In [2]:
```
# 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
```

Loading Data (RR 5DR) FOCUS ON DELIVERIES

In [3]:
```
df = pd.read_csv("OB.csv")
#df = pd.read_csv("ob_extended.csv")
df.head()
```

```
C:\Users\local_CAFu\Temp\12\ipykernel_7892\1047085193.py:1: DtypeWarning: Columns (3,27,28,30,31,33) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("OB.csv")
```

Out[3]:

	PatientEncounterCSNID	PatientID	CheckinTime	CheckoutTime	HospitalAdmissionTime	HospitalDischargeTime	DepartmentID	LengthOfStay	LaborStatusCode	LastRoomID	...	BirthDateTime	DeliveryMethodCode	CesareanReason	AnesthesiaMethod	NumberOfBabies	EstimatedDateOfDelivery	TotalL
0	1792773576	261909045	NaN	NaN	NaN	NaN	60373	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
1	1738080526	1822641858	NaN	NaN	NaN	NaN	60369	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
2	814139051	575210674	NaN	NaN	NaN	NaN	50637	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
3	651554377	757285833	NaN	NaN	NaN	NaN	60373	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
4	1119728660	627688132	NaN	NaN	NaN	NaN	60373	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 38 columns

In [4]:
```
df.shape
```

Out[4]:
```
(1000000, 38)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['PatientEncounterCSNID', 'PatientID', 'CheckinTime', 'CheckoutTime',
       'HospitalAdmissionTime', 'HospitalDischargeTime', 'DepartmentID',
       'LengthOfStay', 'LaborStatusCode', 'LastRoomID', 'IsInpatient',
       'IsObservation', 'DepartmentName', 'BedID', 'BedName', 'BedInCensus',
       'RoomID', 'RoomName', 'RoomGroupName', 'DateValue', 'IsWeekend',
       'IsHoliday', 'MonthName', 'QuarterNumber', 'Year',
       'DeliveryEncounterCSNID', 'LaborOnsetDateTime', 'DeliveryDate',
       'BirthDateTime', 'DeliveryMethodCode', 'CesareanReason',
       'AnesthesiaMethod', 'NumberOfBabies', 'EstimatedDateOfDelivery',
       'TotalLaborMinutes', 'GravidaCount', 'ParaCount', 'PretermCount'],
      dtype='object')
```

In [2]:
```
# 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
```

Loading Data (RR 5DR) FOCUS ON DELIVERIES

In [3]:
```
df = pd.read_csv("OB.csv")
#df = pd.read_csv("ob_extended.csv")
df.head()
```

```
C:\Users\local_CAFu\Temp\12\ipykernel_7892\1047085193.py:1: DtypeWarning: Columns (3,27,28,30,31,33) have mixed types. Specify dtype option on import or set low_memory=False.
  df = pd.read_csv("OB.csv")
```

Out[3]:

	PatientEncounterCSNID	PatientID	CheckinTime	CheckoutTime	HospitalAdmissionTime	HospitalDischargeTime	DepartmentID	LengthOfStay	LaborStatusCode	LastRoomID	...	BirthDateTime	DeliveryMethodCode	CesareanReason	AnesthesiaMethod	NumberOfBabies	EstimatedDateOfDelivery	TotalL
0	1792773576	261909045	NaN	NaN	NaN	NaN	60373	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
1	1738080526	1822641858	NaN	NaN	NaN	NaN	60369	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
2	814139051	575210674	NaN	NaN	NaN	NaN	50637	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
3	651554377	757285833	NaN	NaN	NaN	NaN	60373	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	
4	1119728660	627688132	NaN	NaN	NaN	NaN	60373	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	

5 rows × 38 columns

In [4]:
```
df.shape
```

Out[4]:
```
(1000000, 38)
```

In [5]:
```
df.columns
```

Out[5]:
```
Index(['PatientEncounterCSNID', 'PatientID', 'CheckinTime', 'CheckoutTime',
       'HospitalAdmissionTime', 'HospitalDischargeTime', 'DepartmentID',
       'LengthOfStay', 'LaborStatusCode', 'LastRoomID', 'IsInpatient',
       'IsObservation', 'DepartmentName', 'BedID', 'BedName', 'BedInCensus',
       'RoomID', 'RoomName', 'RoomGroupName', 'DateValue', 'IsWeekend',
       'IsHoliday', 'MonthName', 'QuarterNumber', 'Year',
       'DeliveryEncounterCSNID', 'LaborOnsetDateTime', 'DeliveryDate',
       'BirthDateTime', 'DeliveryMethodCode', 'CesareanReason',
       'AnesthesiaMethod', 'NumberOfBabies', 'EstimatedDateOfDelivery',
       'TotalLaborMinutes', 'GravidaCount', 'ParaCount', 'PretermCount'],
      dtype='object')
```

