In [1]:
```
# Libraries
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from datetime import datetime
import statsmodels.api as sm
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from scipy.stats import f_oneway
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from xgboost import XGBRegressor
from sklearn.impute import SimpleImputer
```

Loading Data (RR 5DR) FOCUS ON DELIVERIES

In [2]:
```
df = pd.read_csv("OB DELIVERY.csv", parse_dates = ["HospitalAdmissionTime",
                                                   "HospitalDischargeTime",
                                                   "DeliveryDate",
                                                   "BirthDateTime",
                                                   "LaborOnsetDateTime"])
#df = pd.read_csv("ob_extended.csv")
df.head()
```

Out[2]:

	PatientEncounterCSNID	EpisodeID	CheckinTime	CheckoutTime	HospitalAdmissionTime	HospitalDischargeTime	LengthOfStay	IsInpatient	IsObservation	BedID	...	IsPrimaryDeliveryUnit	TotalLaborMinutes	GravidaCount	ParaCount	PretermCount	IsWeekend	IsHoliday	MonthName	QuarterNumber	Year
0	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1848.0	...	1.0	NaN	NaN	NaN	NaN	0.0	0.0	June	2.0	2022.0
1	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1856.0	...	1.0	NaN	NaN	NaN	NaN	0.0	0.0	June	2.0	2022.0
2	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1849.0	...	1.0	NaN	NaN	NaN	NaN	0.0	0.0	June	2.0	2022.0
3	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1855.0	...	1.0	NaN	NaN	NaN	NaN	0.0	0.0	June	2.0	2022.0
4	2103031973	5.542547e+18	NaN	NaN	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	4455.0	...	1.0	NaN	NaN	NaN	NaN	0.0	0.0	June	2.0	2022.0

5 rows × 36 columns

In [16]:
```
df.shape
```

Out[16]:
```
(80956, 36)
```

In [17]:
```
df.columns
```

Out[17]:
```
Index(['PatientEncounterCSNID', 'EpisodeID', 'CheckinTime', 'CheckoutTime',
       'HospitalAdmissionTime', 'HospitalDischargeTime', 'LengthOfStay',
       'IsInpatient', 'IsObservation', 'BedID', 'BedName', 'BedInCensus',
       'RoomID', 'RoomName', 'RoomGroupName', 'DeliveryDate', 'BirthDateTime',
       'LaborOnsetDateTime', 'DeliveryMethodCode', 'FirstStageLengthHours',
       'SecondStageLengthHours', 'ThirdStageLengthHours', 'NumberOfBabies',
       'EstimatedDateOfDelivery', 'DeliveryDepartmentID', 'DepartmentName',
       'IsPrimaryDeliveryUnit', 'TotalLaborMinutes', 'GravidaCount',
       'ParaCount', 'PretermCount', 'IsWeekend', 'IsHoliday', 'MonthName',
       'QuarterNumber', 'Year'],
      dtype='object')
```

EDA

In [18]:
```
# info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 80956 entries, 0 to 80955
Data columns (total 36 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   PatientEncounterCSNID    80956 non-null  int64
 1   EpisodeID                80954 non-null  float64
 2   CheckinTime              0 non-null      float64
 3   CheckoutTime             0 non-null      float64
 4   HospitalAdmissionTime    80955 non-null  datetime64[ns]
 5   HospitalDischargeTime    80683 non-null  datetime64[ns]
 6   LengthOfStay             80645 non-null  float64
 7   IsInpatient              80955 non-null  float64
 8   IsObservation            80955 non-null  float64
 9   BedID                    43325 non-null  float64
 10  BedName                  43325 non-null  object
 11  BedInCensus              43325 non-null  float64
 12  RoomID                   66724 non-null  float64
 13  RoomName                 71675 non-null  object
 14  RoomGroupName            6946 non-null   object
 15  DeliveryDate             80955 non-null  datetime64[ns]
 16  BirthDateTime            77606 non-null  datetime64[ns]
 17  LaborOnsetDateTime       47422 non-null  datetime64[ns]
 18  DeliveryMethodCode       77346 non-null  float64
 19  FirstStageLengthHours    38297 non-null  float64
 20  SecondStageLengthHours   50767 non-null  float64
 21  ThirdStageLengthHours    78475 non-null  float64
 22  NumberOfBabies           3347 non-null   float64
 23  EstimatedDateOfDelivery  3341 non-null   object
 24  DeliveryDepartmentID     77599 non-null  float64
 25  DepartmentName           80955 non-null  object
 26  IsPrimaryDeliveryUnit    80955 non-null  float64
 27  TotalLaborMinutes        46196 non-null  float64
 28  GravidaCount             3347 non-null   float64
 29  ParaCount                3347 non-null   float64
 30  PretermCount             3347 non-null   float64
 31  IsWeekend                80955 non-null  float64
 32  IsHoliday                80955 non-null  float64
 33  MonthName                80955 non-null  object
 34  QuarterNumber            80955 non-null  float64
 35  Year                     80955 non-null  float64
dtypes: datetime64[ns](5), float64(24), int64(1), object(6)
memory usage: 22.2+ MB
```

In [19]:
```
# summary statistics
df.describe()
```

Out[19]:

	PatientEncounterCSNID	EpisodeID	CheckinTime	CheckoutTime	LengthOfStay	IsInpatient	IsObservation	BedID	BedInCensus	RoomID	...	DeliveryDepartmentID	IsPrimaryDeliveryUnit	TotalLaborMinutes	GravidaCount	ParaCount	PretermCount	IsWeekend	IsHoliday	QuarterNumber	Year
count	8.095600e+04	8.095400e+04	0.0	0.0	80645.000000	80955.000000	80955.000000	43325.000000	43325.000000	66724.000000	...	77599.000000	80955.000000	46196.000000	3347.000000	3347.000000	3347.000000	80955.000000	80955.000000	80955.000000	80955.000000
mean	1.081218e+09	2.301473e+18	NaN	NaN	3.675498	0.999531	0.000469	2602.872937	0.775188	621.638960	...	10026.379129	0.914137	4142.093276	2.073499	0.600239	0.057664	0.278575	0.035822	2.329492	2022.786425
std	6.253885e+08	2.002151e+18	NaN	NaN	4.417778	0.021661	0.021661	1303.296745	0.417464	68.921009	...	38.246116	0.280163	26477.499120	1.335290	0.897816	0.307250	0.448300	0.185848	1.124790	0.714944
min	1.281280e+05	1.852020e+14	NaN	NaN	0.270139	0.000000	0.000000	1338.000000	0.000000	268.000000	...	10018.000000	0.000000	-524521.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	2022.000000
25%	5.400711e+08	6.242609e+17	NaN	NaN	2.245139	1.000000	0.000000	1847.000000	1.000000	630.000000	...	10018.000000	1.000000	939.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	2022.000000
50%	1.060613e+09	1.809687e+18	NaN	NaN	2.898611	1.000000	0.000000	1853.000000	1.000000	635.000000	...	10018.000000	1.000000	1454.000000	2.000000	0.000000	0.000000	0.000000	0.000000	2.000000	2023.000000
75%	1.641470e+09	3.408671e+18	NaN	NaN	3.861806	1.000000	0.000000	4454.000000	1.000000	638.000000	...	10018.000000	1.000000	2295.000000	3.000000	1.000000	0.000000	1.000000	0.000000	3.000000	2023.000000
max	2.146801e+09	9.073822e+18	NaN	NaN	208.839583	1.000000	1.000000	5601.000000	1.000000	1187.000000	...	10201.000000	1.000000	527585.000000	22.000000	10.000000	7.000000	1.000000	1.000000	4.000000	2024.000000

8 rows × 25 columns

In [3]:
```
# timeframe
df["DeliveryDate"].describe()
```

```
C:\Users\local_CAFu\Temp\16\ipykernel_18072\3594141849.py:2: FutureWarning: Treating datetime data as categorical rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Specify `datetime_is_numeric=True` to silence this warning and adopt the future behavior now.
  df["DeliveryDate"].describe()
```

Out[3]:
```
count                   80955
unique                    917
top       2022-02-13 00:00:00
freq                      421
first     2022-01-02 00:00:00
last      2024-08-01 00:00:00
Name: DeliveryDate, dtype: object
```

In [20]:
```
# missing data
# many columns with near or exactly null values close to 1 million
df.isnull().sum()
```

Out[20]:
```
PatientEncounterCSNID          0
EpisodeID                      2
CheckinTime                80956
CheckoutTime               80956
HospitalAdmissionTime          1
HospitalDischargeTime        273
LengthOfStay                 311
IsInpatient                    1
IsObservation                  1
BedID                      37631
BedName                    37631
BedInCensus                37631
RoomID                     14232
RoomName                    9281
RoomGroupName              74010
DeliveryDate                   1
BirthDateTime               3350
LaborOnsetDateTime         33534
DeliveryMethodCode          3610
FirstStageLengthHours      42659
SecondStageLengthHours     30189
ThirdStageLengthHours       2481
NumberOfBabies             77609
EstimatedDateOfDelivery    77615
DeliveryDepartmentID        3357
DepartmentName                 1
IsPrimaryDeliveryUnit          1
TotalLaborMinutes          34760
GravidaCount               77609
ParaCount                  77609
PretermCount               77609
IsWeekend                      1
IsHoliday                      1
MonthName                      1
QuarterNumber                  1
Year                           1
dtype: int64
```

Data Cleaning

In [57]:
```
# drop empty columns
cols = ["CheckinTime", "CheckoutTime"]

df = df.drop(cols, axis = 1)
```

In [58]:
```
# Handle missing values
imputer = SimpleImputer(strategy="median")  # Replace missing numerical values with the median
numerical_cols = df.select_dtypes(include=np.number).columns
df[numerical_cols] = imputer.fit_transform(df[numerical_cols])
```

