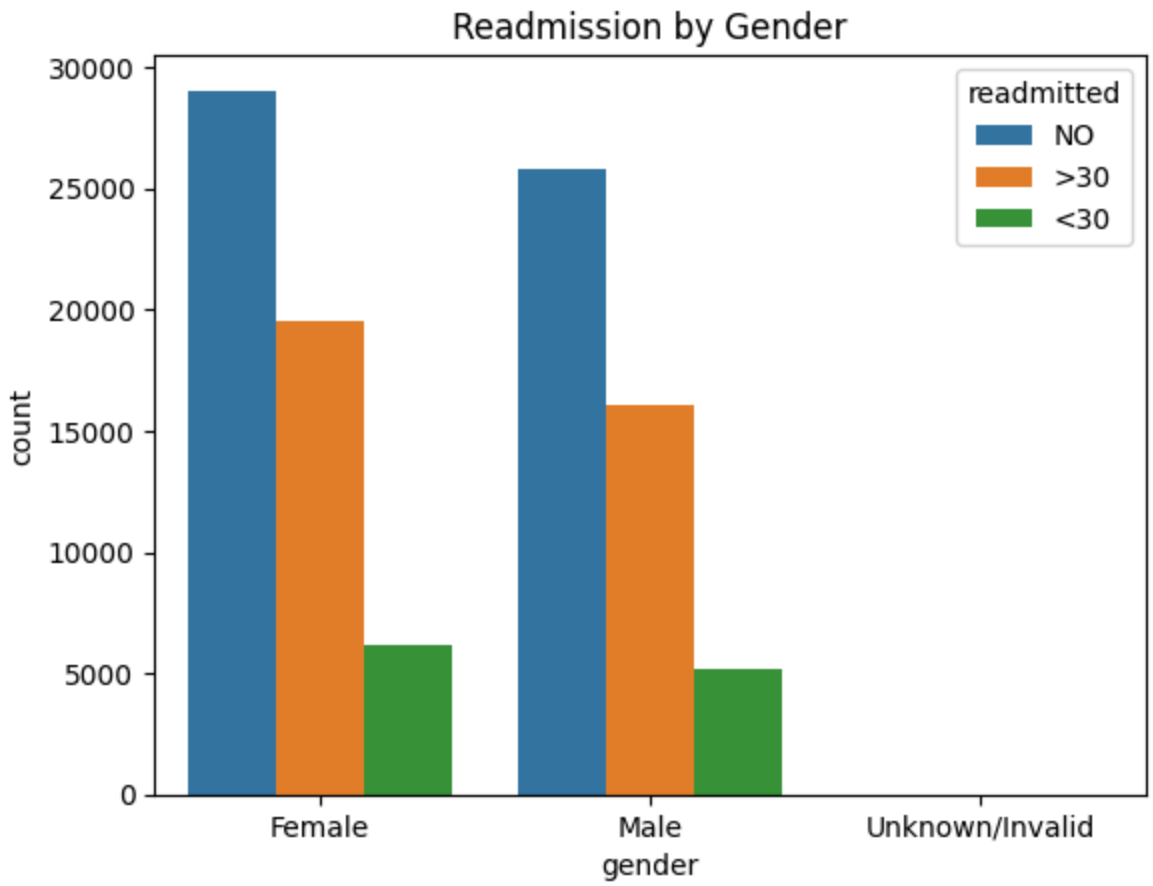
Out[10]: np.int64(0)

In [11]:
```
# Columns to drop
cols_to_drop = ['encounter_id', 'patient_nbr']  # unique identifiers
df.drop(columns=cols_to_drop, inplace=True)
```