EDA

In [6]:
```
# info
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 177438 entries, 0 to 177437
Data columns (total 30 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   PatientEncounterCSNID     177438 non-null  int64
 1   EpisodeID                 177437 non-null  float64
 2   CheckinTime               39 non-null      object
 3   CheckoutTime              39 non-null      object
 4   HospitalAdmissionTime     177438 non-null  object
 5   HospitalDischargeTime     176927 non-null  object
 6   LengthOfStay              176886 non-null  float64
 7   IsInpatient               177438 non-null  int64
 8   IsObservation             177438 non-null  int64
 9   DeliveryDate              177438 non-null  object
 10  BirthDateTime             170051 non-null  object
 11  LaborOnsetDateTime        103317 non-null  object
 12  DeliveryMethodCode        169791 non-null  float64
 13  FirstStageLengthHours     82496 non-null   float64
 14  SecondStageLengthHours    111873 non-null  float64
 15  ThirdStageLengthHours     172790 non-null  float64
 16  NumberOfBabies            7385 non-null    float64
 17  EstimatedDateOfDelivery   7377 non-null    object
 18  DeliveryDepartmentID      170041 non-null  float64
 19  DepartmentName            177438 non-null  object
 20  IsPrimaryDeliveryUnit     177438 non-null  int64
 21  TotalLaborMinutes         100634 non-null  float64
 22  GravidaCount              7385 non-null    float64
 23  ParaCount                 7385 non-null    float64
 24  PretermCount              7385 non-null    float64
 25  IsWeekend                 177438 non-null  int64
 26  IsHoliday                 177438 non-null  int64
 27  MonthName                 177438 non-null  object
 28  QuarterNumber             177438 non-null  int64
 29  Year                      177438 non-null  int64
dtypes: float64(12), int64(8), object(10)
memory usage: 40.6+ MB
```

In [7]:
```
# summary statistics
df.describe()
```

Out[7]:

	PatientEncounterCSNID	EpisodeID	LengthOfStay	IsInpatient	IsObservation	DeliveryMethodCode	FirstStageLengthHours	SecondStageLengthHours	ThirdStageLengthHours	NumberOfBabies	DeliveryDepartmentID	IsPrimaryDeliveryUnit	TotalLaborMinutes	GravidaCount	ParaCount	Preterm
count	1.774380e+05	1.774370e+05	176886.000000	177438.000000	177438.000000	169791.000000	82496.000000	111873.000000	172790.000000	7385.000000	170041.000000	177438.000000	1.006340e+05	7385.000000	7385.000000	7385.
mean	1.098996e+09	2.245642e+18	3.725850	0.999769	0.000220	250.918093	58.945646	7.335407	9.957758	1.018957	10026.615787	0.911676	5.494986e+03	2.085579	0.596750	0.
std	6.166410e+08	1.995993e+18	5.665832	0.015199	0.014824	4.236137	608.837253	2070.475163	198.916489	0.139331	38.729126	0.283766	3.908809e+04	1.321540	0.867987	0.
min	7.261865e+06	3.640588e+13	0.270139	0.000000	0.000000	7.000000	0.000000	0.000000	0.000000	1.000000	10018.000000	0.000000	-5.245210e+05	1.000000	0.000000	0.
25%	5.667608e+08	6.143201e+17	2.258333	1.000000	0.000000	250.000000	6.000000	0.000000	0.000000	1.000000	10018.000000	1.000000	9.300000e+02	1.000000	0.000000	0.
50%	1.093594e+09	1.669475e+18	2.927083	1.000000	0.000000	250.000000	14.000000	0.000000	0.000000	1.000000	10018.000000	1.000000	1.458000e+03	2.000000	0.000000	0.
75%	1.649508e+09	3.375105e+18	3.816667	1.000000	0.000000	251.000000	24.000000	2.000000	0.000000	1.000000	10018.000000	1.000000	2.312000e+03	3.000000	1.000000	0.
max	2.147378e+09	9.093663e+18	208.839583	1.000000	1.000000	260.000000	26405.000000	692521.000000	5259.000000	3.000000	10201.000000	1.000000	1.584861e+06	22.000000	10.000000	7.

In [9]:
```
# missing data
# many columns with near or exactly null values close to 1 million
df.isnull().sum()
```

Out[9]:
```
PatientEncounterCSNID           0
PatientID                       0
CheckinTime                1000000
CheckoutTime                999952
HospitalAdmissionTime       952715
HospitalDischargeTime       952809
DepartmentID                    0
LengthOfStay                999997
LaborStatusCode             999997
LastRoomID                  999997
IsInpatient                     0
IsObservation                   0
DepartmentName                  0
BedID                       660611
BedName                     660611
BedInCensus                 660611
RoomID                      575746
RoomName                    570041
RoomGroupName               991412
DateValue                       0
IsWeekend                       0
IsHoliday                       0
MonthName                       0
QuarterNumber                   0
Year                            0
DeliveryEncounterCSNID      999997
LaborOnsetDateTime         1000000
DeliveryDate                999997
BirthDateTime               999998
DeliveryMethodCode          999998
CesareanReason              999998
AnesthesiaMethod            999998
NumberOfBabies              999999
EstimatedDateOfDelivery     999999
TotalLaborMinutes          1000000
GravidaCount                999999
ParaCount                   999999
PretermCount                999999
dtype: int64
```

Data Cleaning

In [3]:
```
# keep relevant columns; remove null columns, ignore ID columns (except bed maybe)
cols = ["DepartmentID",
       "IsInpatient",
       "IsObservation",
       "DepartmentName",
       "BedID",
       "BedName",
       "BedInCensus",
       "RoomID",
       "RoomName",
       "RoomGroupName",
       "DateValue",
       "IsWeekend",
       "IsHoliday",
       "MonthName",
       "QuarterNumber",
       "Year"]

df = df[cols]
```

In [4]:
```
# drop missing values
# drop beds that are missing
df = df.dropna(subset = ['BedID'])
```

In [5]:
```
# replace missing values
df["BedInCensus"] = df["BedInCensus"].fillna(0)
```

In [80]:
```
# remove duplicates
# skip
```

In [6]:
```
#### IMPORTANT: TIMEFRAME IS 1/1/2022 to 7/1/2024 ########
df = df[(df['DateValue'] >= '2022-01-01') & (df['DateValue'] <= '2024-07-01')]
```