In [59]:
```
# Ensure timestamps are in datetime format
df['HospitalAdmissionTime'] = pd.to_datetime(df['HospitalAdmissionTime'], errors='coerce')
df['HospitalDischargeTime'] = pd.to_datetime(df['HospitalDischargeTime'], errors='coerce')
```

In [61]:
```
print(np.isfinite(df.select_dtypes(include=np.number)).all())
```

```
PatientEncounterCSNID     True
EpisodeID                 True
LengthOfStay              True
IsInpatient               True
IsObservation             True
BedID                     True
BedInCensus               True
RoomID                    True
DeliveryMethodCode        True
FirstStageLengthHours     True
SecondStageLengthHours    True
ThirdStageLengthHours     True
NumberOfBabies            True
DeliveryDepartmentID      True
IsPrimaryDeliveryUnit     True
TotalLaborMinutes         True
GravidaCount              True
ParaCount                 True
PretermCount              True
IsWeekend                 True
IsHoliday                 True
QuarterNumber             True
Year                      True
dtype: bool
```

In [63]:
```
print(df.isna().sum())  # Check the number of missing values in each column
```

```
PatientEncounterCSNID          0
EpisodeID                      0
HospitalAdmissionTime          1
HospitalDischargeTime        273
LengthOfStay                   0
IsInpatient                    0
IsObservation                  0
BedID                          0
BedName                    37631
BedInCensus                    0
RoomID                         0
RoomName                    9281
RoomGroupName              74010
DeliveryDate                   1
BirthDateTime               3350
LaborOnsetDateTime         33534
DeliveryMethodCode             0
FirstStageLengthHours          0
SecondStageLengthHours         0
ThirdStageLengthHours          0
NumberOfBabies                 0
EstimatedDateOfDelivery    77615
DeliveryDepartmentID           0
DepartmentName                 1
IsPrimaryDeliveryUnit          0
TotalLaborMinutes              0
GravidaCount                   0
ParaCount                      0
PretermCount                   0
IsWeekend                      0
IsHoliday                      0
MonthName                      1
QuarterNumber                  0
Year                           0
dtype: int64
```

In [65]:
```
print(np.isinf(df.select_dtypes(include=np.number)).sum())  # Check for infinite values
```

```
PatientEncounterCSNID     0
EpisodeID                 0
LengthOfStay              0
IsInpatient               0
IsObservation             0
BedID                     0
BedInCensus               0
RoomID                    0
DeliveryMethodCode        0
FirstStageLengthHours     0
SecondStageLengthHours    0
ThirdStageLengthHours     0
NumberOfBabies            0
DeliveryDepartmentID      0
IsPrimaryDeliveryUnit     0
TotalLaborMinutes         0
GravidaCount              0
ParaCount                 0
PretermCount              0
IsWeekend                 0
IsHoliday                 0
QuarterNumber             0
Year                      0
dtype: int64
```

Fix NaN

In [67]:
```
df['HospitalDischargeTime'].fillna(df['HospitalAdmissionTime'], inplace=True)
```

In [69]:
```
df['HospitalDischargeTime'].fillna(df['HospitalDischargeTime'].median(), inplace=True)
```

In [70]:
```
df['HospitalAdmissionTime'].fillna(df['HospitalAdmissionTime'].median(), inplace=True)
```

In [71]:
```
print(df.isna().sum())  # Check the number of missing values in each column
```

```
PatientEncounterCSNID          0
EpisodeID                      0
HospitalAdmissionTime          0
HospitalDischargeTime          0
LengthOfStay                   0
IsInpatient                    0
IsObservation                  0
BedID                          0
BedName                    37631
BedInCensus                    0
RoomID                         0
RoomName                    9281
RoomGroupName              74010
DeliveryDate                   1
BirthDateTime               3350
LaborOnsetDateTime         33534
DeliveryMethodCode             0
FirstStageLengthHours          0
SecondStageLengthHours         0
ThirdStageLengthHours          0
NumberOfBabies                 0
EstimatedDateOfDelivery    77615
DeliveryDepartmentID           0
DepartmentName                 1
IsPrimaryDeliveryUnit          0
TotalLaborMinutes              0
GravidaCount                   0
ParaCount                      0
PretermCount                   0
IsWeekend                      0
IsHoliday                      0
MonthName                      1
QuarterNumber                  0
Year                           0
dtype: int64
```

In [72]:
```
df.head()
```

Out[72]:

	PatientEncounterCSNID	EpisodeID	HospitalAdmissionTime	HospitalDischargeTime	LengthOfStay	IsInpatient	IsObservation	BedID	BedName	BedInCensus	...	IsPrimaryDeliveryUnit	TotalLaborMinutes	GravidaCount	ParaCount	PretermCount	IsWeekend	IsHoliday	MonthName	QuarterNumber	Year
0	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1848.0	A	1.0	...	1.0	1454.0	2.0	0.0	0.0	0.0	0.0	June	2.0	2022.0
1	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1856.0	02	0.0	...	1.0	1454.0	2.0	0.0	0.0	0.0	0.0	June	2.0	2022.0
2	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1849.0	A	1.0	...	1.0	1454.0	2.0	0.0	0.0	0.0	0.0	June	2.0	2022.0
3	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1855.0	01	0.0	...	1.0	1454.0	2.0	0.0	0.0	0.0	0.0	June	2.0	2022.0
4	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	4455.0	05	0.0	...	1.0	1454.0	2.0	0.0	0.0	0.0	0.0	June	2.0	2022.0

5 rows × 34 columns

Feature Engineering

In [75]:
```
# Feature engineering
df['Week'] = df['HospitalAdmissionTime'].dt.isocalendar().week
df['Year'] = df['HospitalAdmissionTime'].dt.year
df['IsWeekendAdmission'] = df['HospitalAdmissionTime'].dt.weekday >= 5
```

In [76]:
```
# inspect revised data
df.head()
```

Out[76]:

	PatientEncounterCSNID	EpisodeID	HospitalAdmissionTime	HospitalDischargeTime	LengthOfStay	IsInpatient	IsObservation	BedID	BedName	BedInCensus	...	GravidaCount	ParaCount	PretermCount	IsWeekend	IsHoliday	MonthName	QuarterNumber	Year	Week	IsWeekendAdmission
0	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1848.0	A	1.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
1	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1856.0	02	0.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
2	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1849.0	A	1.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
3	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1855.0	01	0.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
4	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	4455.0	05	0.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False

5 rows × 36 columns

In [77]:
```
# new dimensions of revised data
df.shape
```

Out[77]:
```
(80956, 36)
```

In [78]:
```
# Aggregating weekly data
week_df = df.groupby(['Year', 'Week']).agg({
    'BedInCensus': 'sum',
    'LengthOfStay': 'mean',
    'IsWeekendAdmission': 'sum',
    'IsHoliday': 'sum',
    'TotalLaborMinutes': 'mean',
    'GravidaCount': 'mean',
}).reset_index()
```

In [79]:
```
# Bed Utilization Rate
week_df['BUR'] = week_df['BedInCensus'] / week_df['BedInCensus'].max()
```

In [80]:
```
week_df.head()
```

Out[80]:

	Year	Week	BedInCensus	LengthOfStay	IsWeekendAdmission	IsHoliday	TotalLaborMinutes	GravidaCount	BUR
0	2022	1	349.0	2.773313	45	0.0	1668.666667	1.987469	0.339164
1	2022	2	723.0	3.356665	279	41.0	2510.565217	2.013285	0.702624
2	2022	3	417.0	2.729994	159	0.0	1228.469602	1.983229	0.405248
3	2022	4	631.0	3.856938	81	0.0	2415.542302	2.002774	0.613217
4	2022	5	389.0	3.680463	159	0.0	1988.824324	2.020270	0.378037

In [81]:
```
week_df.shape
```

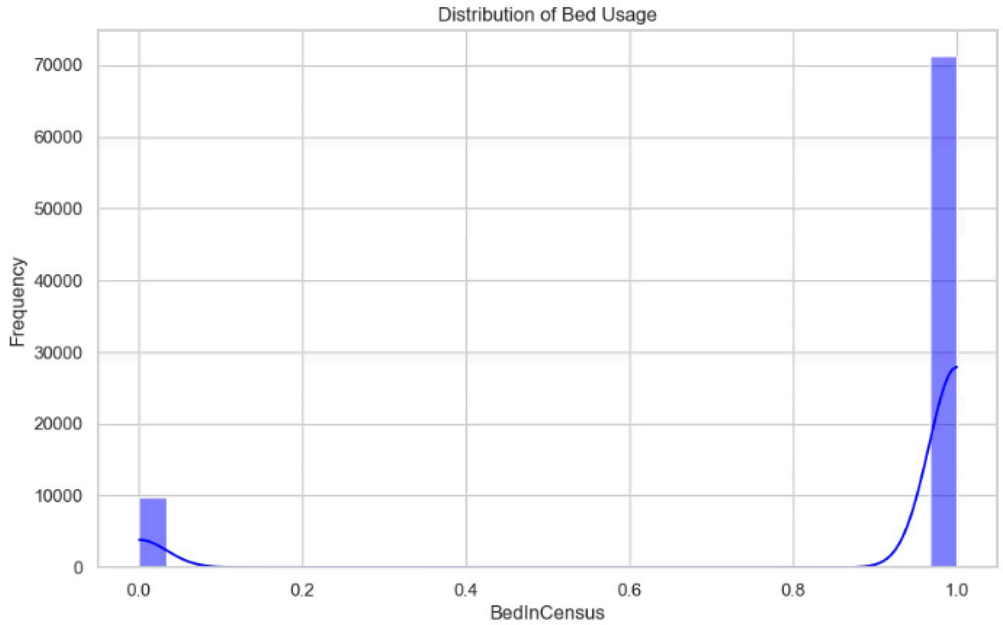
Out[81]:
```
(135, 9)
```

Data Visualizations

In [5]:
```
sns.set(style="whitegrid")
```

Distribution of Bed Usage

In [83]:
```
# 1. Distribution of Bed Usage
plt.figure(figsize=(10, 6))
sns.histplot(df['BedInCensus'], kde=True, bins=30, color='blue')
plt.title("Distribution of Bed Usage")
plt.xlabel("BedInCensus")
plt.ylabel("Frequency")
plt.show()
```