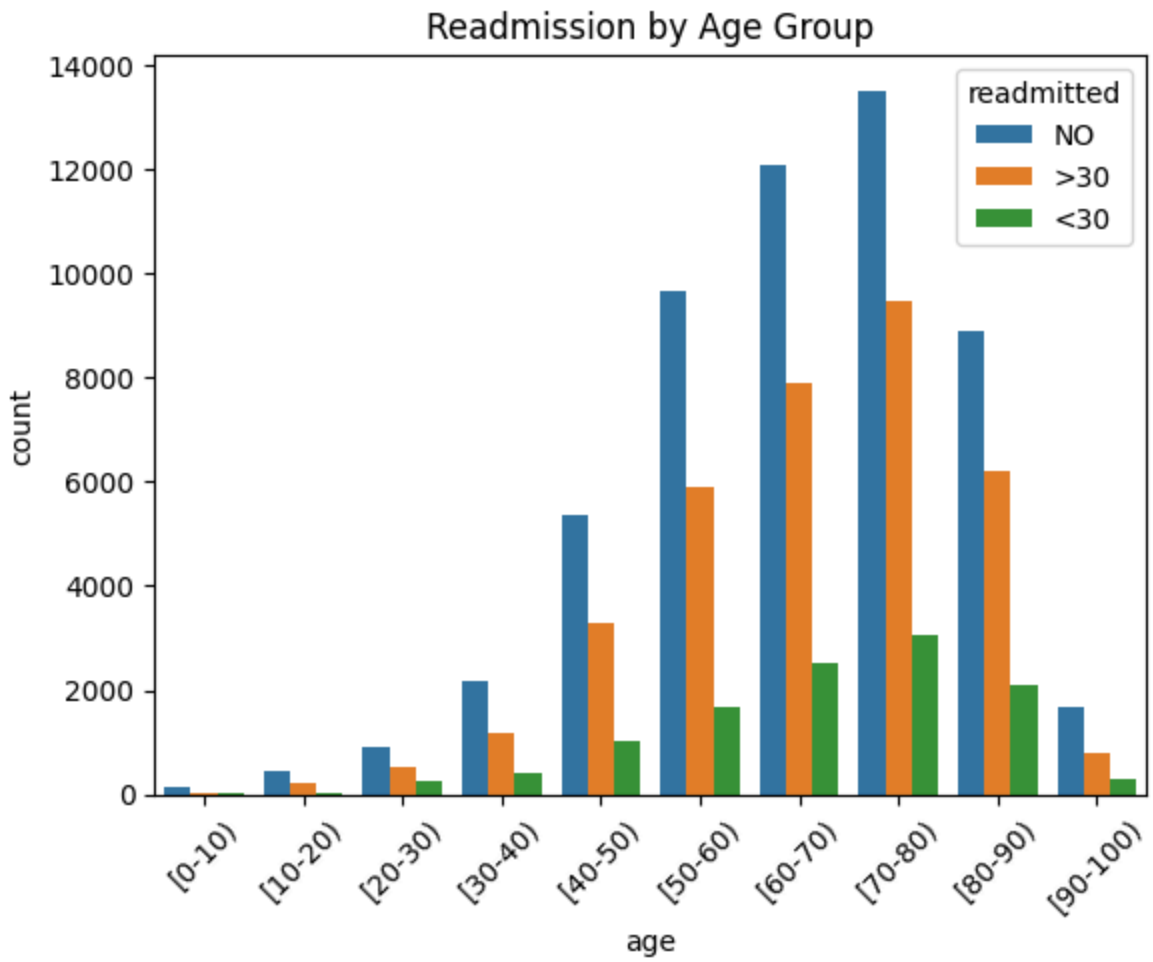
In [12]:
```
import matplotlib.pyplot as plt
import seaborn as sns

# Count of readmissions
sns.countplot(data=df, x='readmitted')
plt.title('Readmission Distribution')
plt.show()
```