In [7]:
```
# data type revisions
# temporal
df["DateValue"] = pd.to_datetime(df["DateValue"], errors = 'coerce')
df["WeekNum"] = df["DateValue"].dt.isocalendar().week
df["WeekYear"] = df["Year"].astype(str) + '-' + df["WeekNum"].astype(str)
df["IsWeekend"] = df["DateValue"].dt.weekday.isin([5,6])
```

In [8]:
```
df['TotalAdmissions'] = df['BedID'].notnull().astype(int)
```

In [119…
```
df['TotalAdmissions'].describe()
```

Out[119]:
```
count    20710.0
mean         1.0
std          0.0
min          1.0
25%          1.0
50%          1.0
75%          1.0
max          1.0
Name: TotalAdmissions, dtype: float64
```

In [9]:
```
# drop inpatient and observation since all 0
df[['IsInpatient', 'IsObservation']].describe()
```

Out[9]:

	IsInpatient	IsObservation
count	20710.0	20710.0
mean	0.0	0.0
std	0.0	0.0
min	0.0	0.0
25%	0.0	0.0
50%	0.0	0.0
75%	0.0	0.0
max	0.0	0.0

In [120…
```
# inspect revised data
df.head()
```

Out[120]:

	DepartmentID	IsInpatient	IsObservation	DepartmentName	BedID	BedName	BedInCensus	RoomID	RoomName	RoomGroupName	DateValue	IsWeekend	IsHoliday	MonthName	QuarterNumber	Year	WeekNum	WeekYear	TotalAdmissions
628	20013	0	0	SM 2OBG	5004.0	1	1.0	773.0	2424	NaN	2022-05-15	True	0	May	2	2022	19	2022-19	1
707	20013	0	0	SM 2OBG	5023.0	1	1.0	791.0	2490	NaN	2022-05-15	True	0	May	2	2022	19	2022-19	1
709	20013	0	0	SM 2OBG	5018.0	1	1.0	786.0	2472	NaN	2022-05-15	True	0	May	2	2022	19	2022-19	1
711	20013	0	0	SM 2OBG	5022.0	1	1.0	790.0	2484	NaN	2022-05-15	True	0	May	2	2022	19	2022-19	1
780	20024	0	0	SM 2OBG MS	9816.0	A	1.0	1609.0	2420	NaN	2022-07-22	False	0	July	3	2022	29	2022-29	1

In [121…
```
# new dimensions of revised data
df.shape
```

Out[121]:
```
(20710, 19)
```

In [10]:
```
# aggregate data to weekly level
week_df = df.groupby('WeekYear').agg({
    "TotalAdmissions": 'sum',
    "BedID": 'nunique',
    "BedInCensus": "sum",
    "IsWeekend": "sum",
    "IsHoliday": "sum",
    "DepartmentID": 'nunique'
}).reset_index()
```

In [11]:
```
# rename columns
week_df.rename(columns={
    "TotalAdmissions": 'TotalAdmissions',
    "BedID": 'UniqueBeds',
    "BedInCensus": "TotalBedCensus",
    "IsWeekend": "WeekendDays",
    "IsHoliday": "HolidayDays",
    "DepartmentID": 'UniqueDepts'
}, inplace = True)
```

In [12]:
```
# weekly bed capacity; bed utilization rate (BUR)
week_df["BUR"] = week_df["TotalBedCensus"] / week_df["UniqueBeds"]

# inf/NaN fix
week_df["BUR"] = week_df["BUR"].fillna(0)
```

In [125…
```
week_df.head()
```

Out[125]:

	WeekYear	TotalAdmissions	UniqueBeds	TotalBedCensus	WeekendDays	HolidayDays	UniqueDepts	TotalIP	TotalObs	BUR
0	2022-1	74	54	72.0	3	0	3	0	0	1.333333
1	2022-10	101	50	98.0	49	0	2	0	0	1.960000
2	2022-11	176	79	174.0	78	0	3	0	0	2.202532
3	2022-12	203	79	199.0	62	53	3	0	0	2.518987
4	2022-13	50	49	49.0	49	0	2	0	0	1.000000