Weekly Bed Utilization Trends

In [7]:
```
weekly_trends = df.groupby(['Year', 'Week'])['BedInCensus'].sum().reset_index()

weekly_trends = weekly_trends.astype({
    'Week': 'int64',
    'BedInCensus': 'float64',
    'Year': 'int64'
})
```

In [9]:
```
weekly_trends
```

In [9]:
```
weekly_trends
```

Out[9]:

	Year	Week	BedInCensus
0	2022	1	349.0
1	2022	2	723.0
2	2022	3	417.0
3	2022	4	631.0
4	2022	5	389.0
...	...	...	...
130	2024	27	171.0
131	2024	28	271.0
132	2024	29	204.0
133	2024	30	102.0
134	2024	31	101.0

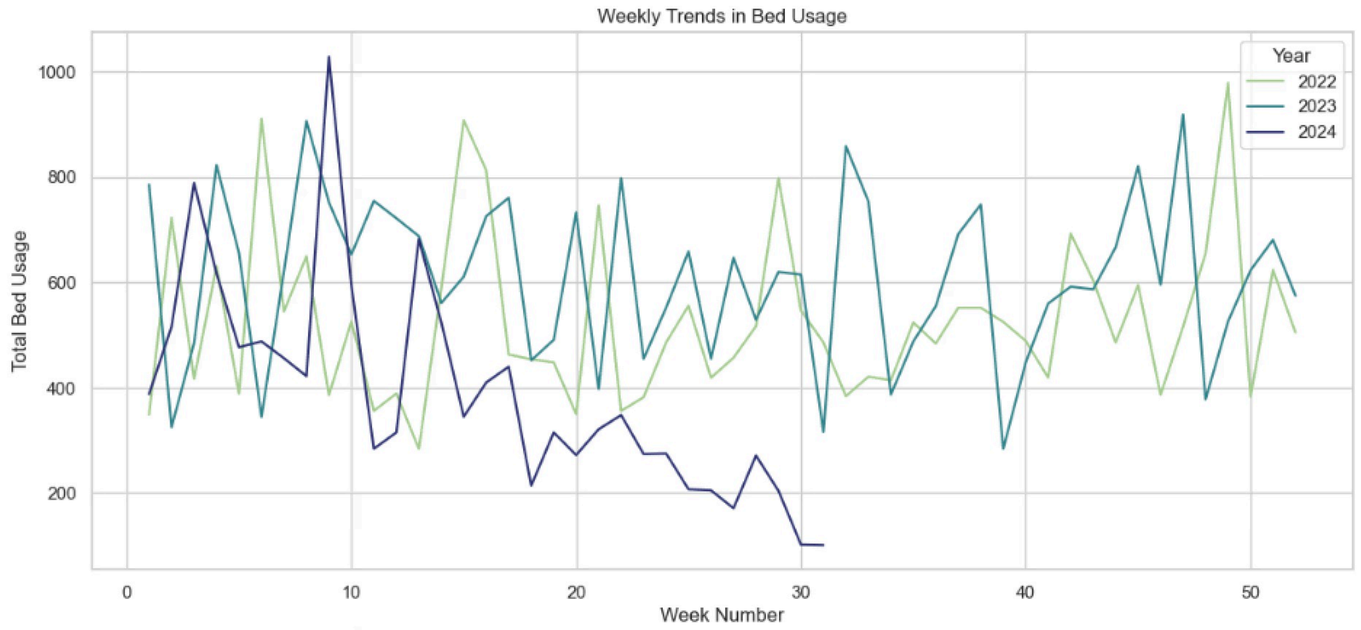
135 rows × 3 columns

In [8]:
```
weekly_trends.describe()
```

Out[8]:

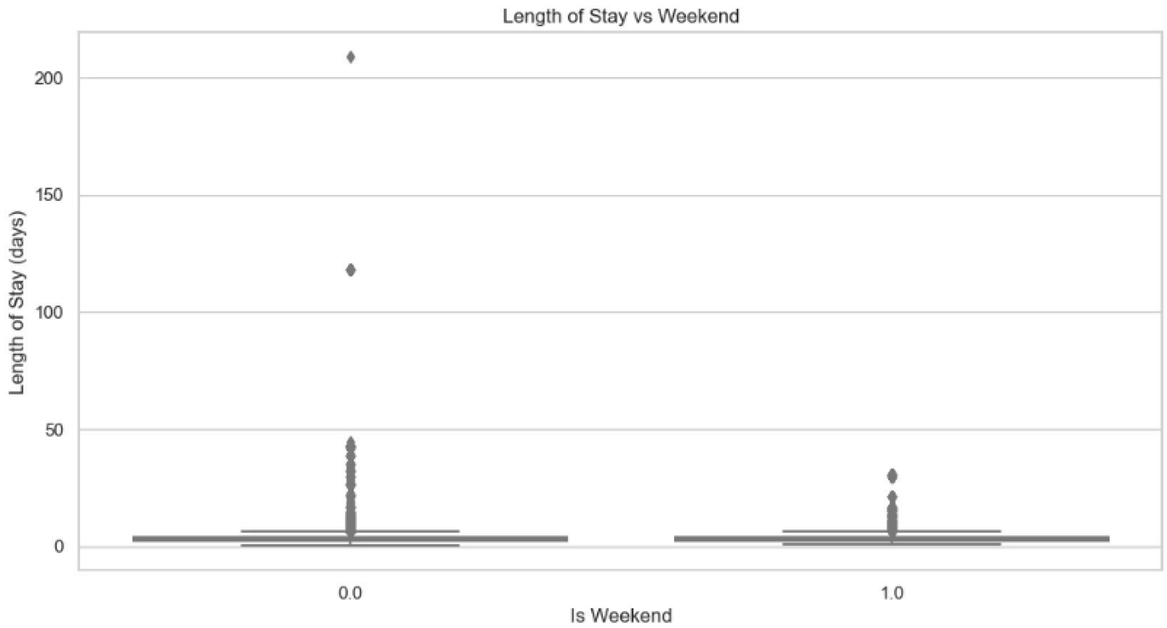
	Year	Week	BedInCensus
count	135.000000	135.000000	135.000000
mean	2022.844444	24.088889	527.525926
std	0.771379	14.593735	186.456983
min	2022.000000	1.000000	101.000000
25%	2022.000000	12.000000	388.500000
50%	2023.000000	23.000000	517.000000
75%	2023.000000	35.500000	651.500000
max	2024.000000	52.000000	1029.000000

In [19]:
```
# 2. Weekly Trends in Bed Usage
plt.figure(figsize=(14, 6))
sns.lineplot(data=weekly_trends,
             x='Week',
             y='BedInCensus',
             hue='Year',
             palette='crest')
plt.title("Weekly Trends in Bed Usage")
plt.xlabel("Week Number")
plt.ylabel("Total Bed Usage")
plt.show()
```