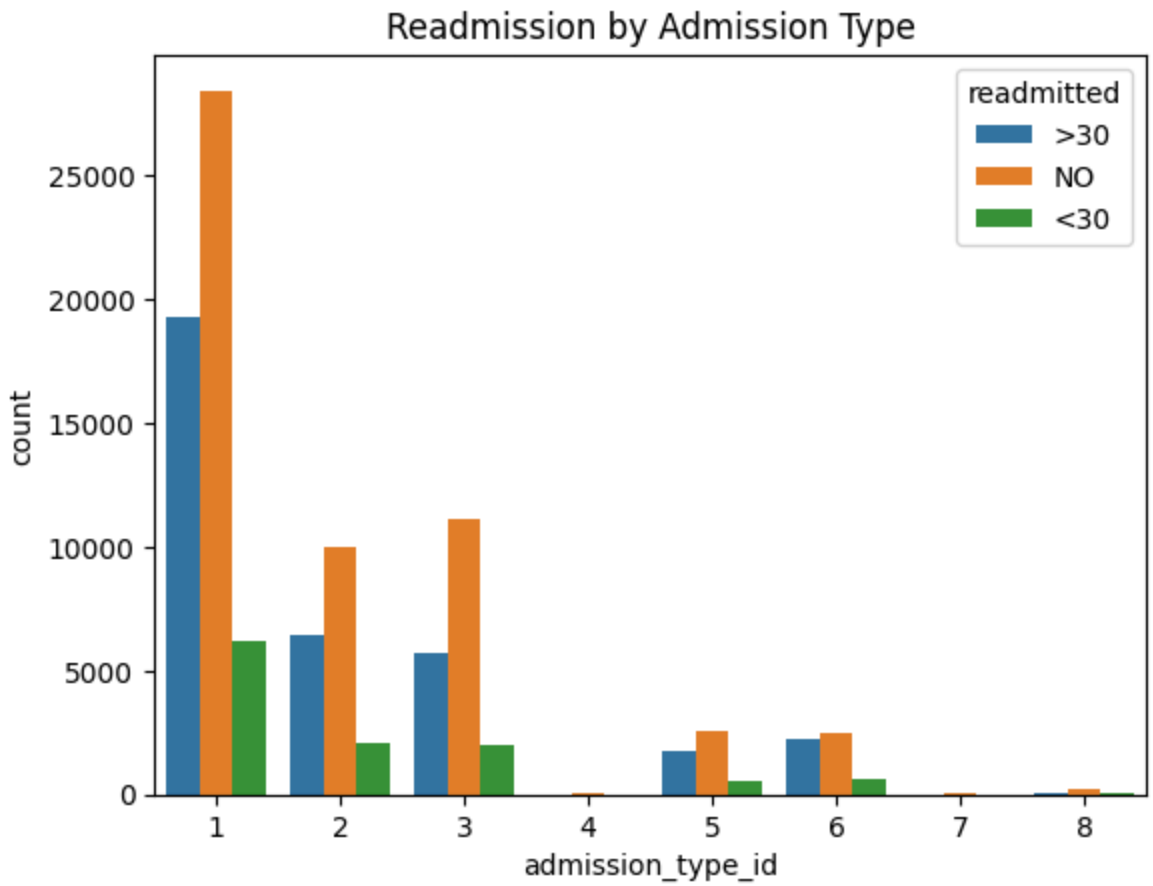
In [13]:
```
sns.countplot(data=df, x='gender', hue='readmitted')
plt.title('Readmission by Gender')
plt.show()
```