In [126…
```
week_df.shape
```

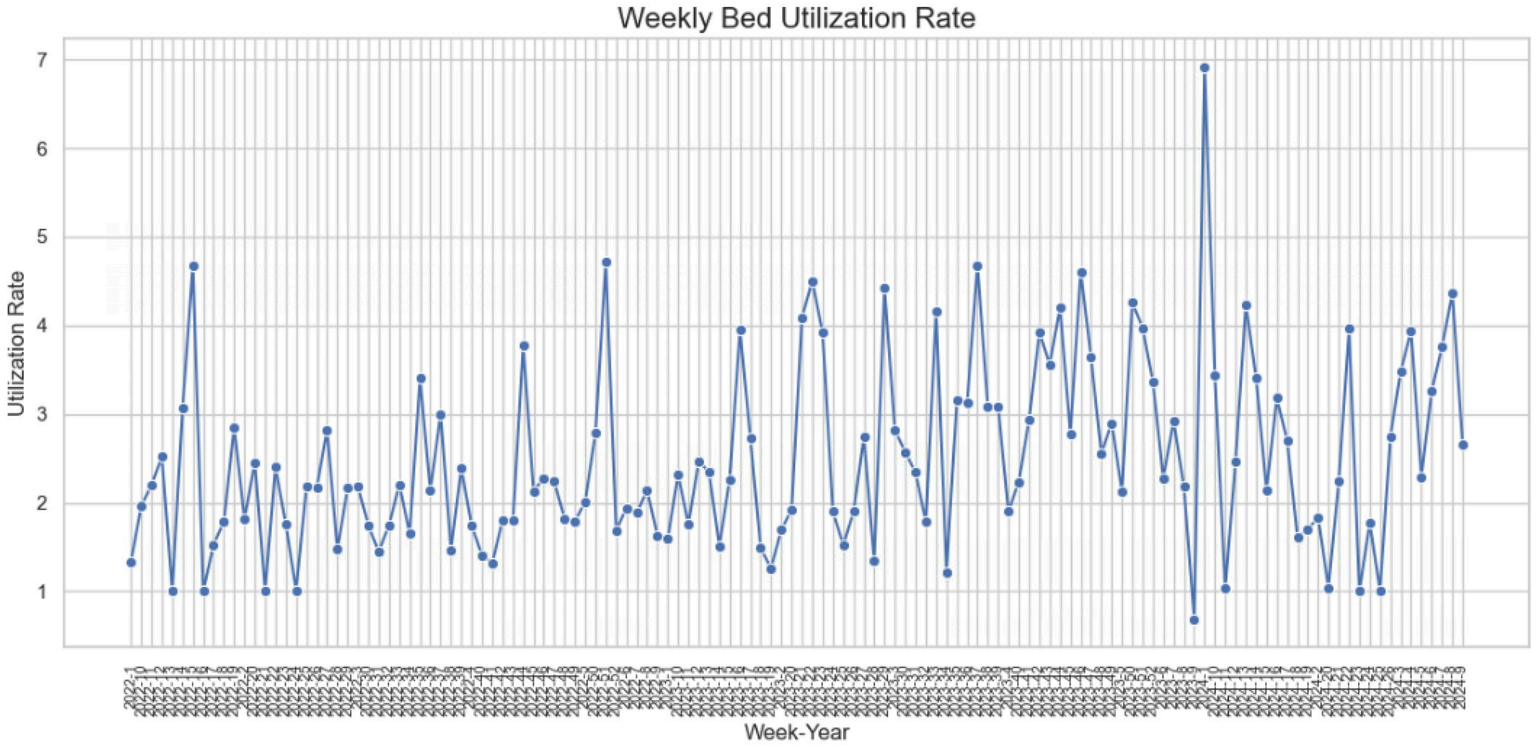
Out[126]:
```
(130, 10)
```

Data Visualizations

In [127…
```
sns.set(style="whitegrid")
```

Weekly Bed Utilization Trends

In [128…
```
# weekly bed utilization trends
plt.figure(figsize=(12, 6))
sns.lineplot(data=week_df,
             x='WeekYear',
             y='BUR',
             marker='o')
plt.title('Weekly Bed Utilization Rate', fontsize=16)
plt.xlabel('Week-Year', fontsize=12)
plt.ylabel('Utilization Rate', fontsize=12)
plt.xticks(rotation=90, fontsize=8)
plt.tight_layout()
plt.show()
```



Seasonal Analysis

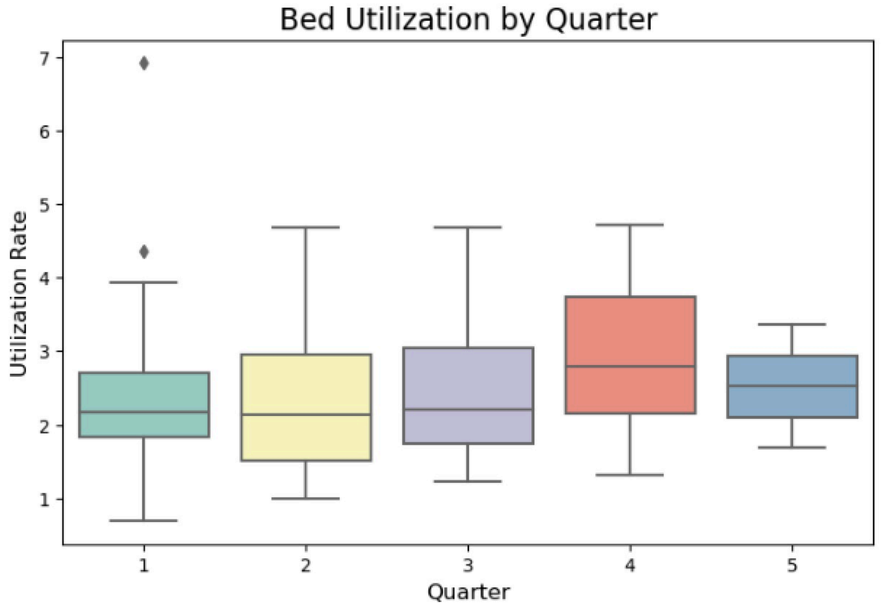
In [13]:
```
# Add seasonal features to weekly data
week_df['Quarter'] = week_df['WeekYear'].str.split('-').str[1].astype(int) // 13 + 1
```

In [15]:
```
week_df.head() # inspect quarter features; 3 months per quarter is roughly 12 weeks
```

Out[15]:

	WeekYear	TotalAdmissions	UniqueBeds	TotalBedCensus	WeekendDays	HolidayDays	UniqueDepts	BUR	Quarter
0	2022-1	74	54	72.0	3	0	3	1.333333	1
1	2022-10	101	50	98.0	49	0	2	1.960000	1
2	2022-11	176	79	174.0	78	0	3	2.202532	1
3	2022-12	203	79	199.0	62	53	3	2.518987	1
4	2022-13	50	49	49.0	49	0	2	1.000000	2