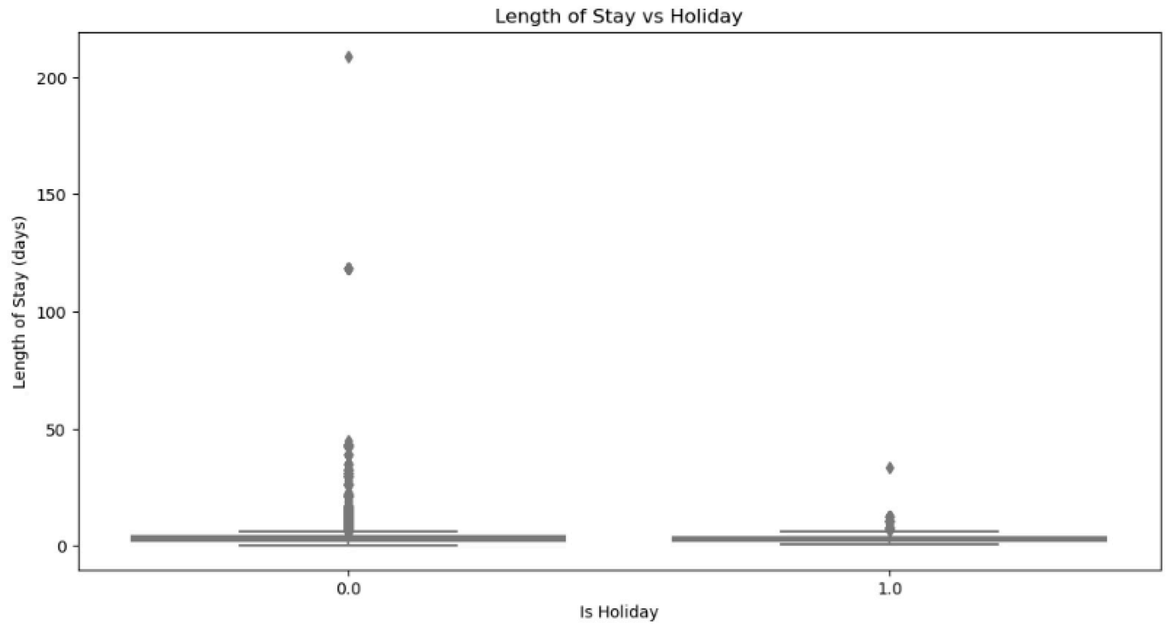


Temporal Features

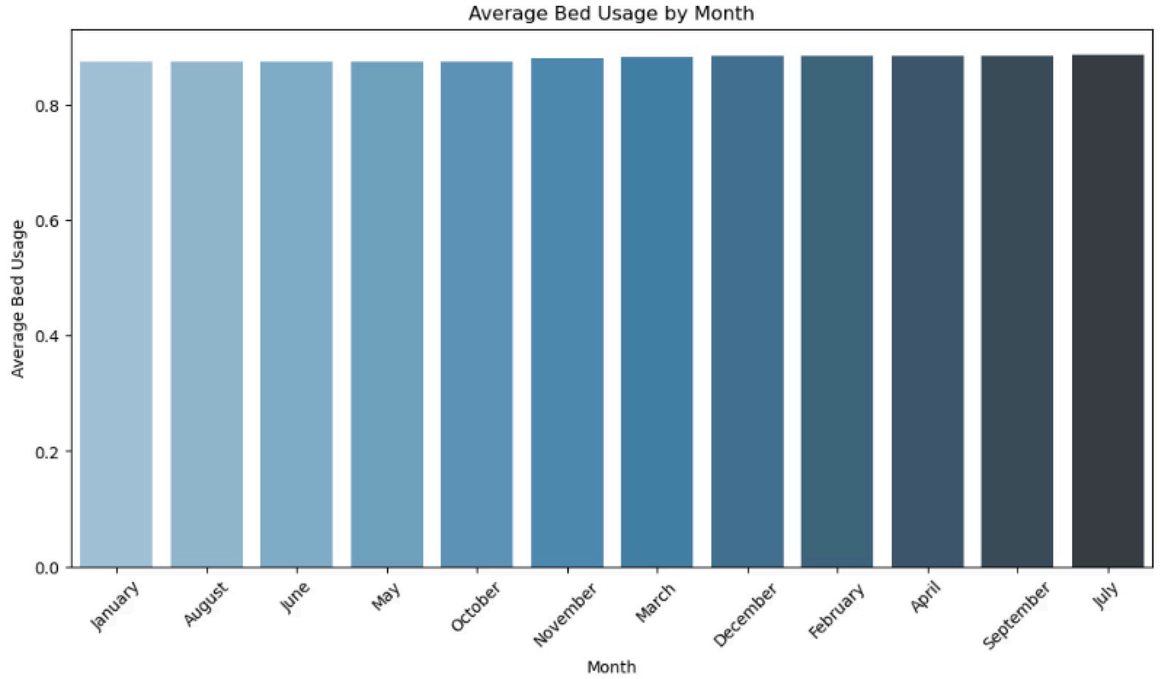
In [6]:
```
# Length of Stay vs Weekend and Holiday
plt.figure(figsize=(12, 6))
sns.boxplot(x='IsWeekend',
            y='LengthOfStay',
            data=df,
            palette='coolwarm')
plt.title("Length of Stay vs Weekend")
plt.xlabel("Is Weekend")
plt.ylabel("Length of Stay (days)")
plt.show()
```



In [43]:
```
plt.figure(figsize=(12, 6))
sns.boxplot(x='IsHoliday',
            y='LengthOfStay',
            data=df,
            palette='coolwarm')
plt.title("Length of Stay vs Holiday")
plt.xlabel("Is Holiday")
plt.ylabel("Length of Stay (days)")
plt.show()
```



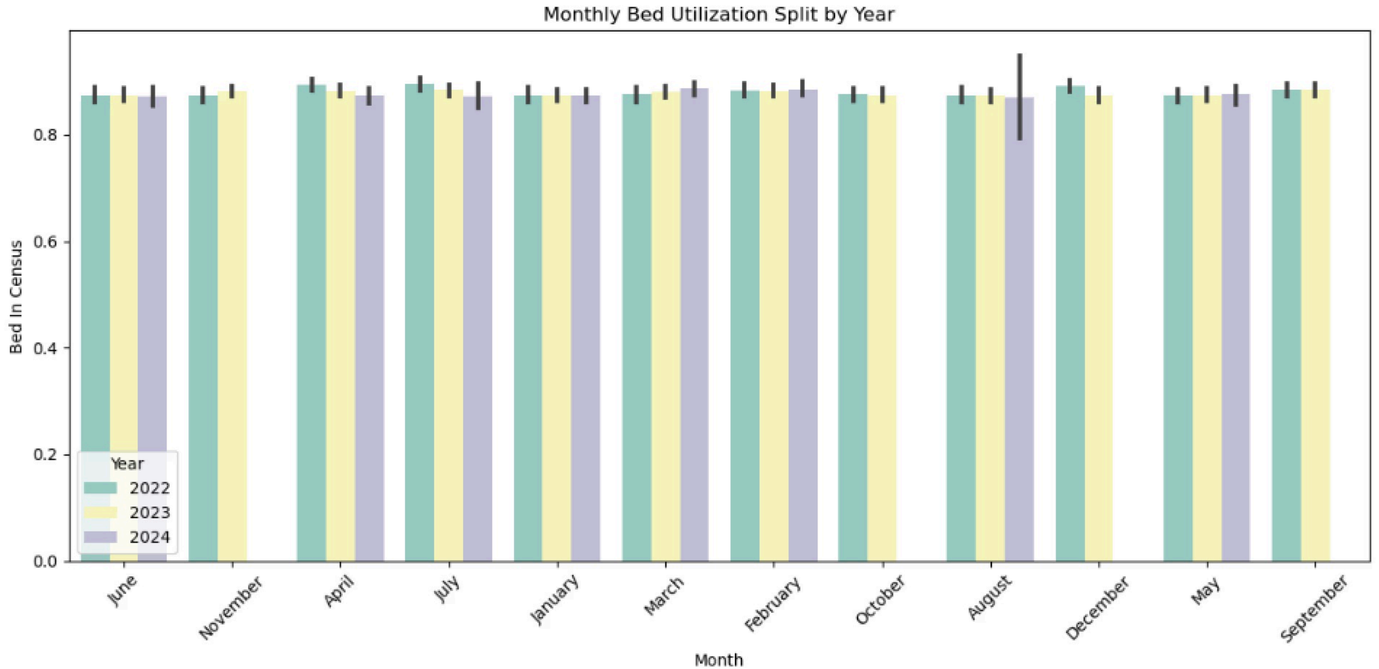
In [5]:
```
# Average Bed Usage by Month
monthly_usage = df.groupby('MonthName')['BedInCensus'].mean().sort_values()
plt.figure(figsize=(12, 6))
sns.barplot(x=monthly_usage.index,
            y=monthly_usage.values,
            palette='Blues_d')
plt.title("Average Bed Usage by Month")
plt.xlabel("Month")
plt.ylabel("Average Bed Usage")
plt.xticks(rotation=45)
plt.show()
```



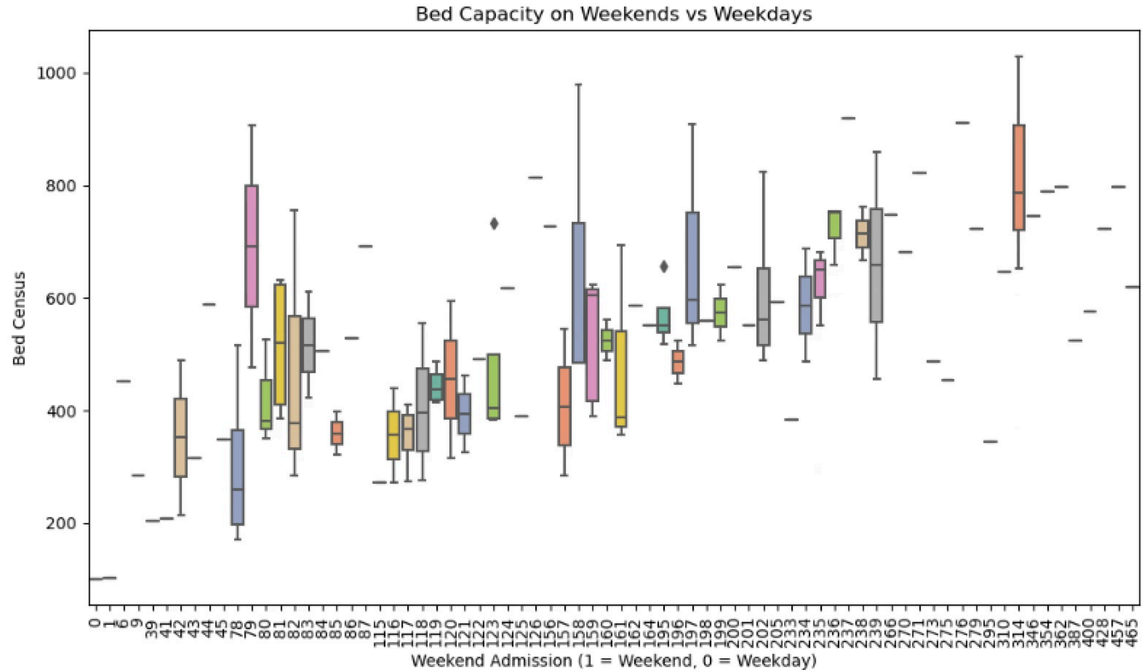
In [6]:
```
monthly_usage
```

Out[6]:
```
MonthName
January      0.873752
August       0.873974
June         0.874112
May          0.874461
October      0.874726
November     0.878896
March        0.881372
December     0.883701
February     0.883988
April        0.884126
September    0.884518
July         0.886836
Name: BedInCensus, dtype: float64
```