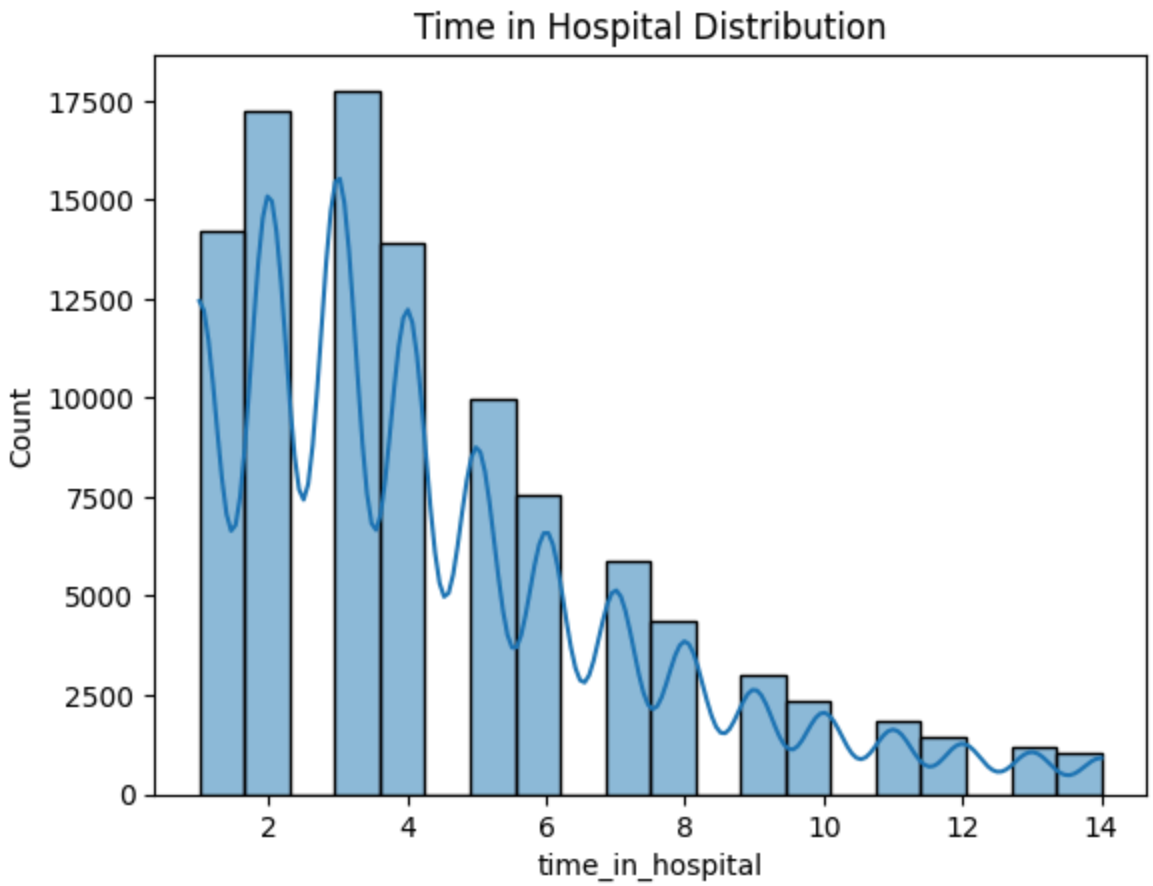
In [14]:
```
sns.countplot(data=df, x='age', hue='readmitted')
plt.title('Readmission by Age Group')
plt.xticks(rotation=45)
plt.show()
```