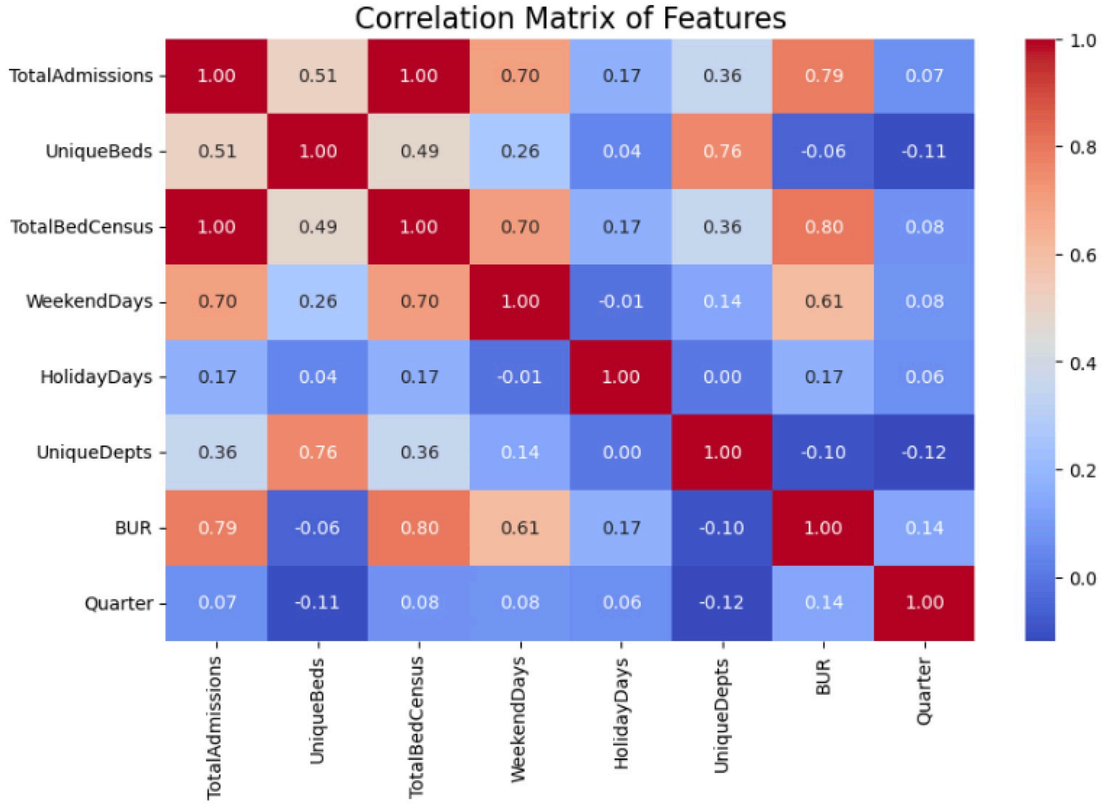
In [16]:
```
# Compare bed utilization across quarters
plt.figure(figsize=(8, 5))
sns.boxplot(data=week_df,
            x='Quarter',
            y='BUR', palette='Set3')
plt.title('Bed Utilization by Quarter', fontsize=16)
plt.xlabel('Quarter', fontsize=12)
plt.ylabel('Utilization Rate', fontsize=12)
plt.show()
```



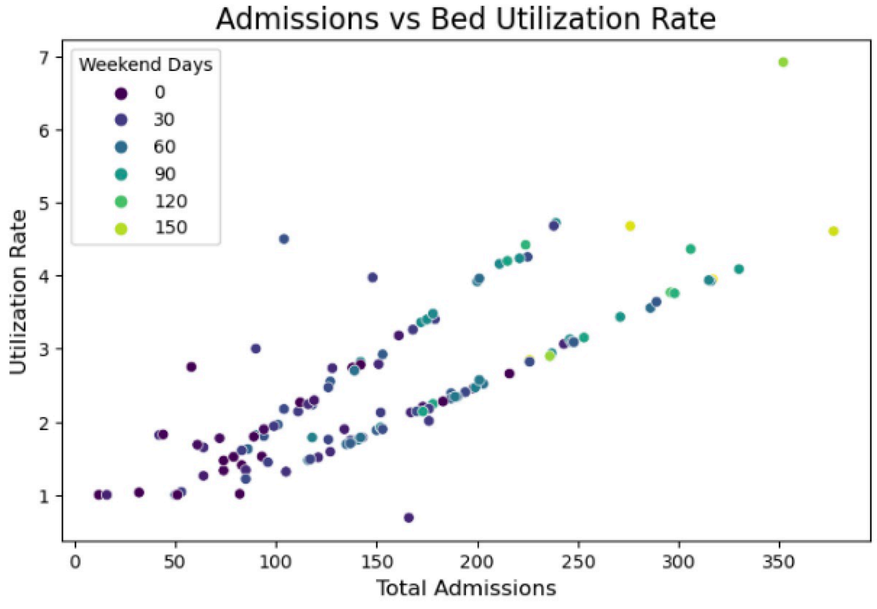
Key Features

In [17]:
```
# Heatmap for correlation analysis
plt.figure(figsize=(10, 6))
correlation_matrix = week_df.corr()
sns.heatmap(correlation_matrix,
            annot=True,
            cmap='coolwarm',
            fmt='.2f')
plt.title('Correlation Matrix of Features', fontsize=16)
plt.show()
```