In [14]:
```
# 2. Bar Plot of Monthly Bed Utilization by Year
plt.figure(figsize=(12, 6))
sns.barplot(x='MonthName',
            y='BedInCensus',
            hue='Year',
            data=df, palette="Set3")
plt.title('Monthly Bed Utilization Split by Year')
plt.xlabel('Month')
plt.ylabel('Bed In Census')
plt.xticks(rotation=45)
plt.legend(title='Year')
plt.tight_layout()
plt.show()
```

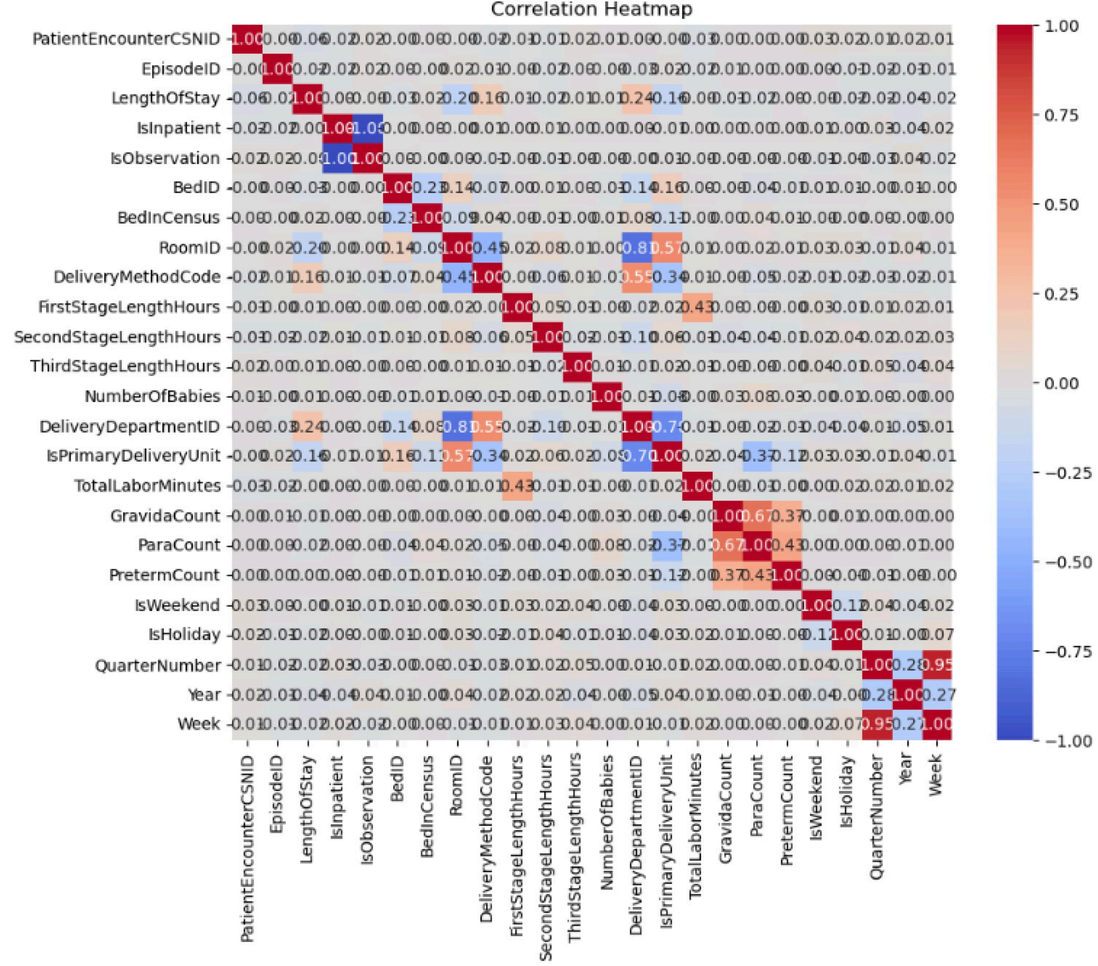


In [10]:
```
# 3. Weekend vs Weekday Admissions
plt.figure(figsize=(10, 6))
sns.boxplot(x='IsWeekendAdmission',
            y='BedInCensus',
            data=week_df,
            palette="Set2")
plt.title('Bed Capacity on Weekends vs Weekdays')
plt.xlabel('Weekend Admission (1 = Weekend, 0 = Weekday)')
plt.xticks(rotation = 90)
plt.ylabel('Bed Census')
plt.tight_layout()
plt.show()
```



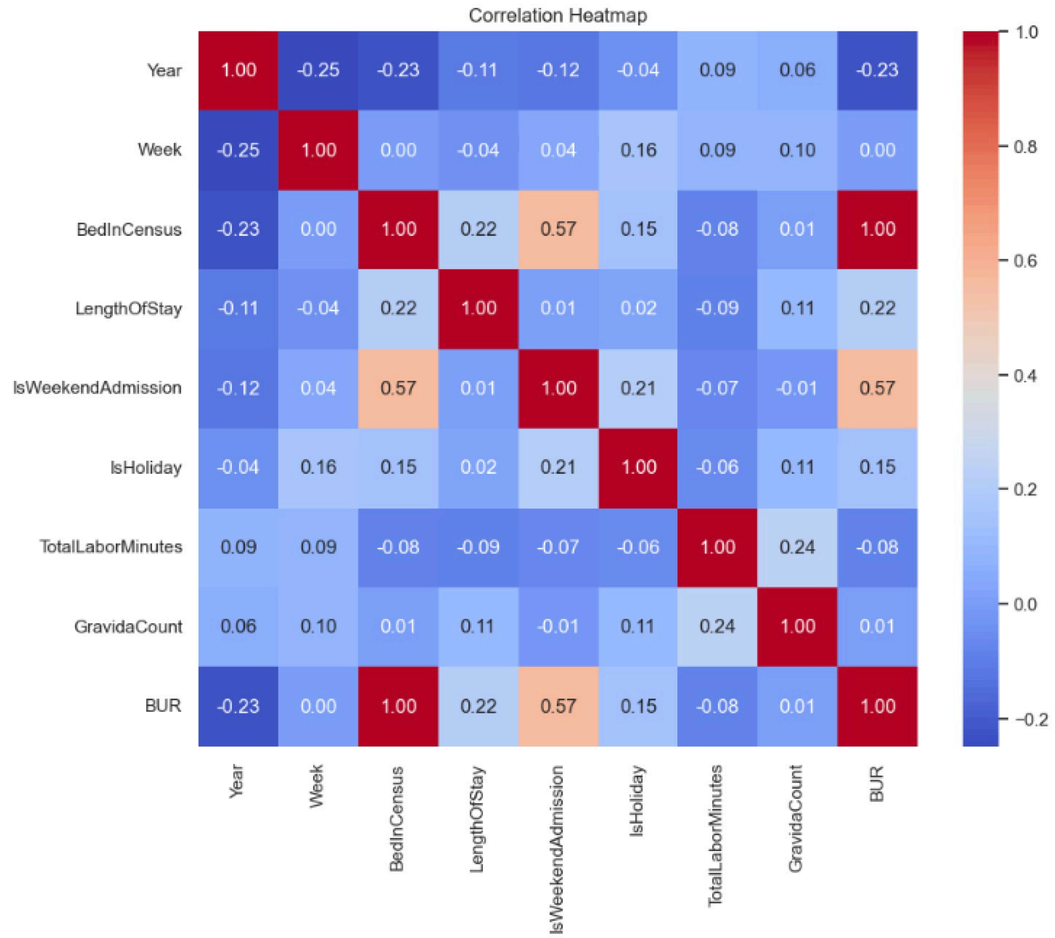
Key Features

In [37]:
```
# 4. Correlation Heatmap
numerical_data = df.select_dtypes(include='number')
plt.figure(figsize=(10, 8))
sns.heatmap(numerical_data.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap")
plt.show()
```