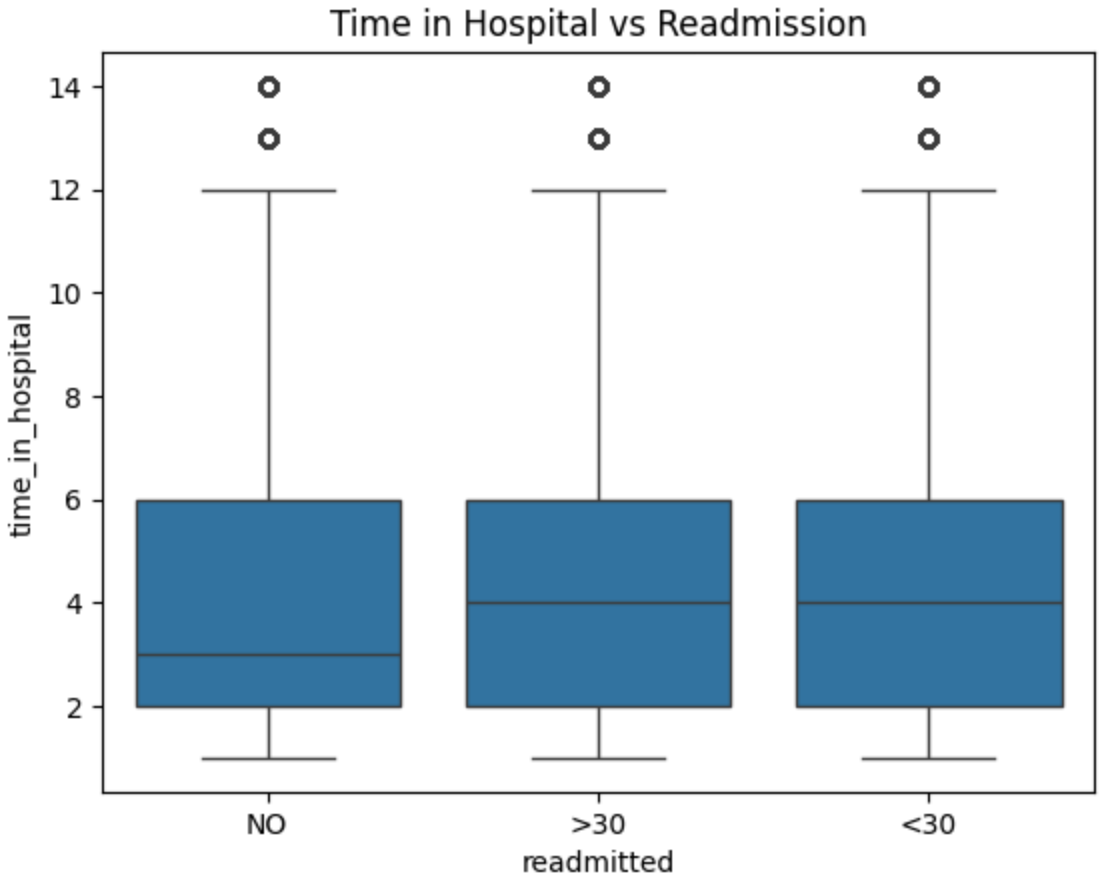


In [15]:
```
sns.countplot(data=df, x='admission_type_id', hue='readmitted')
plt.title('Readmission by Admission Type')
plt.show()
```