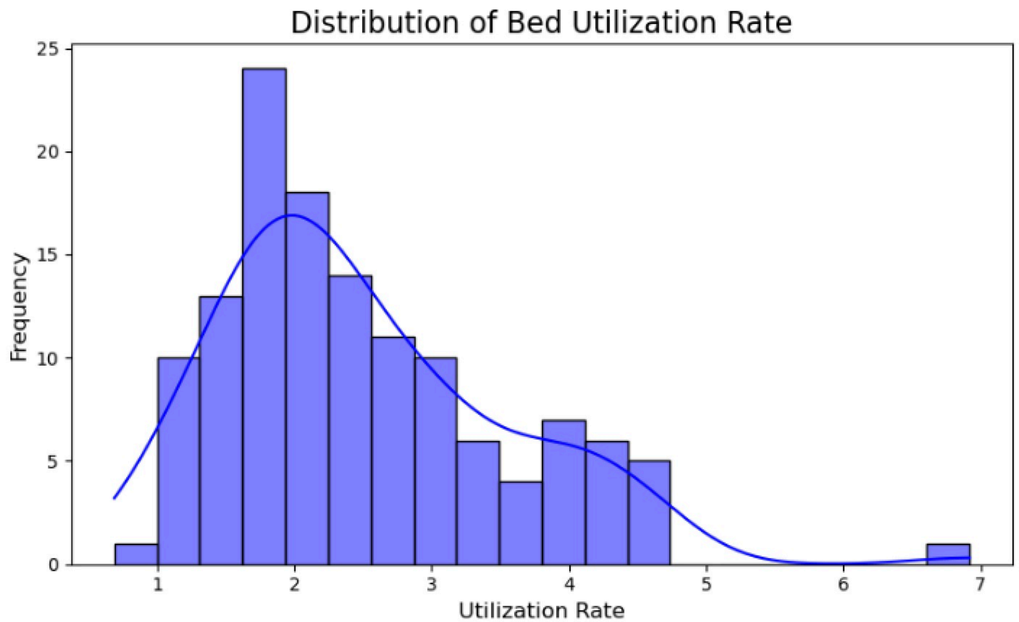
```
C:\Users\local_CAFu\Temp\4\ipykernel_520\2712137047.py:3: FutureWarning: The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or specify the value of numeric_only to silence this warning.
  correlation_matrix = week_df.corr()
```



In [18]:
```
# Scatter plot: Total Admissions vs Bed Utilization Rate
plt.figure(figsize=(8, 5))
sns.scatterplot(data=week_df,
                x='TotalAdmissions',
                y='BUR',
                hue='WeekendDays',
                palette='viridis')
plt.title('Admissions vs Bed Utilization Rate', fontsize=16)
plt.xlabel('Total Admissions', fontsize=12)
plt.ylabel('Utilization Rate', fontsize=12)
plt.legend(title='Weekend Days')
plt.show()
```



In [18]:
```
plt.figure(figsize=(8, 5))
sns.histplot(week_df['BUR'],
             bins=20,
             kde=True,
             color='blue')
plt.title('Distribution of Bed Utilization Rate', fontsize=16)
plt.xlabel('Utilization Rate', fontsize=12)
plt.ylabel('Frequency', fontsize=12)
plt.tight_layout()
plt.show()
```



Saving Revised Data

In [21]:
```
# save revised data
df.to_csv('OB revised.csv', index=False)

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

Loading Revised Data

In [2]:
```
df = pd.read_csv("weekdf.csv")
df.head()
```

Out[2]:

	WeekYear	TotalAdmissions	UniqueBeds	TotalBedCensus	WeekendDays	HolidayDays	UniqueDepts	BUR	Quarter
0	2022-1	74	54	72.0	3	0	3	1.333333	1
1	2022-10	101	50	98.0	49	0	2	1.960000	1
2	2022-11	176	79	174.0	78	0	3	2.202532	1
3	2022-12	203	79	199.0	62	53	3	2.518987	1
4	2022-13	50	49	49.0	49	0	2	1.000000	2

In [3]:
```
df.shape
```

Out[3]:
```
(130, 9)
```

Analysis

In [5]:
```
# define features and target
# Use only TotalAdmissions or TotalBedCensus (since perfect correlation)
X = df[["TotalBedCensus",
        "UniqueBeds",
        "WeekendDays",
        "HolidayDays",
        "UniqueDepts"]]
y = df["BUR"]
```

In [6]:
```
# train test split 80-20
X_tr, X_t, y_tr, y_t = train_test_split(X, y, test_size=0.2, random_state=42)
```

In [7]:
```
print("Training set size:", X_tr.shape)
print("Test set size:", X_t.shape)
```

```
Training set size: (104, 5)
Test set size: (26, 5)
```

Random Forest Regressor

In [10]:
```
# initialize model
rfrm = RandomForestRegressor(n_estimators = 100, random_state = 42)
```

In [11]:
```
# train model
rfrm.fit(X_tr, y_tr)
```

Out[11]:
```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

In [12]:
```
# predict
ypred = rfrm.predict(X_t)
```

In [14]:
```
# model performance
mae = mean_absolute_error(y_t, ypred)
mse = mean_squared_error(y_t, ypred)
r2 = r2_score(y_t, ypred)

print(f"Mean Absolute Error (MAE): {mae:.4f}\nMean Squared Error (MSE): {mse:.4f}\nR-Squared (R²): {r2:.4f}")
```