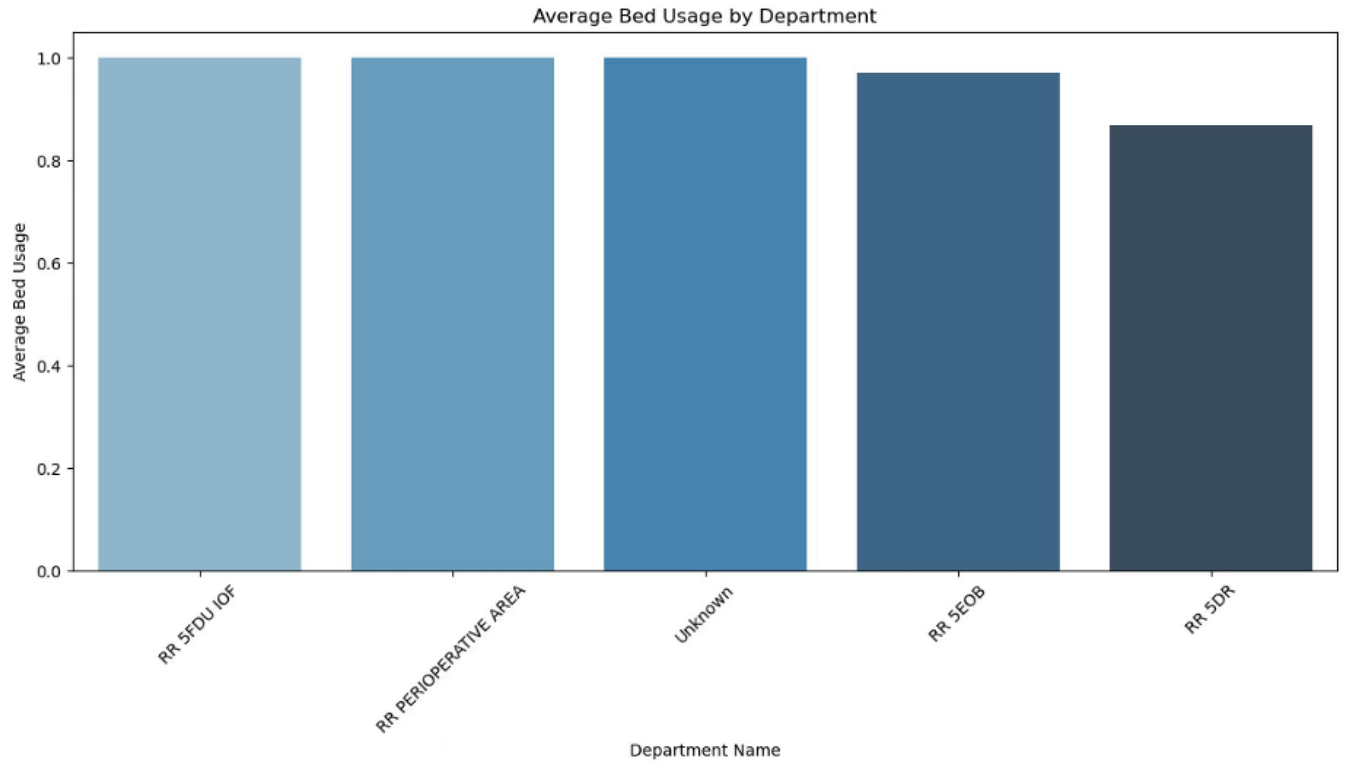
In [141…

```
# 4. Correlation Heatmap
numerical_data = week_df.select_dtypes(include='number')
plt.figure(figsize=(10, 8))
sns.heatmap(numerical_data.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title("Correlation Heatmap")
plt.show()
```



Department-Wise Bed Usage

In [11]:
```
# 5. Department-Wise Bed Usage
department_usage = df.groupby('DepartmentName')['BedInCensus'].mean().sort_values(ascending=False)
plt.figure(figsize=(14, 6))
sns.barplot(x=department_usage.index,
            y=department_usage.values,
            palette='Blues_d')
plt.title("Average Bed Usage by Department")
plt.xlabel("Department Name")
plt.ylabel("Average Bed Usage")
plt.xticks(rotation=45)
plt.show()
```



In [12]:
```
department_usage
```

Out[12]:
```
DepartmentName
RR 5FDU IOF              1.000000
RR PERIOPERATIVE AREA    1.000000
Unknown                  1.000000
RR 5EOB                  0.970588
RR 5DR                   0.868399
Name: BedInCensus, dtype: float64
```

Saving Revised Data

In [3]:
```
# save revised data
#df.to_csv('OB DELIVERY revised.csv', index=False)

# save week data for analysis
#week_df.to_csv('weekdf_del.csv', index=False)
```

In [ ]:
```

```

Loading Revised Data

In [4]:
```
df = pd.read_csv("OB DELIVERY revised.csv")
week_df = pd.read_csv("weekdf_del.csv")
```

In [5]:
```
df.describe()
```

Out[5]:

	PatientEncounterCSNID	EpisodeID	LengthOfStay	IsInpatient	IsObservation	BedID	BedInCensus	RoomID	DeliveryMethodCode	FirstStageLengthHours	...	IsPrimaryDeliveryUnit	TotalLaborMinutes	GravidaCount	ParaCount	PretermCount	IsWeekend	IsHoliday	QuarterNumber	Year	Week
count	8.095600e+04	8.095600e+04	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	...	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000	80956.000000
mean	1.081218e+09	2.301461e+18	3.672513	0.999531	0.000469	2254.307439	0.879688	623.987821	250.944315	30.230286	...	0.914139	2987.909247	2.003039	0.024816	0.002384	0.278571	0.035822	2.329488	2022.783969	24.103525
std	6.253885e+08	2.002128e+18	4.409546	0.021660	0.021660	1024.157985	0.325328	62.776563	2.149398	146.416771	...	0.280161	20045.251151	0.271861	0.218165	0.063511	0.448299	0.185847	1.124784	0.713189	15.167503
min	1.281280e+05	1.852020e+14	0.270139	0.000000	0.000000	1338.000000	0.000000	268.000000	250.000000	0.000000	...	0.000000	-524521.000000	1.000000	0.000000	0.000000	0.000000	0.000000	1.000000	2022.000000	1.000000
25%	5.400711e+08	6.242609e+17	2.250694	1.000000	0.000000	1852.000000	1.000000	632.000000	250.000000	14.000000	...	1.000000	1345.000000	2.000000	0.000000	0.000000	0.000000	0.000000	1.000000	2022.000000	11.000000
50%	1.060613e+09	1.809687e+18	2.898611	1.000000	0.000000	1853.000000	1.000000	635.000000	250.000000	14.000000	...	1.000000	1454.000000	2.000000	0.000000	0.000000	0.000000	0.000000	2.000000	2023.000000	22.000000
75%	1.641470e+09	3.408671e+18	3.855556	1.000000	0.000000	1853.000000	1.000000	637.000000	251.000000	14.000000	...	1.000000	1583.000000	2.000000	0.000000	0.000000	1.000000	0.000000	3.000000	2023.000000	37.000000
max	2.146801e+09	9.073822e+18	208.839583	1.000000	1.000000	5601.000000	1.000000	1187.000000	260.000000	2895.000000	...	1.000000	527585.000000	22.000000	10.000000	7.000000	1.000000	1.000000	4.000000	2024.000000	52.000000

8 rows × 24 columns

In [10]:
```
df
```

Out[10]:

	PatientEncounterCSNID	EpisodeID	HospitalAdmissionTime	HospitalDischargeTime	LengthOfStay	IsInpatient	IsObservation	BedID	BedName	BedInCensus	...	GravidaCount	ParaCount	PretermCount	IsWeekend	IsHoliday	MonthName	QuarterNumber	Year	Week	IsWeekendAdmission
0	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1848.0	A	1.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
1	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1856.0	02	0.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
2	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1849.0	A	1.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
3	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	1855.0	01	0.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
4	2.103032e+09	5.542547e+18	2022-06-07 16:44:00	2022-06-11 13:48:00	3.836806	1.0	0.0	4455.0	05	0.0	...	2.0	0.0	0.0	0.0	0.0	June	2.0	2022	23	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
80951	1.281286e+09	3.426151e+18	2023-02-12 19:35:00	2023-02-15 12:23:00	2.657639	1.0	0.0	1857.0	03	0.0	...	2.0	0.0	0.0	0.0	0.0	February	1.0	2023	6	True
80952	1.281286e+09	3.426151e+18	2023-02-12 19:35:00	2023-02-15 12:23:00	2.657639	1.0	0.0	1844.0	A	1.0	...	2.0	0.0	0.0	0.0	0.0	February	1.0	2023	6	True
80953	1.281286e+09	3.426151e+18	2023-02-12 19:35:00	2023-02-15 12:23:00	2.657639	1.0	0.0	1850.0	A	1.0	...	2.0	0.0	0.0	0.0	0.0	February	1.0	2023	6	True
80954	1.281286e+09	3.426151e+18	2023-02-12 19:35:00	2023-02-15 12:23:00	2.657639	1.0	0.0	4608.0	B	1.0	...	2.0	0.0	0.0	0.0	0.0	February	1.0	2023	6	True
80955	1.281280e+05	1.809687e+18	2023-03-27 23:52:00	2023-03-31 15:58:00	2.898611	1.0	0.0	1853.0	NaN	1.0	...	2.0	0.0	0.0	0.0	0.0	NaN	2.0	2023	13	False

80956 rows × 36 columns

In [95]:
```
df = pd.read_csv("weekdf_del.csv")
df.head()
```

Out[95]:

	Year	Week	BedInCensus	LengthOfStay	IsWeekendAdmission	IsHoliday	TotalLaborMinutes	GravidaCount	BUR
0	2022	1	349.0	2.773313	45	0.0	1668.666667	1.987469	0.339164
1	2022	2	723.0	3.356665	279	41.0	2510.565217	2.013285	0.702624
2	2022	3	417.0	2.729994	159	0.0	1228.469602	1.983229	0.405248
3	2022	4	631.0	3.856938	81	0.0	2415.542302	2.002774	0.613217
4	2022	5	389.0	3.680463	159	0.0	1988.824324	2.020270	0.378037

In [96]:
```
df.shape
```

Out[96]:
```
(135, 9)
```

Analysis

In [142…
```
# define features and target
X = df.drop(columns = ['BUR'])
y = df["BUR"]
```

In [143…
```
# Scaling the features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

In [144…
```
# train test split 80-20
X_tr, X_t, y_tr, y_t = train_test_split(X_scaled, y, test_size=0.2, random_state=42)
```

In [145…
```
print("Training set size:", X_tr.shape)
print("Test set size:", X_t.shape)
```