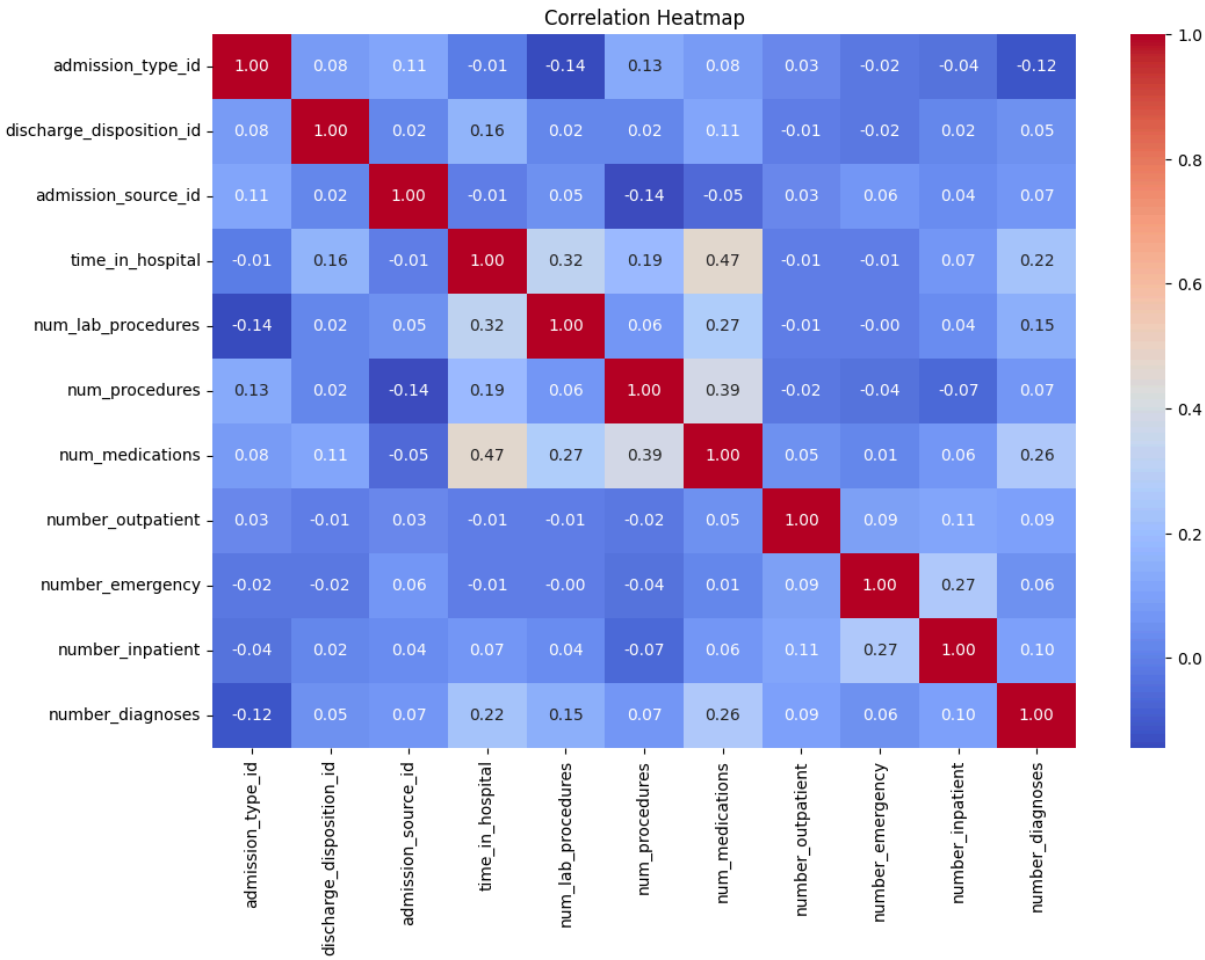


In [16]:
```
# Histogram for time in hospital
sns.histplot(data=df, x='time_in_hospital', bins=20, kde=True)
plt.title('Time in Hospital Distribution')
plt.show()

# Boxplot: time_in_hospital vs readmission
sns.boxplot(data=df, x='readmitted', y='time_in_hospital')
plt.title('Time in Hospital vs Readmission')
plt.show()
```


Time in Hospital Distribution
Count
17500
15000
12500
10000
7500
5000
2500
0
2
4
6
8
10
12
14
time_in_hospital


Time in Hospital vs Readmission
time_in_hospital
14
12
10
8
6
4
2
NO
>30
<30
readmitted

In [17]:
```python
# Select numeric columns
numeric_cols = df.select_dtypes(include=['int64', 'float64']).columns

# Correlation matrix
corr = df[numeric_cols].corr()

# Heatmap
plt.figure(figsize=(12,8))
sns.heatmap(corr, annot=True, fmt=".2f", cmap="coolwarm")
plt.title("Correlation Heatmap")
plt.show()
```



In [25]:
```python
# Target
y = df['readmitted']

# Features (drop target and any other irrelevant columns)
X = df.drop(columns=['readmitted'])
```

In [24]:
```python
from sklearn.preprocessing import LabelEncoder

# Identify categorical columns
categorical_cols = X.select_dtypes(include='object').columns

# Apply LabelEncoder to each categorical column
le = LabelEncoder()
```

```
for col in categorical_cols:
    X[col] = le.fit_transform(X[col].astype(str))
```

In [23]:
```
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
)
```

In [35]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Initialize model
model = LogisticRegression(max_iter=1000)

# Train
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)

# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
```

```
C:\Users\tkavali\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\l
inear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to converge after 100
0 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=1000).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Accuracy: 0.5712881988798271