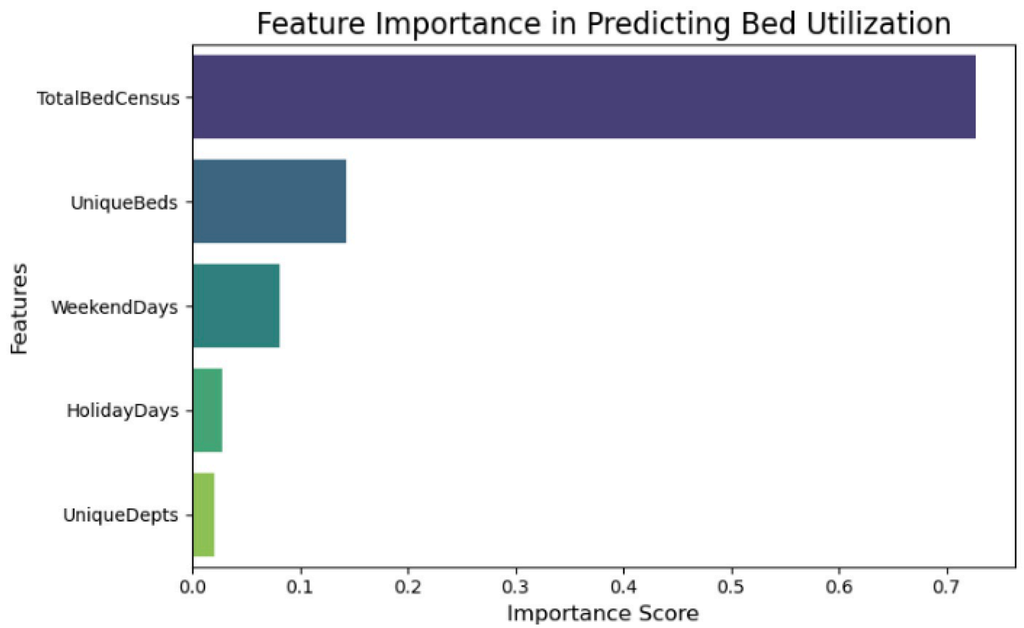
```
Mean Absolute Error (MAE): 0.1779
Mean Squared Error (MSE): 0.0627
R-Squared (R²): 0.8913
```

NOTES:

- MAE: Random Forest Regressor Model's predicted bed utilization rate (BUR) differs from actual rate by 5.2%
- MSE: Random Forest Regressor Model avoids extreme deviations
- R^2: 89% of variability explained by model

In [16]:
```
# feature importance
# Plot feature importances
importances = rfrm.feature_importances_
features = X.columns
```

In [17]:
```
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:

- TotalBedCensus accounts for 70% of predictive power, UniqueBeds ~15%, Weekends for 10%, o/w <5%
- number of beds occupied affect bed utilization rate the most; all other factors less important

Improving Performance

In [20]:
```
# 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
}
```

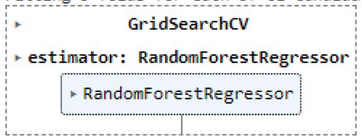
In [21]:
```
# Initialize the Random Forest model
rf = RandomForestRegressor(random_state=42)
```

In [22]:
```
# 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 [23]:
```
# 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[23]:



In [24]:
```
# 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': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 50}
Best Cross-Validated MSE: 0.2471
```

In [25]:
```
# 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[25]:

```
RandomForestRegressor
RandomForestRegressor(max_depth=20, n_estimators=50, random_state=42)
```

In [27]:
```
# 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: 0.1731
Tuned Model RMSE: 0.2320
Tuned Model R²: 0.9067
```

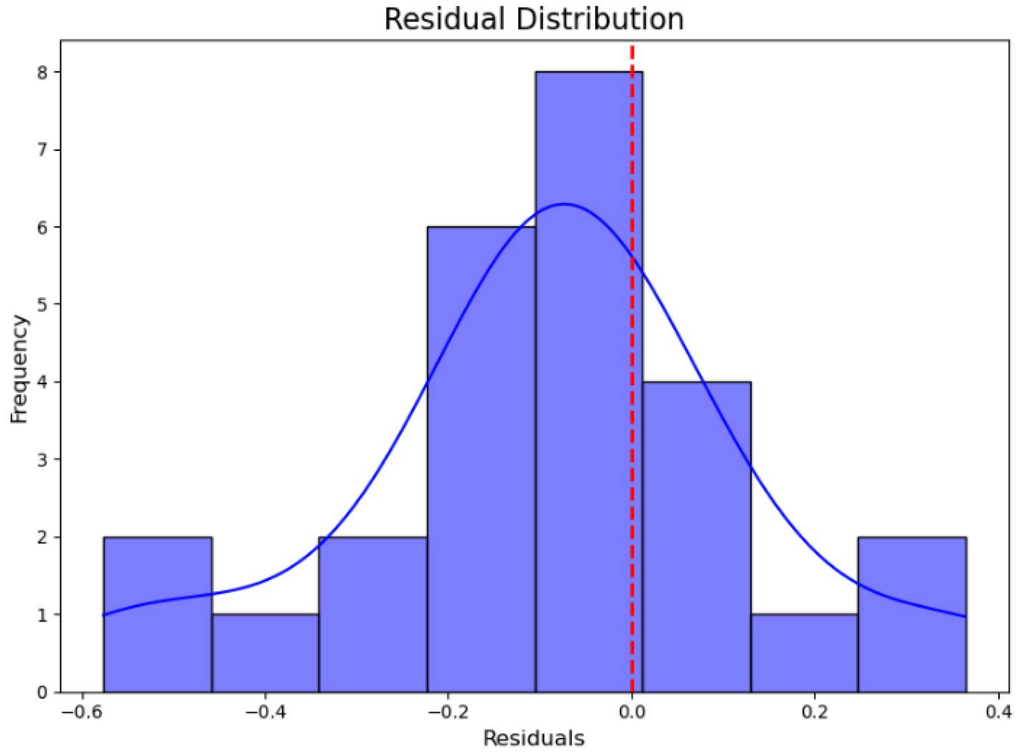
In [ ]:
```
# feature importance
# Plot feature importances
importances = best_rf_model.feature_importances_
features = X.columns
```

In [ ]:
```
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()
```

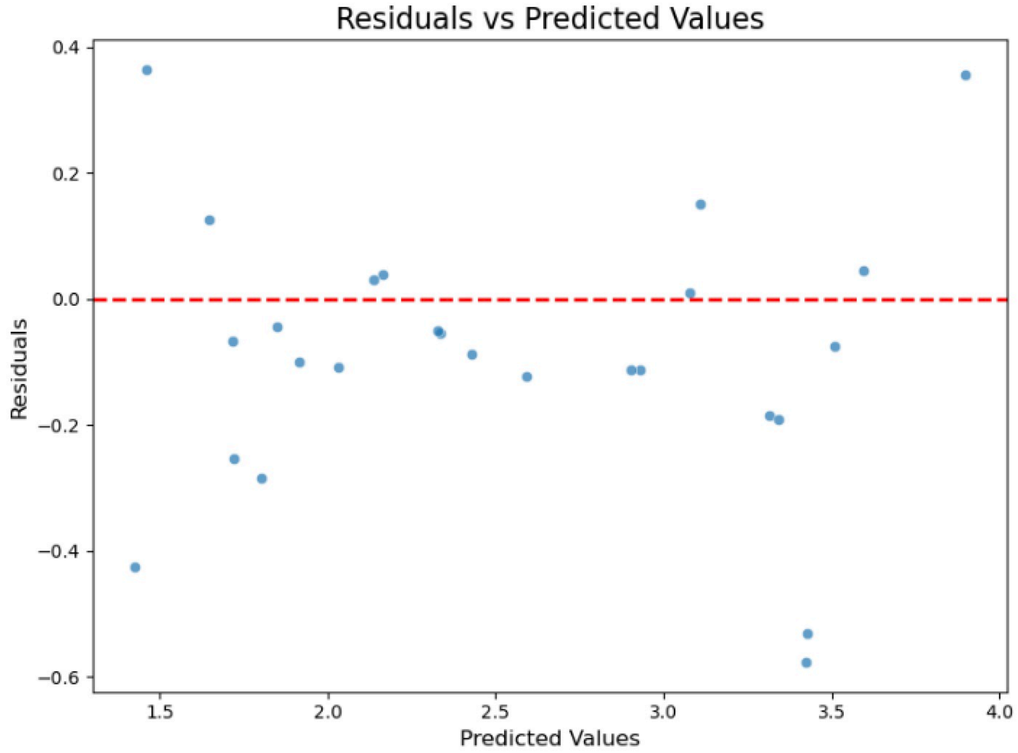
Diagnostics

In [29]:
```
# Calculate residuals
residuals = y_t - y_pred_tuned
```

In [30]:
```
# 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 [38]:
```
# 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 [39]:
```
# Residuals Standard Deviation
residuals_std = np.std(residuals)
print(f"Residuals Standard Deviation: {residuals_std:.4f}")
```

```
Residuals Standard Deviation: 0.2153
```

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

In [36]:
```
xgb_model = XGBRegressor(n_estimators=200,
                         max_depth=6,
                         learning_rate=0.1,
                         random_state=42)
xgb_model.fit(X_tr, y_tr)
```

Out[36]:

```
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 [37]:

```
# 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: 0.9091
```

Gradient Boosting Regressor (Gradient Boosting Machines)

In [4]:

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

In [8]:

```
# Fit the model
gbm.fit(X_tr, y_tr)
```

Out[8]:

```
GradientBoostingRegressor(max_depth=6, random_state=42)
```

In [9]:

```
# Predictions
ypred = gbm.predict(X_t)
```

In [11]:

```
# 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: 0.0975
Gradient Boosting RMSE: 0.1771
Gradient Boosting R²: 0.9456
```

Improving Performance

In [12]:
```
# Define the GradientBoostingRegressor
gbm = GradientBoostingRegressor(random_state=42)
```

In [13]:
```
# 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 [14]:
```
# 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 [15]:
```
# Fit the grid search
grid_search.fit(X_tr, y_tr)
```

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

Out[15]:

GridSearchCV

estimator: GradientBoostingRegressor

GradientBoostingRegressor

In [16]:
```
# Retrieve the best parameters
best_params = grid_search.best_params_
print("Best Parameters for GradientBoostingRegressor:", best_params)
```

```
Best Parameters for GradientBoostingRegressor: {'learning_rate': 0.05, 'max_depth': 3, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 300, 'subsample': 1.0}
```

In [17]:
```
# 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[17]:

GradientBoostingRegressor

```
GradientBoostingRegressor(learning_rate=0.05, min_samples_split=5,
                          n_estimators=300, random_state=42)
```

In [18]:
```
# Predict on the test set
y_pred_gbm_best = gbm_best.predict(X_t)
```

In [19]:
```
# 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: 0.1228
Tuned GradientBoostingRegressor RMSE: 0.1897
Tuned GradientBoostingRegressor R²: 0.9376
```

In [21]:
```
# feature importance
# Plot feature importances
importances = gbm_best.feature_importances_
features = X.columns
```

In [22]:
```
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()
```