```
Training set size: (108, 8)
Test set size: (27, 8)
```

Random Forest Regressor

In [146…
```
# initialize model
rfrm = RandomForestRegressor(n_estimators = 100, random_state = 42)
```

In [147…
```
# train model
rfrm.fit(X_tr, y_tr)
```

Out[147]:
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In [148…
```
# predict
ypred = rfrm.predict(X_t)
```

In [149…
```
# model performance
mae = mean_absolute_error(y_t, ypred)
rmse = np.sqrt(mean_squared_error(y_t, ypred))
r2 = r2_score(y_t, ypred)

print(f"Mean Absolute Error (MAE): {mae:.4f}\nRoot Mean Squared Error (RMSE): {rmse:.4f}\nR-Squared (R²): {r2:.4f}")
```

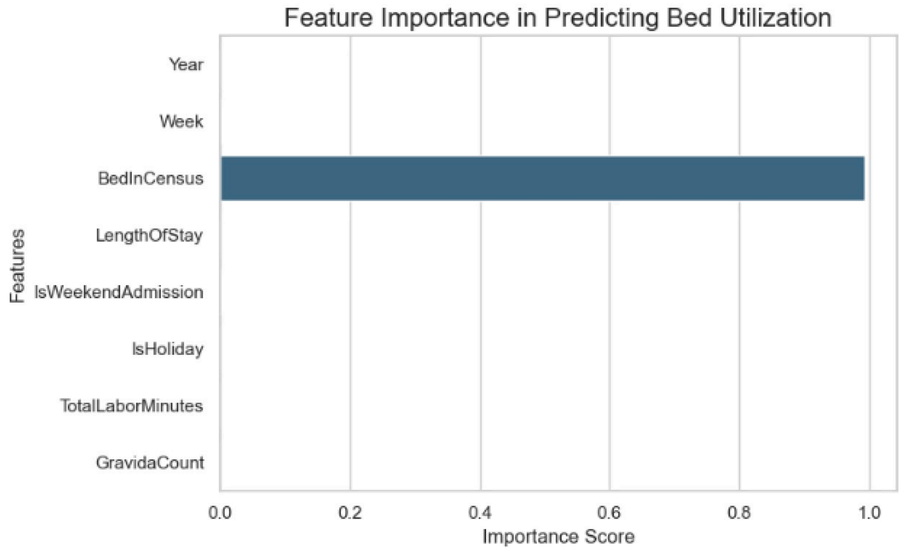
```
Mean Absolute Error (MAE): 0.0041
Root Mean Squared Error (RMSE): 0.0054
R-Squared (R²): 0.9987
```

NOTES:

- MAE: Random Forest Regressor Model's predicted bed utilization rate (BUR) differs from actual rate by 0.0054 units
- MSE: Random Forest Regressor Model avoids extreme deviations
- R^2: 89% of variability explained by model

In [107…
```
# feature importance
# Plot feature importances
importances = rfrm.feature_importances_
features = X.columns
```

In [108…
```
plt.figure(figsize=(8, 5))
sns.barplot(x=importances,
            y=features,
            palette='viridis')
plt.title('Feature Importance in Predicting Bed Utilization', fontsize=16)
plt.xlabel('Importance Score', fontsize=12)
plt.ylabel('Features', fontsize=12)
plt.tight_layout()
plt.show()
```



NOTES:

- BedInCensus accounts for almost 100% of predictive power, o/w none
- number of beds occupied affect bed utilization rate the most; all other factors less important

Improving Performance

In [109…
```
# Grid Search
# Define the parameter grid
param_grid = {
    'n_estimators': [50, 100, 200],          # Number of trees
    'max_depth': [10, 20, None],             # Maximum depth of trees
    'min_samples_split': [2, 5, 10],         # Min samples to split a node
    'min_samples_leaf': [1, 2, 4],           # Min samples at a leaf node
}
```

In [110…
```
# Initialize the Random Forest model
rf = RandomForestRegressor(random_state=42)
```

In [111…
```
# Set up Grid Search with 3-fold cross-validation
grid_search = GridSearchCV(estimator=rf,
                           param_grid=param_grid,
                           cv=3,
                           scoring='neg_mean_squared_error',
                           verbose=2,
                           n_jobs=-1)
```

In [112…
```
# Fit the grid search to the data
grid_search.fit(X_tr, y_tr)
```

```
Fitting 3 folds for each of 81 candidates, totalling 243 fits
```

Out[112]:

GridSearchCV
estimator: RandomForestRegressor
RandomForestRegressor

In [113…
```
# Get the best parameters and the best score
best_params = grid_search.best_params_
best_score = -grid_search.best_score_  # Convert back to positive since we're minimizing MSE
print("Best Parameters:", best_params)
print(f"Best Cross-Validated MSE: {best_score:.4f}")
```

```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 100}
Best Cross-Validated MSE: 0.0005
```

In [116…
```
# Train the final model using the best parameters
best_rf_model = RandomForestRegressor(**best_params, random_state=42)
best_rf_model.fit(X_tr, y_tr)
```

Out[116]:

RandomForestRegressor
RandomForestRegressor(max_depth=10, min_samples_leaf=2, random_state=42)

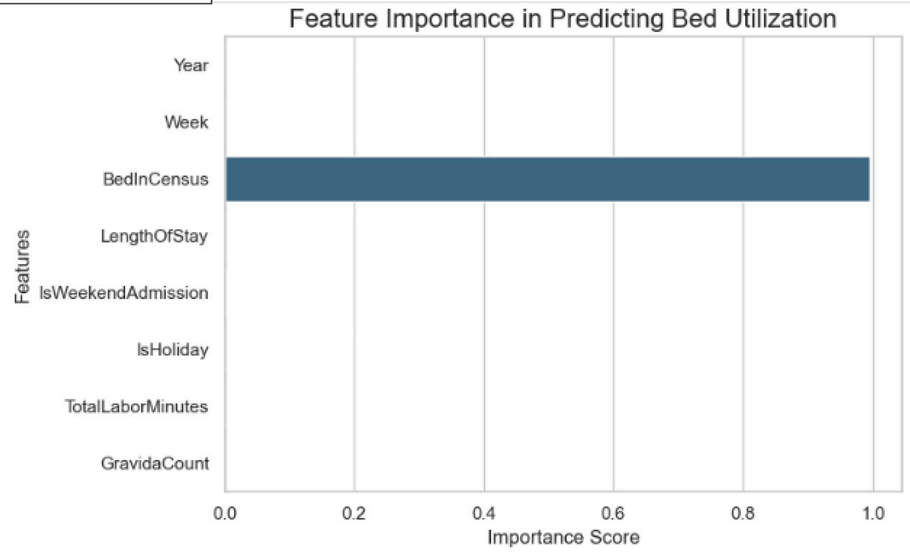
In [117…
```
# Evaluate the tuned model on the test set
y_pred_tuned = best_rf_model.predict(X_t)
mae_tuned = mean_absolute_error(y_t, y_pred_tuned)
rmse_tuned = np.sqrt(mean_squared_error(y_t, y_pred_tuned))
r2_tuned = r2_score(y_t, y_pred_tuned)

print(f"Tuned Model MAE: {mae_tuned:.4f}\nTuned Model RMSE: {rmse_tuned:.4f}\nTuned Model R²: {r2_tuned:.4f}")
```

```
Tuned Model MAE: 0.0042
Tuned Model RMSE: 0.0054
Tuned Model R²: 0.9987
```

In [118…
```
# feature importance
# Plot feature importances
importances = best_rf_model.feature_importances_
features = X.columns
```

In [119…
```
plt.figure(figsize=(8, 5))
sns.barplot(x=importances,
            y=features,
            palette='viridis')
plt.title('Feature Importance in Predicting Bed Utilization', fontsize=16)
plt.xlabel('Importance Score', fontsize=12)
plt.ylabel('Features', fontsize=12)
plt.tight_layout()
plt.show()
```



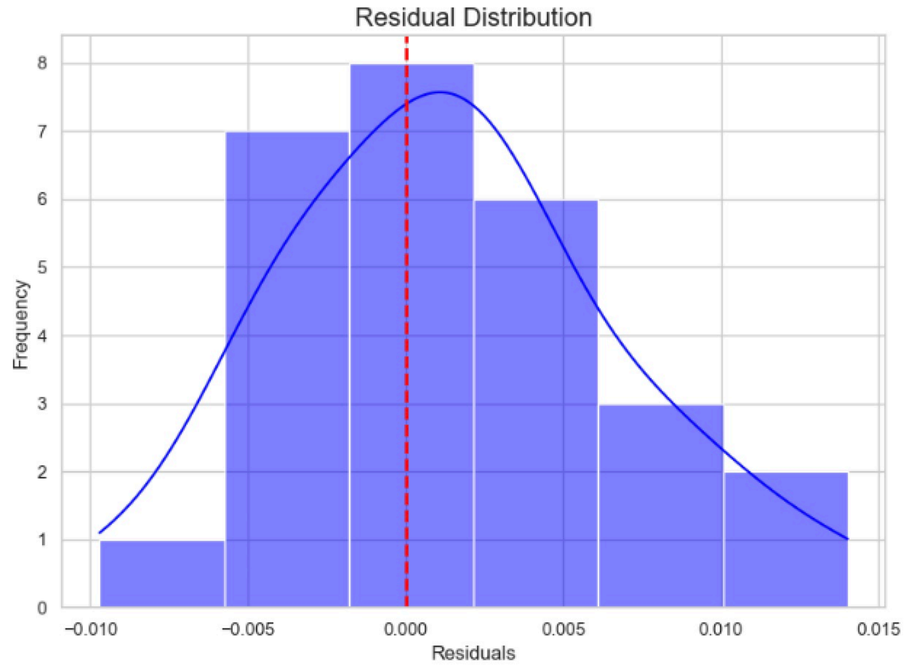
Diagnostics

In [120…

```
# Calculate residuals
residuals = y_t - y_pred_tuned
```

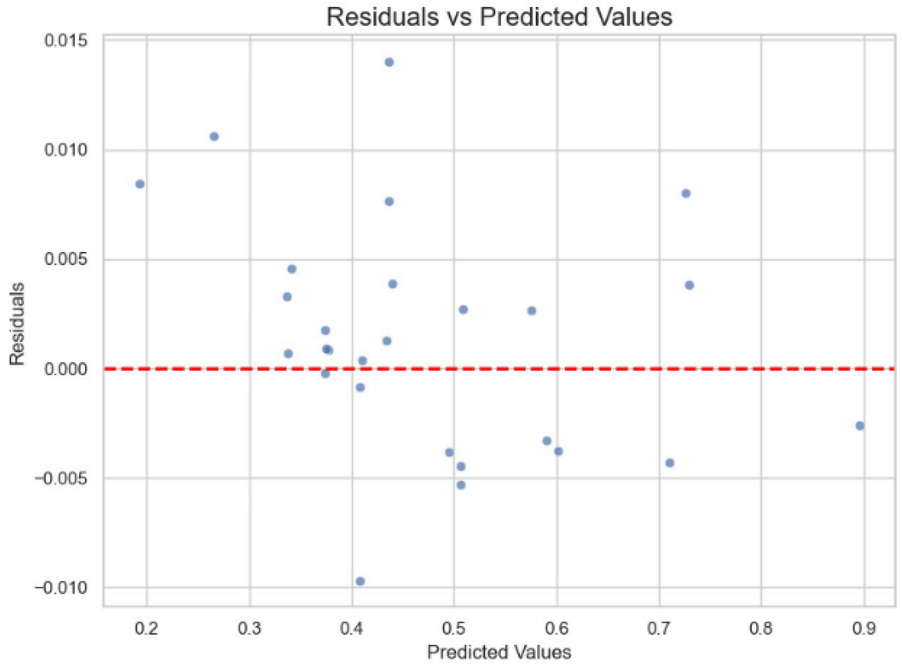
In [121…

```
# Plot residuals
plt.figure(figsize=(8, 6))
sns.histplot(residuals, kde=True, color='blue')
plt.title('Residual Distribution', fontsize=16)
plt.xlabel('Residuals', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.axvline(0, color='red', linestyle='--', linewidth=2)
plt.tight_layout()
plt.show()
```



In [122…

```
# 2. Plot residuals vs. predicted values
plt.figure(figsize=(8, 6))
sns.scatterplot(x=y_pred_tuned, y=residuals, alpha=0.7)
plt.axhline(0, color='red', linestyle='--', linewidth=2)
plt.title('Residuals vs Predicted Values', fontsize=16)
plt.xlabel('Predicted Values', fontsize=12)
plt.ylabel('Residuals', fontsize=12)
plt.tight_layout()
plt.show()
```



In [123…
```
# Residuals Standard Deviation
residuals_std = np.std(residuals)
print(f"Residuals Standard Deviation: {residuals_std:.4f}")
```

```
Residuals Standard Deviation: 0.0052
```

- Possible mild bias; residuals not exactly around 0
- Some heteroskedasticity maybe and a bit patterned (try to test other models)

In [124…
```
xgb_model = XGBRegressor(n_estimators=200,
                         max_depth=6,
                         learning_rate=0.1,
                         random_state=42)
xgb_model.fit(X_tr, y_tr)
```

Out[124]:

XGBRegressor

```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=0.1, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=200, n_jobs=None,
             num_parallel_tree=None, random_state=42, ...)
```

In [125…
```
# Predictions and evaluation
y_pred_xgb = xgb_model.predict(X_t)
r2_xgb = r2_score(y_t, y_pred_xgb)
print(f"XGBoost R²: {r2_xgb:.4f}")
```

```
XGBoost R²: 0.9992
```

Gradient Boosting Regressor (Gradient Boosting Machines)

In [126…
```
# Initialize the Gradient Boosting Regressor
gbm = GradientBoostingRegressor(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=6,
    random_state=42
)
```

In [127…
```
# Fit the model
gbm.fit(X_tr, y_tr)
```

Out[127]:
```
GradientBoostingRegressor
GradientBoostingRegressor(max_depth=6, random_state=42)
```

In [128…
```
# Predictions
ypred = gbm.predict(X_t)
```

In [129…
```
# Evaluation
mae_gbm = mean_absolute_error(y_t, ypred)
rmse_gbm = np.sqrt(mean_squared_error(y_t, ypred))
r2_gbm = r2_score(y_t, ypred)

print(f"Gradient Boosting MAE: {mae_gbm:.4f}")
print(f"Gradient Boosting RMSE: {rmse_gbm:.4f}")
print(f"Gradient Boosting R²: {r2_gbm:.4f}")
```

```
Gradient Boosting MAE: 0.0054
Gradient Boosting RMSE: 0.0079
Gradient Boosting R²: 0.9973
```

Improving Performance

In [130…
```
# Define the GradientBoostingRegressor
gbm = GradientBoostingRegressor(random_state=42)
```

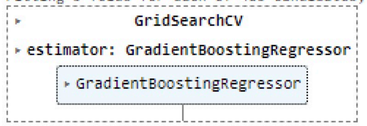
In [131…
```
# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],       # Number of trees
    'learning_rate': [0.01, 0.05, 0.1],   # Learning rate
    'max_depth': [3, 5, 7],               # Maximum depth of trees
    'min_samples_split': [2, 5, 10],      # Min samples to split a node
    'min_samples_leaf': [1, 3, 5],        # Min samples at a leaf node
    'subsample': [0.8, 1.0]               # Subsampling fraction
}
```

In [132…
```
# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=gbm,
    param_grid=param_grid,
    scoring='neg_mean_squared_error',  # Use MSE as the metric
    cv=3,  # 3-fold cross-validation
    verbose=2,
    n_jobs=-1  # Use all available cores
)
```

In [133…
```
# Fit the grid search
grid_search.fit(X_tr, y_tr)
```

```
Fitting 3 folds for each of 486 candidates, totalling 1458 fits
```

Out[133]:



In [134…
```
# Retrieve the best parameters
best_params = grid_search.best_params_
print("Best Parameters for GradientBoostingRegressor:", best_params)
```

```
Best Parameters for GradientBoostingRegressor: {'learning_rate': 0.1, 'max_depth': 7, 'min_samples_leaf': 1, 'min_samples_split': 10, 'n_estimators': 300, 'subsample': 0.8}
```

In [135…
```
# Train the best model on the full training set
gbm_best = GradientBoostingRegressor(**best_params, random_state=42)
gbm_best.fit(X_tr, y_tr)
```

Out[135]:

```
GradientBoostingRegressor
GradientBoostingRegressor(max_depth=7, min_samples_split=10, n_estimators=300,
                          random_state=42, subsample=0.8)
```

In [136…
```
# Predict on the test set
y_pred_gbm_best = gbm_best.predict(X_t)
```

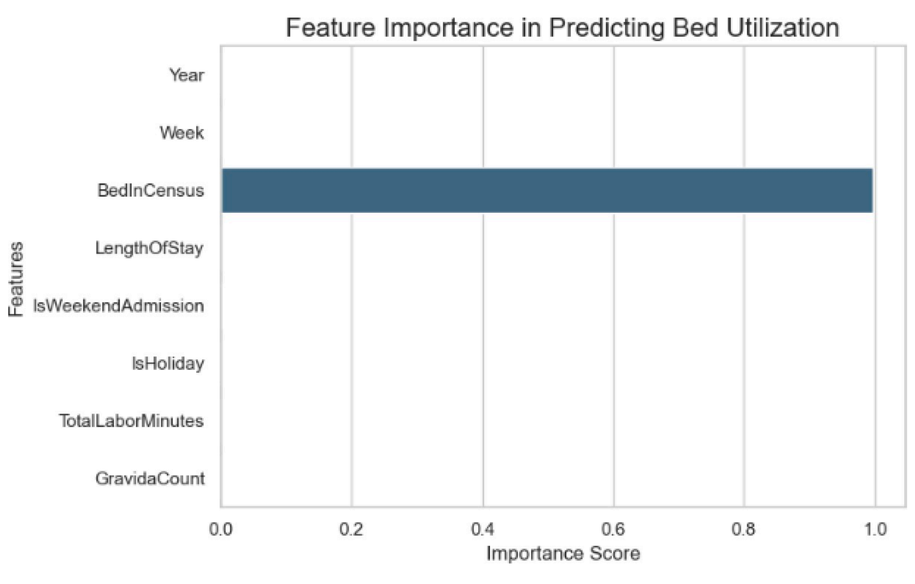
In [137…
```
# Evaluate the best model
mae_gbm_best = mean_absolute_error(y_t, y_pred_gbm_best)
rmse_gbm_best = np.sqrt(mean_squared_error(y_t, y_pred_gbm_best))
r2_gbm_best = r2_score(y_t, y_pred_gbm_best)

print(f"Tuned GradientBoostingRegressor MAE: {mae_gbm_best:.4f}")
print(f"Tuned GradientBoostingRegressor RMSE: {rmse_gbm_best:.4f}")
print(f"Tuned GradientBoostingRegressor R²: {r2_gbm_best:.4f}")
```

```
Tuned GradientBoostingRegressor MAE: 0.0030
Tuned GradientBoostingRegressor RMSE: 0.0043
Tuned GradientBoostingRegressor R²: 0.9992
```

In [138…
```
# feature importance
# Plot feature importances
importances = gbm_best.feature_importances_
features = X.columns
```

In [139…
```
plt.figure(figsize=(8, 5))
sns.barplot(x=importances,
            y=features,
            palette='viridis')
plt.title('Feature Importance in Predicting Bed Utilization', fontsize=16)
plt.xlabel('Importance Score', fontsize=12)
plt.ylabel('Features', fontsize=12)
plt.tight_layout()
plt.show()
```


Feature Importance in Predicting Bed Utilization
Year
Week
BedInCensus
LengthOfStay
IsWeekendAdmission
IsHoliday
TotalLaborMinutes
GravidaCount
Features
0.0
0.2
0.4
0.6
0.8
1.0
Importance Score