Classification Report:
               precision    recall  f1-score   support

         <30       0.34      0.01      0.01      2285
         >30       0.50      0.26      0.34      7117
          NO       0.59      0.89      0.71     10952

    accuracy                           0.57     20354
   macro avg       0.48      0.39      0.35     20354
weighted avg       0.53      0.57      0.50     20354
```

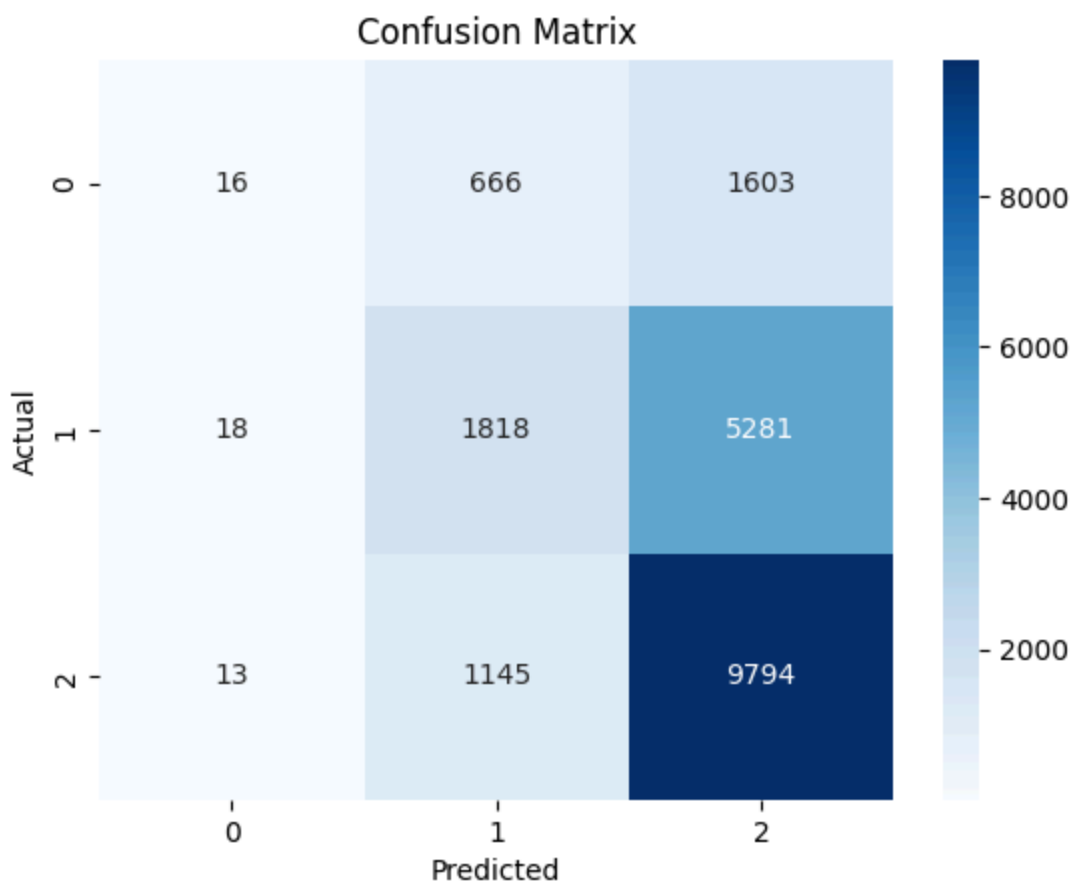
In [28]:
```
from sklearn.preprocessing import StandardScaler

# Identify numeric columns
numeric_cols = X.select_dtypes(include=['int64', 'float64']).columns
```

```
scaler = StandardScaler()
X[numeric_cols] = scaler.fit_transform(X[numeric_cols])
```

In [31]:
```
model = LogisticRegression(solver='saga', max_iter=5000)
model.fit(X_train, y_train)
```

Out[31]:
```
▾ LogisticRegression
▶ Parameters
```

In [33]:
```
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```



In [34]:
```
model = LogisticRegression(max_iter=5000)  # increase from 1000 to 5000
model.fit(X_train, y_train)
```

```
C:\Users\tkavali\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\l
inear_model\_logistic.py:473: ConvergenceWarning: lbfgs failed to converge after 500
0 iteration(s) (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT

Increase the number of iterations to improve the convergence (max_iter=5000).
You might also want to scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[34]:

```
▾ LogisticRegression

▶ Parameters
```

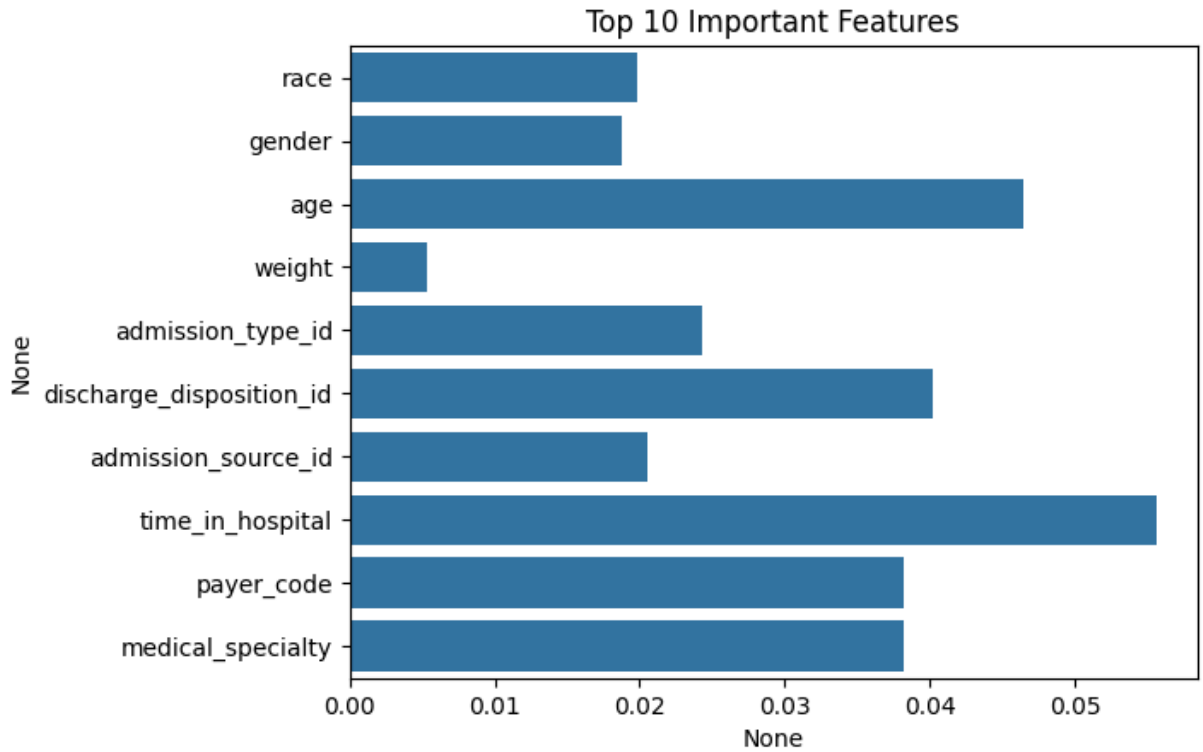
In [36]:

```python
from sklearn.ensemble import RandomForestClassifier

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
y_pred_rf = rf_model.predict(X_test)

from sklearn.metrics import accuracy_score, classification_report
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred_rf))
print("\nClassification Report:\n", classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.5846025351282303

Classification Report:
               precision    recall  f1-score   support

         <30       0.43      0.01      0.03      2285
         >30       0.50      0.39      0.44      7117
          NO       0.62      0.83      0.71     10952

    accuracy                           0.58     20354
   macro avg       0.52      0.41      0.39     20354
weighted avg       0.56      0.58      0.54     20354
```

In [37]:

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

feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns)
feature_importances.sort_values(ascending=False).head(10)  # top 10 features

# Plot top 10 features
sns.barplot(x=feature_importances.head(10), y=feature_importances.head(10).index)
plt.title("Top 10 Important Features")
plt.show()
```



In [39]:
```
import pickle

# Save model
with open("rf_model.pkl", "wb") as f:
    pickle.dump(rf_model, f)

# Load model later
# with open("rf_model.pkl", "rb") as f:
#     rf_model = pickle.load(f)
```

In [ ]: