HealthFlix Machine Learning Project

Project By : PRASAD JADHAV

In [1]:
```python
import pandas as pd
import numpy as np

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

In [2]:
```python
import warnings
warnings.filterwarnings('ignore')
```

In [12]:
```python
# Load Dataset
file_path = 'healthcare_dataset.csv'
df = pd.read_csv(file_path)
pd.set_option('display.max_columns',30)
print(df.shape)
```

```
(55500, 15)
```

In [22]:
```python
df.head()
```

Out[22]:

	Name	Age	Gender	Blood Type	Medical Condition	Date of Admission	Doctor	Hospital	Insurance Provider	Billin Amoun
0	Bobby JacksOn	30	Male	B-	Cancer	2024-01-31	Matthew Smith	Sons and Miller	Blue Cross	18856.28130
1	LesLie TErRy	62	Male	A+	Obesity	2019-08-20	Samantha Davies	Kim Inc	Medicare	33643.32728
2	DaNnY sMitH	76	Female	A-	Obesity	2022-09-22	Tiffany Mitchell	Cook PLC	Aetna	27955.09607
3	andrEw waTtS	28	Female	O+	Diabetes	2020-11-18	Kevin Wells	Hernandez Rogers and Vang,	Medicare	37909.78241
4	adrIENNE bEll	43	Female	AB+	Cancer	2022-09-19	Kathleen Hanna	White-White	Aetna	14238.31781

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

Out[5]:

	Name	Age	Gender	Blood Type	Medical Condition	Date of Admission	Doctor	Hospital	Insuranc Provide
55495	eLIZABeTH jaCkSOn	42	Female	O+	Asthma	2020-08-16	Joshua Jarvis	Jones-Thompson	Blue Cros
55496	kYle pEREz	61	Female	AB-	Obesity	2020-01-23	Taylor Sullivan	Tucker-Moyer	Cign
55497	HEAther WaNG	38	Female	B+	Hypertension	2020-07-13	Joe Jacobs DVM	and Mahoney Johnson Vasquez,	UnitedHealthcar
55498	JENniFER JOneS	43	Male	O-	Arthritis	2019-05-25	Kimberly Curry	Jackson Todd and Castro,	Medicar
55499	jAMES GARCiA	53	Female	O+	Arthritis	2024-04-02	Dennis Warren	Henry Sons and	Aetn

Data Exploration

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55500 entries, 0 to 55499
Data columns (total 15 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Name                55500 non-null  object 
 1   Age                 55500 non-null  int64  
 2   Gender              55500 non-null  object 
 3   Blood Type          55500 non-null  object 
 4   Medical Condition   55500 non-null  object 
 5   Date of Admission   55500 non-null  object 
 6   Doctor              55500 non-null  object 
 7   Hospital            55500 non-null  object 
 8   Insurance Provider  55500 non-null  object 
 9   Billing Amount      55500 non-null  float64
 10  Room Number         55500 non-null  int64  
 11  Admission Type      55500 non-null  object 
 12  Discharge Date      55500 non-null  object 
 13  Medication          55500 non-null  object 
 14  Test Results        55500 non-null  object 
dtypes: float64(1), int64(2), object(12)
memory usage: 6.4+ MB
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Name                  0
Age                   0
Gender                0
Blood Type            0
Medical Condition     0
Date of Admission     0
Doctor                0
Hospital              0
Insurance Provider    0
Billing Amount        0
Room Number           0
Admission Type        0
Discharge Date        0
Medication            0
Test Results          0
dtype: int64
```

In [13]:
```
df.duplicated().sum()
```

Out[13]:
```
534
```

In [5]:
```
# df = df.drop_duplicates()
```

In [12]:
```
df.describe()
```

Out[12]:

	Age	Billing Amount	Room Number
count	54966.000000	54966.000000	54966.000000
mean	51.535185	25544.306284	301.124404
std	19.605661	14208.409711	115.223143
min	13.000000	-2008.492140	101.000000
25%	35.000000	13243.718641	202.000000
50%	52.000000	25542.749145	302.000000
75%	68.000000	37819.858159	401.000000
max	89.000000	52764.276736	500.000000

In [13]:

```
cat_cols = [x for x in df.columns if df[x].dtypes != 'float64']

for col in cat_cols:
    print(f"Value counts for column '{col}':")
    print(df[col].value_counts())
    print("\n" + "_"*40 + "\n")
```

```
Value counts for column 'Name':
Name
DAvId muNoZ          3
kaTheRIne WeBSTer    2
mICHael aNdERSon     2
DaVID caLhouN        2
MELiSsA COloN        2
                    ..
dUstin blaCKwELl     1
MARc CLaRK           1
sTEphen AyaLa        1
ThOMaS torreS        1
HARoLD ACOSTa        1
Name: count, Length: 49992, dtype: int64
```

```
Value counts for column 'Age':
Age
38    890
57    881
37    880
34    858
80    855
     ...
88     25
16     24
14     18
13     14
89      8
Name: count, Length: 77, dtype: int64
```

```
Value counts for column 'Gender':
Gender
Male      27496
Female    27470
Name: count, dtype: int64
```

```
Value counts for column 'Blood Type':
Blood Type
A-     6898
A+     6896
B+     6885
AB+    6882
AB-    6874
B-     6872
O+     6855
O-     6804
Name: count, dtype: int64
```

```
Value counts for column 'Medical Condition':
Medical Condition
Arthritis       9218
Diabetes        9216
Hypertension    9151
Obesity         9146
Cancer          9140
Asthma          9095
Name: count, dtype: int64
```

____

```
Value counts for column 'Date of Admission':
Date of Admission
2024-03-16    50
2020-10-22    49
2021-12-28    48
2023-08-10    47
2022-07-24    47
              ..
2022-05-28    14
2023-04-12    14
2022-05-23    13
2019-07-22    13
2022-02-05    12
Name: count, Length: 1827, dtype: int64
```

____

```
Value counts for column 'Doctor':
Doctor
Michael Smith      27
John Smith         22
Robert Smith       21
James Smith        20
Michael Johnson    20
                   ..
Shane Tate          1
Christy Parker      1
Larry Miller        1
Chelsea Neal        1
Jeffrey Moore       1
Name: count, Length: 40341, dtype: int64
```

____

```
Value counts for column 'Hospital':
Hospital
LLC Smith                        44
Ltd Smith                        39
Johnson PLC                      37
Smith Ltd                        37
Smith Group                      36
                                 ..
PLC Navarro                       1
PLC Mcintosh                      1
```

```
and Hernandez, Hughes Walton     1
Myers-Williams                   1
Moreno Murphy, Griffith and      1
Name: count, Length: 39876, dtype: int64
```

```
Value counts for column 'Insurance Provider':
Insurance Provider
Cigna               11139
Medicare            11039
UnitedHealthcare    11014
Blue Cross          10952
Aetna               10822
Name: count, dtype: int64
```

```
Value counts for column 'Room Number':
Room Number
393    176
104    174
420    174
491    173
209    170
      ...
189    112
257    111
381    110
254    108
398    108
Name: count, Length: 400, dtype: int64
```

```
Value counts for column 'Admission Type':
Admission Type
Elective     18473
Urgent       18391
Emergency    18102
Name: count, dtype: int64
```

```
Value counts for column 'Discharge Date':
Discharge Date
2020-03-15    53
2021-12-13    51
2023-04-29    51
2020-12-02    50
2020-08-11    50
              ..
2024-06-04     2
2024-06-05     2
2019-05-11     2
2019-05-09     1
```

```
2024-06-06     1
Name: count, Length: 1856, dtype: int64

________________________________________

Value counts for column 'Medication':
Medication
Lipitor        11038
Ibuprofen      11023
Aspirin        10984
Paracetamol    10965
Penicillin     10956
Name: count, dtype: int64

________________________________________

Value counts for column 'Test Results':
Test Results
Abnormal        18437
Normal          18331
Inconclusive    18198
Name: count, dtype: int64

________________________________________
```

In [14]:

```
num_cols = [x for x in df.columns if df[x].dtypes == 'float64']

for col in num_cols:
    print(f"Value counts for column '{col}':")
    print(df[col].value_counts())
    print("\n" + "_"*40 + "\n")
```

```
Value counts for column 'Billing Amount':
Billing Amount
8926.285937     2
8693.755844     2
17889.765079    2
30679.871088    2
1709.059684     2
               ..
46506.415756    1
5343.806298     1
17180.108948    1
47078.702712    1
40116.177618    1
Name: count, Length: 50000, dtype: int64

________________________________________
```

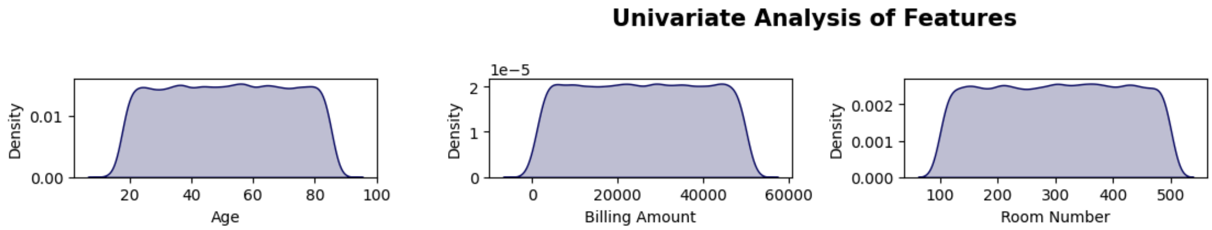
In [15]:

```
num_features = df.select_dtypes(include = ['int64', 'float64']).dtypes.index
```

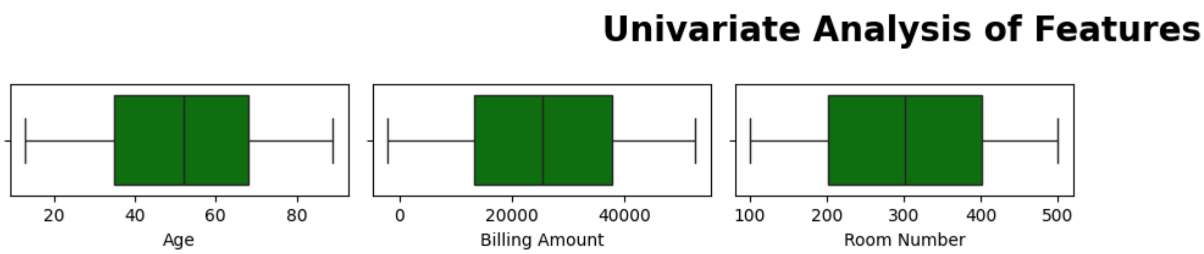
In [16]:

```
plt.figure(figsize=(15,15))
plt.suptitle('Univariate Analysis of Features',fontweight='bold',fontsize=15
```

```python
for i in range(0,len(num_features)):
    plt.subplot(10,4,i+1)
    sns.kdeplot(x=df[num_features[i]],shade=True,color='#03045E')
    plt.tight_layout()
```



In [17]:
```python
plt.figure(figsize = (15,15))
plt.suptitle('Univariate Analysis of Features',fontweight='bold',fontsize=20

for i in range(0,len(num_features)):
    plt.subplot(10,5,i+1)
    sns.boxplot(data=df,x=num_features[i],color='#008000')
    plt.xlabel(num_features[i])
    plt.tight_layout()
```



Feature Engineering & Preprocessing

In [6]:
```python
# Drop irrelevant columns
df = df.drop(columns=['Name', 'Doctor', 'Hospital', 'Date of Admission', 'Di
```

In [20]:
```python
from sklearn.preprocessing import LabelEncoder
```

In [21]:
```python
# Encode categorical variables
label_encoders = {}
for column in df.select_dtypes(include='object').columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le
```

Classification: Predict Medical Condition

- Goal: Predict a patient' s medical condition based on features like age, gender, blood type, test results, etc.
- ML Approach: Supervised Learning (Classification)

In [23]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score, confusion
from imblearn.over_sampling import SMOTE
from lightgbm import LGBMClassifier
```

In [24]:
```
# Split features and target
X = df.drop('Medical Condition', axis=1)
y = df['Medical Condition']
```

In [25]:
```
# Train/test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran
```

In [26]:
```
# Balance with SMOTE
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)
```

In [27]:
```
# Train LightGBM model
model = LGBMClassifier(random_state=42)
model.fit(X_res, y_res)
```

```
[LightGBM] [Warning] Found whitespace in feature_names, replace with underlin
es
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of tes
ting was 0.003825 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 359
[LightGBM] [Info] Number of data points in the train set: 44364, number of us
ed features: 8
[LightGBM] [Info] Start training from score -1.791759
[LightGBM] [Info] Start training from score -1.791759
[LightGBM] [Info] Start training from score -1.791759
[LightGBM] [Info] Start training from score -1.791759
[LightGBM] [Info] Start training from score -1.791759
[LightGBM] [Info] Start training from score -1.791759
```

Out[27]:

LGBMClassifier

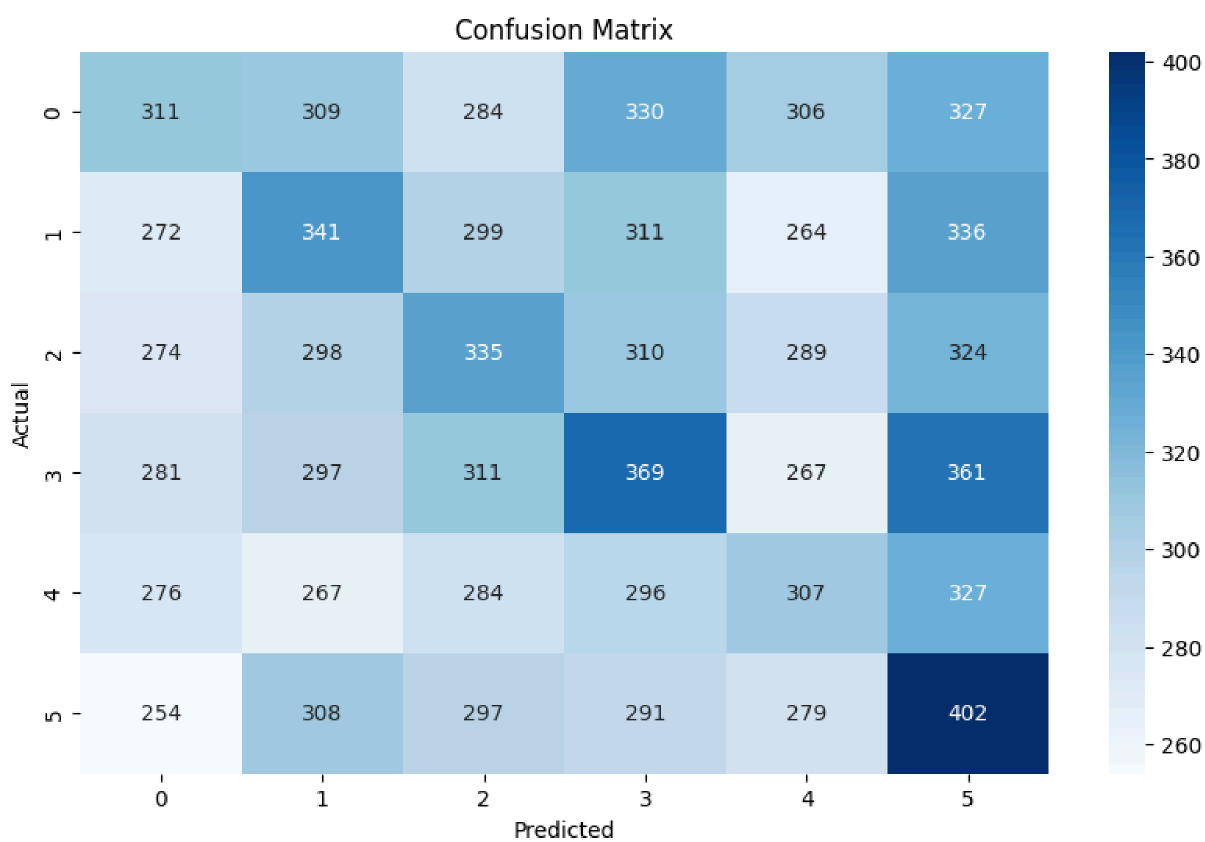
```
LGBMClassifier(random_state=42)
```

In [28]:
```
# Predictions
y_pred = model.predict(X_test)
```

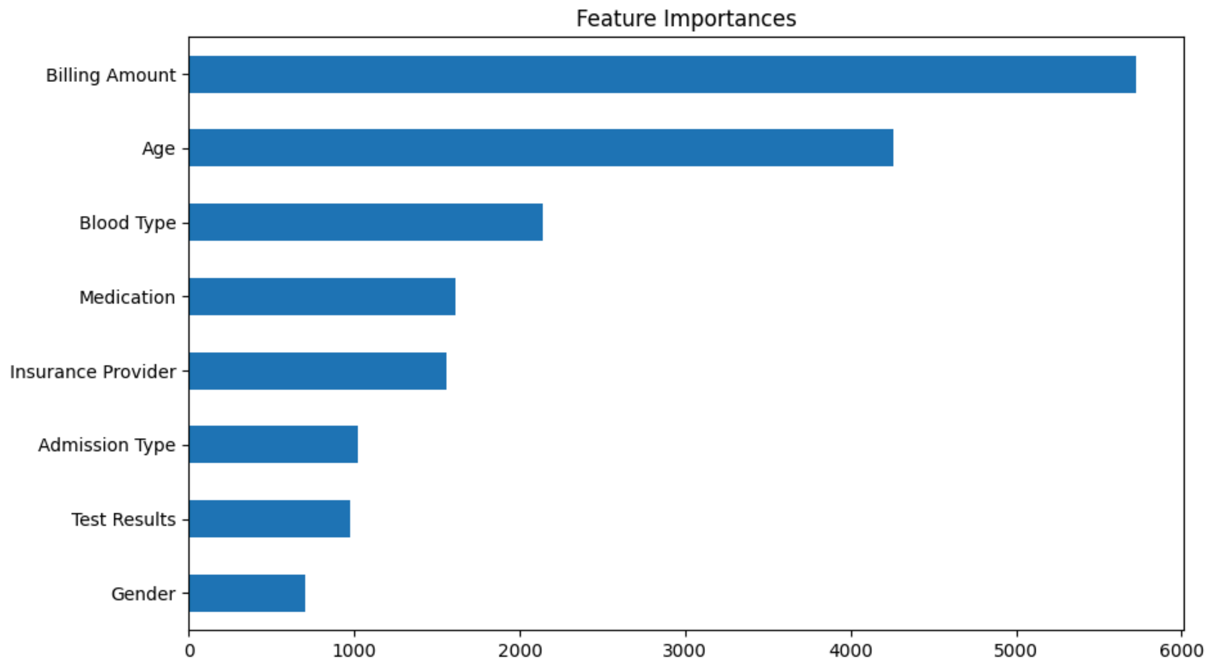
In [29]:
```
# Evaluation
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
conf_matrix = confusion_matrix(y_test, y_pred)
```

```
Accuracy: 0.18782972530471165
Classification Report:
              precision    recall  f1-score   support

           0       0.19      0.17      0.18      1867
           1       0.19      0.19      0.19      1823
           2       0.19      0.18      0.18      1830
           3       0.19      0.20      0.19      1886
           4       0.18      0.17      0.18      1757
           5       0.19      0.22      0.21      1831

    accuracy                           0.19     10994
   macro avg       0.19      0.19      0.19     10994
weighted avg       0.19      0.19      0.19     10994
```

In [30]:
```python
# Confusion matrix heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.title("Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```



In [31]:
```python
# Feature importance
importance = pd.Series(model.feature_importances_, index=X.columns).sort_val
importance.plot(kind='barh', figsize=(10, 6), title="Feature Importances")
plt.gca().invert_yaxis()
plt.show()
```



In [32]:
```python
from sklearn.ensemble import RandomForestClassifier
```

In [33]:
```python
# Features and target
X = df.drop('Medical Condition', axis=1)
y = df['Medical Condition']

# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, ran

# Train model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predictions and evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred, output_dict=True)

# Feature importance
feature_importances = pd.Series(model.feature_importances_, index=X.columns)

accuracy, report, feature_importances
```

Out[33]:
```
(0.27796980171002367,
 {'0': {'precision': 0.2756892230576441,
   'recall': 0.2945902517407606,
   'f1-score': 0.28482651475919213,
   'support': 1867.0},
  '1': {'precision': 0.2679162072767365,
   'recall': 0.2665935271530444,
   'f1-score': 0.2672532306846302,
   'support': 1823.0},
  '2': {'precision': 0.28483491885842194,
   'recall': 0.27814207650273226,
   'f1-score': 0.2814487144042024,
   'support': 1830.0},
  '3': {'precision': 0.2838709677419355,
   'recall': 0.2799575821845175,
   'f1-score': 0.28190069407367857,
   'support': 1886.0},
  '4': {'precision': 0.2778702163061564,
   'recall': 0.28514513375071143,
   'f1-score': 0.28146067415730336,
   'support': 1757.0},
  '5': {'precision': 0.27780979827089336,
   'recall': 0.2632441288913162,
   'f1-score': 0.2703309029725182,
   'support': 1831.0},
  'accuracy': 0.27796980171002367,
  'macro avg': {'precision': 0.27799855525196465,
   'recall': 0.27794545003718035,
   'f1-score': 0.2778701218419208,
   'support': 10994.0},
  'weighted avg': {'precision': 0.2780279500336763,
   'recall': 0.27796980171002367,
   'f1-score': 0.2778962967812628,
   'support': 10994.0}},
 Billing Amount        0.342840
 Age                   0.279717
 Blood Type            0.108606
 Insurance Provider    0.082574
 Medication            0.082214
 Test Results          0.039470
 Admission Type        0.037921
 Gender                0.026659
 dtype: float64)
```

Admission Type Classification (Classification Problem)

- Predict the type of hospital admission using patient details.

Billing Amount Prediction (Regression Problem)

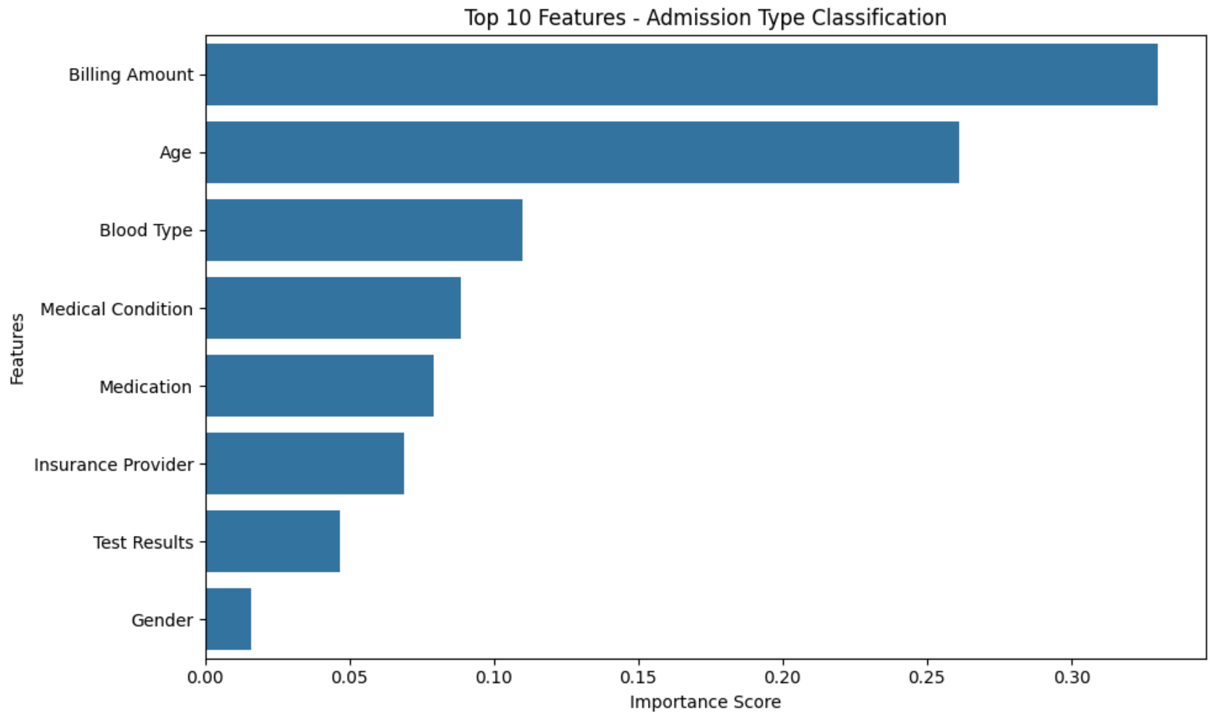
- Predict the billing amount based on patient and treatment features.

In [34]:
```python
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
```

```python
from sklearn.metrics import (
    accuracy_score, classification_report, confusion_matrix,
    mean_absolute_error, mean_squared_error, r2_score
)
```

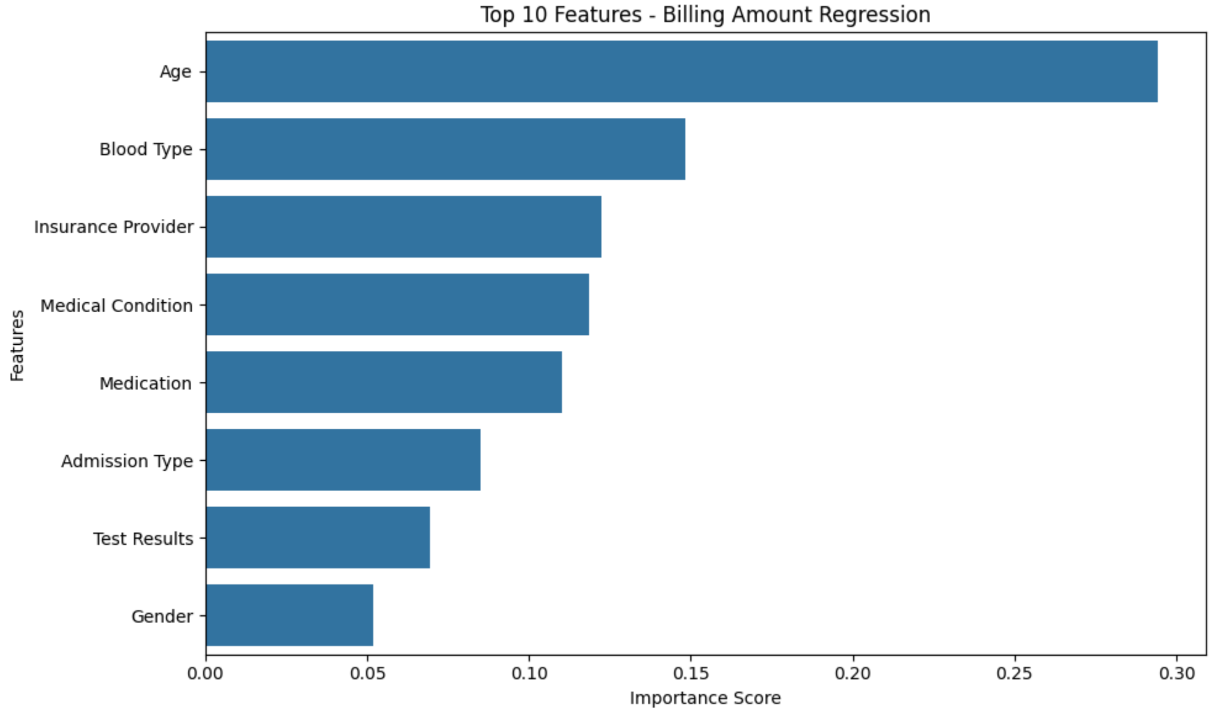
In [35]:
```python
print("\n--- Admission Type Classification ---")

# Features and target
X_class = df.drop(columns=['Admission Type'])
y_class = df['Admission Type']
le_adm = LabelEncoder()
y_class_encoded = le_adm.fit_transform(y_class)

# Train-test split
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(X_class,

# Model training
cls_model = RandomForestClassifier(random_state=42)
cls_model.fit(X_train_cls, y_train_cls)

# Predictions
y_pred_cls = cls_model.predict(X_test_cls)

# Evaluation
print("Accuracy:", accuracy_score(y_test_cls, y_pred_cls))
print("Classification Report:\n", classification_report(y_test_cls, y_pred_c
print("Confusion Matrix:\n", confusion_matrix(y_test_cls, y_pred_cls))

# Feature Importance Plot
importances_cls = pd.Series(cls_model.feature_importances_, index=X_class.co
plt.figure(figsize=(10, 6))
sns.barplot(x=importances_cls[:10], y=importances_cls.index[:10])
plt.title("Top 10 Features - Admission Type Classification")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()
```

```
--- Admission Type Classification ---
Accuracy: 0.4273239949063125
Classification Report:
               precision    recall  f1-score   support

           0       0.43      0.44      0.43      3665
           1       0.43      0.41      0.42      3691
           2       0.42      0.43      0.42      3638

    accuracy                           0.43     10994
   macro avg       0.43      0.43      0.43     10994
weighted avg       0.43      0.43      0.43     10994

Confusion Matrix:
 [[1615 1011 1039]
 [1063 1529 1099]
 [1098  986 1554]]
```



In [36]:
```
print("\n--- Billing Amount Regression ---")

# Features and target
X_reg = df.drop(columns=['Billing Amount'])
y_reg = df['Billing Amount']

# Train-test split
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X_reg, y

# Model training
reg_model = RandomForestRegressor(random_state=42)
reg_model.fit(X_train_reg, y_train_reg)

# Predictions
y_pred_reg = reg_model.predict(X_test_reg)

# Evaluation
print("MAE:", mean_absolute_error(y_test_reg, y_pred_reg))
print("RMSE:", mean_squared_error(y_test_reg, y_pred_reg, squared=False))
print("R^2 Score:", r2_score(y_test_reg, y_pred_reg))

# Feature Importance Plot
importances_reg = pd.Series(reg_model.feature_importances_, index=X_reg.colu
plt.figure(figsize=(10, 6))
sns.barplot(x=importances_reg[:10], y=importances_reg.index[:10])
plt.title("Top 10 Features - Billing Amount Regression")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()
```

```
--- Billing Amount Regression ---
MAE: 12447.875469200017
RMSE: 14583.89163449932
R^2 Score: -0.04429357624767016
```



In [37]:
```python
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
```

In [38]:
```python
# Features and target
X = df.drop(columns=['Admission Type'])
y = df['Admission Type']
le_target = LabelEncoder()
y_encoded = le_target.fit_transform(y)
```

In [39]:
```python
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=
```

In [40]:
```python
# XGBoost model with hyperparameter tuning
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0]
}

grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=3, scori
grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
```

Out[40]:

```
▸ GridSearchCV
▸ best_estimator_: XGBClassifier
  ▸ XGBClassifier
```

In [41]:

```python
# Best model
best_model = grid_search.best_estimator_
```

In [42]:

```python
# Prediction
y_pred = best_model.predict(X_test)
```

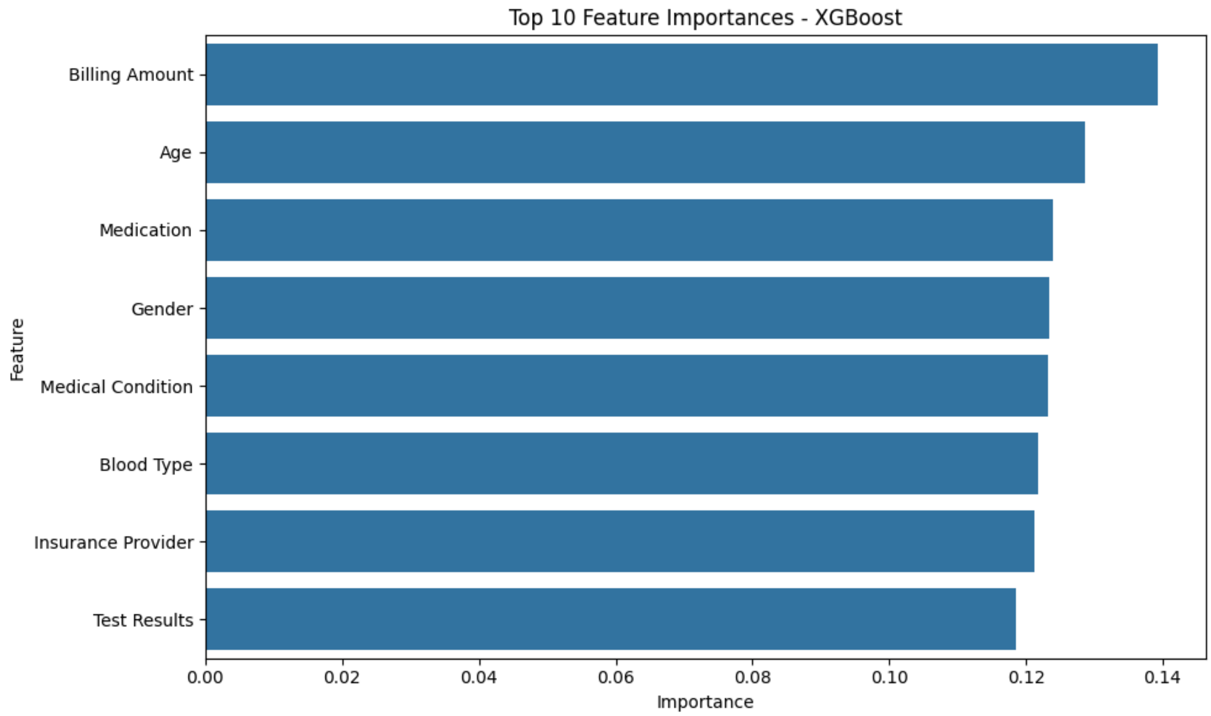
In [43]:

```python
# Evaluation
print("Best Parameters:", grid_search.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Best Parameters: {'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 100,
'subsample': 0.8}
Accuracy: 0.36319810805894126
Classification Report:
               precision    recall  f1-score   support

           0       0.36      0.38      0.37      3665
           1       0.37      0.33      0.35      3691
           2       0.37      0.39      0.38      3638

    accuracy                           0.36     10994
   macro avg       0.36      0.36      0.36     10994
weighted avg       0.36      0.36      0.36     10994

Confusion Matrix:
 [[1375 1083 1207]
 [1268 1216 1207]
 [1208 1028 1402]]
```

In [44]:

```python
# Feature importance plot
importances = pd.Series(best_model.feature_importances_, index=X.columns).so
plt.figure(figsize=(10, 6))
sns.barplot(x=importances[:10], y=importances.index[:10])
plt.title("Top 10 Feature Importances - XGBoost")
plt.xlabel("Importance")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```



In [45]:
```
from sklearn.model_selection import train_test_split, StratifiedKFold
from xgboost import plot_importance
```

In [46]:
```
# Stratified K-Fold for better validation
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# Expanded parameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.05, 0.1],
    'subsample': [0.7, 0.9, 1.0],
    'colsample_bytree': [0.7, 0.9, 1.0],
    'gamma': [0, 1, 5]
}

xgb = XGBClassifier(
    objective='multi:softprob',
    use_label_encoder=False,
    eval_metric='mlogloss',
    random_state=42
)

grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    cv=cv,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
```

In [47]:
```
# Fit with early stopping
grid_search.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 486 candidates, totalling 2430 fits
```

Out[47]:

```
▸ GridSearCV
▸ best_estimator_: XGBClassifier
  ▸ XGBClassifier
```

In [48]:
```
# Best model
best_model = grid_search.best_estimator_
print("Best Parameters:", grid_search.best_params_)
```

```
Best Parameters: {'colsample_bytree': 1.0, 'gamma': 0, 'learning_rate': 0.1,
'max_depth': 7, 'n_estimators': 200, 'subsample': 0.7}
```

In [49]:
```
# Accuracy
y_pred = best_model.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Test Accuracy: 0.3661997453156267
Classification Report:
               precision    recall  f1-score   support

           0       0.36      0.38      0.37      3665
           1       0.38      0.34      0.36      3691
           2       0.37      0.37      0.37      3638

    accuracy                           0.37     10994
   macro avg       0.37      0.37      0.37     10994
weighted avg       0.37      0.37      0.37     10994

Confusion Matrix:
 [[1410 1054 1201]
 [1282 1257 1152]
 [1260 1019 1359]]
```

In [60]:
```
# More Advance Working Sonn..!
```

In [ ]:
```
# Drop irrelevant columns
# df = df.drop(columns=['Name', 'Doctor', 'Hospital', 'Date of Admission', '
```

In [61]:
```
# Apply map() to encode categorical variables
mappings = {}
for col in df.select_dtypes(include='object').columns:
    unique_vals = df[col].unique()
    mapping = {val: idx for idx, val in enumerate(unique_vals)}
    df[col] = df[col].map(mapping)
    mappings[col] = mapping
```

```python
# Features & target
X = df.drop(columns=['Admission Type', 'Insurance Provider', 'Billing Amount
y = df['Admission Type']
target_mapping = {val: idx for idx, val in enumerate(y.unique())}
y = y.map(target_mapping)

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_s
```

In [62]:
```python
# XGBoost model with hyperparameter tuning
xgb = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss')

param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.01, 0.1, 0.2],
    'subsample': [0.8, 1.0]
}

grid_search = GridSearchCV(estimator=xgb, param_grid=param_grid, cv=3, scori
grid_search.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 36 candidates, totalling 108 fits
```

Out[62]:

```
▸ GridSearchCV
▸ best_estimator_: XGBClassifier
  ▸ XGBClassifier
```

In [63]:
```python
# Best model
best_model = grid_search.best_estimator_
```

In [64]:
```python
# Prediction
y_pred = best_model.predict(X_test)
```

In [65]:
```python
# Evaluation
print("Best Parameters:", grid_search.best_params_)
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Best Parameters: {'learning_rate': 0.1, 'max_depth': 7, 'n_estimators': 100,
'subsample': 0.8}
Accuracy: 0.34027651446243407
Classification Report:
               precision    recall  f1-score   support

           0       0.34      0.34      0.34      3678
           1       0.34      0.30      0.32      3621
           2       0.34      0.38      0.36      3695

    accuracy                           0.34     10994
   macro avg       0.34      0.34      0.34     10994
weighted avg       0.34      0.34      0.34     10994

Confusion Matrix:
 [[1253 1086 1339]
 [1188 1085 1348]
 [1258 1034 1403]]
```

In [72]:
```
df['Admission Type'].value_counts()
```

Out[72]:
```
Admission Type
Elective     18655
Urgent       18576
Emergency    18269
Name: count, dtype: int64
```

In [77]:
```
y_test.value_counts()
```

Out[77]:
```
Admission Type
2    3695
0    3678
1    3621
Name: count, dtype: int64
```

In [16]:
```
# Admission Type Classification (XGBoost + Map Encoding)
```

In [57]:
```
from sklearn.model_selection import train_test_split, StratifiedKFold, GridS
from sklearn.metrics import accuracy_score, classification_report, confusion
from xgboost import XGBClassifier, XGBRegressor, plot_importance
from imblearn.over_sampling import SMOTE
```

In [7]:
```
# Map Gender
df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 0})

# Map Blood Type
df['Blood Type'] = df['Blood Type'].map({
    'A-': 0, 'A+': 1, 'B+': 2, 'AB+': 3, 'AB-': 4, 'B-': 5, 'O+': 6, 'O-': 7
})

# Map Medical Condition
df['Medical Condition'] = df['Medical Condition'].map({
    'Arthritis': 0, 'Diabetes': 1, 'Hypertension': 2,
    'Obesity': 3, 'Cancer': 4, 'Asthma': 5
})
```

```
# Map Insurance Provider
df['Insurance Provider'] = df['Insurance Provider'].map({
    'Cigna': 0, 'Medicare': 1, 'UnitedHealthcare': 2,
    'Blue Cross': 3, 'Aetna': 4
})

# Map Medication
df['Medication'] = df['Medication'].map({
    'Lipitor': 0, 'Ibuprofen': 1, 'Aspirin': 2,
    'Paracetamol': 3, 'Penicillin': 4
})

# Map Test Results
df['Test Results'] = df['Test Results'].map({
    'Abnormal': 0, 'Normal': 1, 'Inconclusive': 2
})

# Map target variable Admission Type
df['Admission Type'] = df['Admission Type'].map({
    'Elective': 0, 'Urgent': 1, 'Emergency': 2
})
```

In [38]:

```
# Select relevant features and target
features = ['Age', 'Gender', 'Blood Type', 'Medical Condition', 'Insurance P
target = 'Admission Type'

X = df[features]
y = df[target]
```

In [39]:

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_s
```

In [40]:

```
# Apply SMOTE to balance classes
smote = SMOTE(random_state=42)
X_train_res, y_train_res = smote.fit_resample(X_train, y_train)
```

In [41]:

```
# Define XGBoost classifier
xgb = XGBClassifier(
    objective='multi:softprob',
    use_label_encoder=False,
    eval_metric='mlogloss',
    random_state=42
)

# Hyperparameter grid
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5],
    'learning_rate': [0.05, 0.1],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
    'gamma': [0, 1]
}

# Grid search with cross-validation
```

```python
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(
    estimator=xgb,
    param_grid=param_grid,
    cv=cv,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
```

In [42]:

```python
# Train the model
grid_search.fit(X_train_res, y_train_res)
best_model = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 64 candidates, totalling 320 fits
```

In [43]:

```python
# Predict and evaluate
y_pred = best_model.predict(X_test)
```

In [44]:

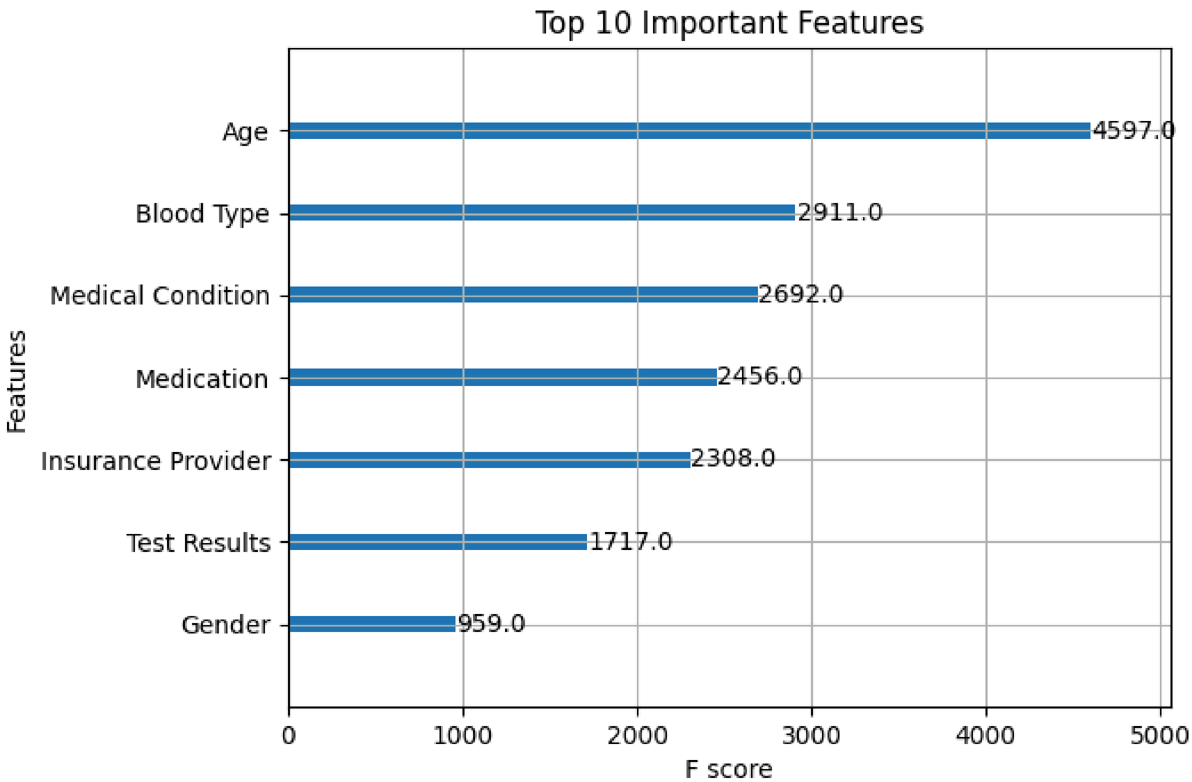
```python
print("\nBest Parameters:", grid_search.best_params_)
print("Test Accuracy:", accuracy_score(y_test, y_pred))
print("Classification Report:\n", classification_report(y_test, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Best Parameters: {'colsample_bytree': 0.8, 'gamma': 0, 'learning_rate': 0.1,
'max_depth': 5, 'n_estimators': 200, 'subsample': 0.8}
Test Accuracy: 0.34300527560487537
Classification Report:
               precision    recall  f1-score   support

           0       0.35      0.34      0.35      3695
           1       0.34      0.35      0.34      3678
           2       0.34      0.34      0.34      3621

    accuracy                           0.34     10994
   macro avg       0.34      0.34      0.34     10994
weighted avg       0.34      0.34      0.34     10994

Confusion Matrix:
 [[1261 1282 1152]
 [1185 1277 1216]
 [1156 1232 1233]]
```

In [45]:

```python
# Plot Feature Importance
plt.figure(figsize=(10, 6))
plot_importance(best_model, max_num_features=10)
plt.title("Top 10 Important Features")
plt.show()
```

```
<Figure size 1000x600 with 0 Axes>
```



In [47]:

```
import numpy as np
# Assuming `best_model` is already loaded or trained

print('--- Enter the Petition Details ---')

# Taking inputs for features
age = int(input('Enter the Age of the Patient: '))
gender = int(input('Enter the Gender (0: Female, 1: Male): '))
blood_type = int(input('Enter the Blood Type (0: A, 1: B, 2: AB, 3: O): '))
medical_condition = float(input('Enter the Medical Condition Score (e.g., 0.
medication = int(input('Enter Medication Code (e.g., 0 or 1): '))
test_results = int(input('Enter the Test Results Score (e.g., 0 or 1): '))
insurance_provider = int(input('Enter the Insurance Provider Code: '))
# billing_amount = float(input('Enter the Billing Amount (in USD): '))

# Preparing input for the model
input_point = np.array([[age, gender, blood_type, medical_condition, medicat
                         test_results, insurance_provider]]) # billing_amoun

# Making prediction
prediction = best_model.predict(input_point)

# Mapping prediction to human-readable label
label_mapping = {0: 'Elective', 1: 'Urgent', 2: 'Emergency'}
predicted_label = label_mapping.get(prediction[0], "Unknown")

# Printing result
print(f'\nPrediction Result: {predicted_label} Admission')
```

```
--- Enter the Petition Details ---
Enter the Age of the Patient: 30
Enter the Gender (0: Female, 1: Male): 1
Enter the Blood Type (0: A, 1: B, 2: AB, 3: O): 5
Enter the Medical Condition Score (e.g., 0.0 to 1.0): 3
Enter Medication Code (e.g., 0 or 1): 3
Enter the Test Results Score (e.g., 0 or 1): 1
Enter the Insurance Provider Code: 2

Prediction Result: Emergency Admission
```

In [48]:
```
# Features and Targets
features = ['Age', 'Gender', 'Blood Type', 'Medical Condition', 'Insurance P
X = df[features]
y_class = df['Admission Type']
y_reg = df['Billing Amount']
```

In [49]:
```
# Split for classification
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
```

In [50]:
```
# Apply SMOTE to balance Admission Type
smote = SMOTE(random_state=42)
X_train_class_res, y_train_class_res = smote.fit_resample(X_train_class, y_t
```

In [52]:
```
# XGBoost Classifier
xgb_clf = XGBClassifier(
    objective='multi:softmax',
    num_class=3,
    eval_metric='mlogloss',
    use_label_encoder=False,
    random_state=42
)
xgb_clf.fit(X_train_class_res, y_train_class_res)
y_pred_class = xgb_clf.predict(X_test_class)
```

In [53]:
```
# Evaluate classifier
print("📌 Admission Type Classification")
print("Accuracy:", accuracy_score(y_test_class, y_pred_class))
print("Classification Report:\n", classification_report(y_test_class, y_pred
print("Confusion Matrix:\n", confusion_matrix(y_test_class, y_pred_class))
```

```
📌 Admission Type Classification
Accuracy: 0.3369110423867564
Classification Report:
              precision    recall  f1-score   support

           0       0.34      0.34      0.34      3695
           1       0.34      0.33      0.34      3678
           2       0.33      0.34      0.34      3621

    accuracy                           0.34     10994
   macro avg       0.34      0.34      0.34     10994
weighted avg       0.34      0.34      0.34     10994

Confusion Matrix:
 [[1242 1213 1240]
 [1229 1232 1217]
 [1225 1166 1230]]
```

In [54]:

```
# Split for regression
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_reg
```

In [55]:

```
# XGBoost Regressor
xgb_reg = XGBRegressor(
    objective='reg:squarederror',
    n_estimators=100,
    learning_rate=0.1,
    max_depth=5,
    random_state=42
)
xgb_reg.fit(X_train_reg, y_train_reg)
y_pred_reg = xgb_reg.predict(X_test_reg)
```

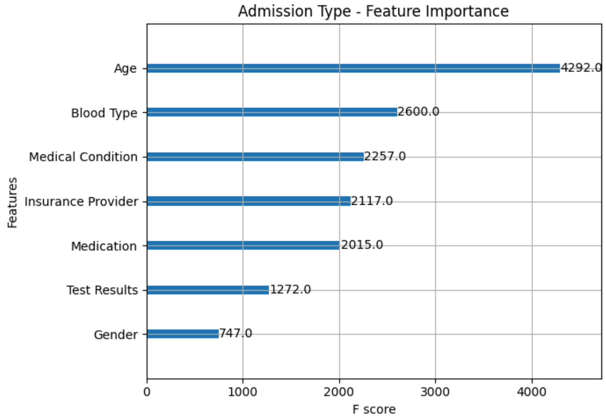
In [58]:

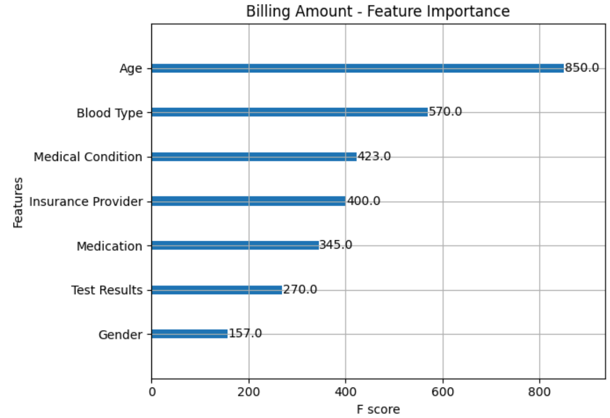
```
# Evaluate regression
print("\n📌 Billing Amount Prediction")
print("MAE:", mean_absolute_error(y_test_reg, y_pred_reg))
print("RMSE:", np.sqrt(mean_squared_error(y_test_reg, y_pred_reg)))
print("R² Score:", r2_score(y_test_reg, y_pred_reg))
```

```
📌 Billing Amount Prediction
MAE: 12408.12414265777
RMSE: 14332.809098201316
R² Score: -0.00864509790921808
```

In [59]:

```
# Plot feature importances for both models
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)
plot_importance(xgb_clf, ax=plt.gca(), title='Admission Type - Feature Impor

plt.subplot(1, 2, 2)
plot_importance(xgb_reg, ax=plt.gca(), title='Billing Amount - Feature Impor

plt.tight_layout()
plt.show()
```





In [ ]:

```
# Classification + Regression + Input Prediction
```

In [62]:

```
# Features and targets
features = ['Age', 'Gender', 'Medical Condition', 'Insurance Provider', 'Med
X = df[features]
y_class = df['Admission Type']
y_reg = df['Billing Amount']

# SMOTE for classification
smote = SMOTE(random_state=42)
X_smote, y_class_smote = smote.fit_resample(X, y_class)

# Split data
X_train_class, X_test_class, y_train_class, y_test_class = train_test_split(
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_reg

# XGBoost Classifier with hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [3, 5, 7],
    'learning_rate': [0.05, 0.1, 0.2]
}
grid_search = GridSearchCV(XGBClassifier(random_state=42, use_label_encoder=
grid_search.fit(X_train_class, y_train_class)
best_model_class = grid_search.best_estimator_

# Evaluate classifier
y_pred_class = best_model_class.predict(X_test_class)
print("\n--- Classification Report (Admission Type) ---")
print(classification_report(y_test_class, y_pred_class))

# XGBoost Regressor for billing amount
best_model_reg = XGBRegressor(n_estimators=100, max_depth=5, learning_rate=0
best_model_reg.fit(X_train_reg, y_train_reg)

# Evaluate regressor
y_pred_reg = best_model_reg.predict(X_test_reg)
rmse = np.sqrt(mean_squared_error(y_test_reg, y_pred_reg))
print(f"\n--- Regression RMSE (Billing Amount): ${rmse:.2f}")
```

```
--- Classification Report (Admission Type) ---
              precision    recall  f1-score   support

           0       0.34      0.34      0.34      3703
           1       0.33      0.35      0.34      3626
           2       0.35      0.34      0.34      3755

    accuracy                           0.34     11084
   macro avg       0.34      0.34      0.34     11084
weighted avg       0.34      0.34      0.34     11084


--- Regression RMSE (Billing Amount): $14321.94
```

In [63]:

```python
# Prediction Code (User Input + Output)
# 🔮 Prediction section

print('--- Enter Patient Details ---')

age = int(input('Enter the Age of the Patient: '))
gender = int(input('Enter the Gender (0: Female, 1: Male): '))
medical_condition = int(input('Enter Medical Condition Code (0: Arthritis, 1
insurance_provider = int(input('Enter Insurance Provider Code (0: Cigna, 1:
medication = int(input('Enter Medication Code (0: Lipitor, 1: Ibuprofen, 2:
test_results = int(input('Enter Test Results Code (0: Abnormal, 1: Normal, 2

# Prepare input for prediction
input_point = np.array([[age, gender, medical_condition, insurance_provider,

# Predictions
admission_prediction = best_model_class.predict(input_point)
billing_prediction = best_model_reg.predict(input_point)

# Mapping Admission Type
label_mapping = {0: 'Elective', 1: 'Urgent', 2: 'Emergency'}
admission_type = label_mapping.get(admission_prediction[0], "Unknown")

# Final Output
print('\n🔍 Predicted Results:')
print(f'🩺 Admission Type: {admission_type}')
print(f'💰 Estimated Billing Amount: ${billing_prediction[0]:.2f}')
```

```
--- Enter Patient Details ---
Enter the Age of the Patient: 30
Enter the Gender (0: Female, 1: Male): 1
Enter Medical Condition Code (0: Arthritis, 1: Diabetes, 2: Hypertension, 3:
Obesity, 4: Cancer, 5: Asthma): 4
Enter Insurance Provider Code (0: Cigna, 1: Medicare, 2: UnitedHealthcare, 3:
Blue Cross, 4: Aetna): 3
Enter Medication Code (0: Lipitor, 1: Ibuprofen, 2: Aspirin, 3: Paracetamol,
4: Penicillin): 3
Enter Test Results Code (0: Abnormal, 1: Normal, 2: Inconclusive): 1

🔍 Predicted Results:
🩺 Admission Type: Elective
💰 Estimated Billing Amount: $26140.86
```

In [ ]:
```
# Blending Machine Learning Method Apply
```

In [27]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, mean_squared_error, accur
from sklearn.ensemble import GradientBoostingClassifier, RandomForestClassif
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegresso
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier, XGBRegressor
```

In [15]:
```
# Mapping categorical columns
df['Gender'] = df['Gender'].map({'Male': 1, 'Female': 0})
df['Blood Type'] = df['Blood Type'].map({'A-': 0, 'A+': 1, 'B+': 2, 'AB+': 3
df['Medical Condition'] = df['Medical Condition'].map({'Arthritis': 0, 'Diab
df['Insurance Provider'] = df['Insurance Provider'].map({'Cigna': 0, 'Medica
df['Medication'] = df['Medication'].map({'Lipitor': 0, 'Ibuprofen': 1, 'Aspi
df['Test Results'] = df['Test Results'].map({'Abnormal': 0, 'Normal': 1, 'In
df['Admission Type'] = df['Admission Type'].map({'Elective': 0, 'Urgent': 1,
```

In [16]:
```
# Features and targets
features = ['Age', 'Gender', 'Blood Type', 'Medical Condition', 'Insurance P
X = df[features]
y_class = df['Admission Type']
y_reg = df['Billing Amount']
```

In [17]:
```
# Apply SMOTE
smote = SMOTE(random_state=42)
X_smote, y_class_smote = smote.fit_resample(X, y_class)
```

In [18]:
```
# Train/test split
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(X_smote,
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_reg
```

In [19]:
```
# Blending for Classification
clf1 = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random
clf2 = GradientBoostingClassifier(random_state=42)
clf3 = RandomForestClassifier(random_state=42)
voting_clf = VotingClassifier(estimators=[('xgb', clf1), ('gb', clf2), ('rf'
voting_clf.fit(X_train_cls, y_train_cls)
```

Out[19]:

VotingClassifier

xgb	gb	rf
XGBClassifier	GradientBoostingClassifier	RandomForestClass

In [20]:
```
# Blending for Regression
reg1 = XGBRegressor(random_state=42)
reg2 = GradientBoostingRegressor(random_state=42)
reg3 = RandomForestRegressor(random_state=42)
```

In [21]:
```
# Fit individual regressors
reg1.fit(X_train_reg, y_train_reg)
reg2.fit(X_train_reg, y_train_reg)
reg3.fit(X_train_reg, y_train_reg)
```

Out[21]:

RandomForestRegressor

```
RandomForestRegressor(random_state=42)
```

In [22]:
```
# Evaluate classification
y_pred_cls = voting_clf.predict(X_test_cls)
cls_report = classification_report(y_test_cls, y_pred_cls, output_dict=True)
```

In [23]:
```
# Evaluate regression (average ensemble)
y_pred_reg1 = reg1.predict(X_test_reg)
y_pred_reg2 = reg2.predict(X_test_reg)
y_pred_reg3 = reg3.predict(X_test_reg)
y_pred_reg_avg = (y_pred_reg1 + y_pred_reg2 + y_pred_reg3) / 3
reg_rmse = np.sqrt(mean_squared_error(y_test_reg, y_pred_reg_avg))
```

In [24]:
```
cls_report, reg_rmse
```

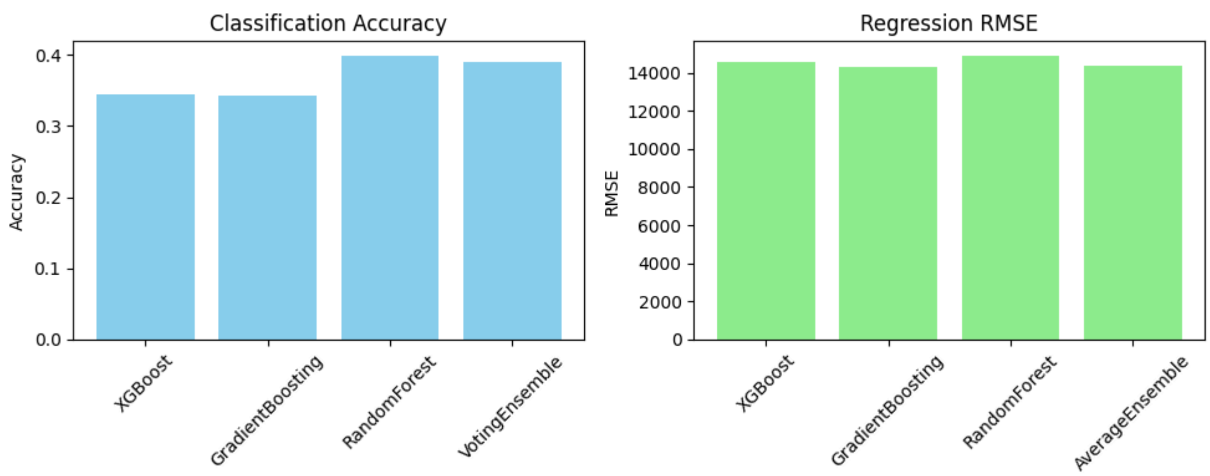
Out[24]:
```
({'0': {'precision': 0.38834688346883467,
   'recall': 0.38698352687010534,
   'f1-score': 0.38766400649262817,
   'support': 3703.0},
  '1': {'precision': 0.3824317086234601,
   'recall': 0.3938223938223938,
   'f1-score': 0.38804347826086955,
   'support': 3626.0},
  '2': {'precision': 0.3983606557377049,
   'recall': 0.3882822902796272,
   'f1-score': 0.39325691166554283,
   'support': 3755.0},
  'accuracy': 0.38966077228437385,
  'macro avg': {'precision': 0.3897130826099999,
   'recall': 0.38969607032404213,
   'f1-score': 0.38965479880634685,
   'support': 11084.0},
  'weighted avg': {'precision': 0.3898042355872287,
   'recall': 0.38966077228437385,
   'f1-score': 0.3896828916925503,
   'support': 11084.0}},
 14367.02594139383)
```

In [28]:
```
# Train/test split
X_train_cls, X_test_cls, y_train_cls, y_test_cls = train_test_split(X_smote,
X_train_reg, X_test_reg, y_train_reg, y_test_reg = train_test_split(X, y_reg

# Classifiers
clf1 = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random
clf2 = GradientBoostingClassifier(random_state=42)
clf3 = RandomForestClassifier(random_state=42)
```

```
voting_clf = VotingClassifier(estimators=[('xgb', clf1), ('gb', clf2), ('rf'

# Train classifiers
clf1.fit(X_train_cls, y_train_cls)
clf2.fit(X_train_cls, y_train_cls)
clf3.fit(X_train_cls, y_train_cls)
voting_clf.fit(X_train_cls, y_train_cls)

# Predict and evaluate classifiers
models_cls = {'XGBoost': clf1, 'GradientBoosting': clf2, 'RandomForest': clf
acc_scores = {}

print("\n--- Classification Accuracy Scores ---")
for name, model in models_cls.items():
    preds = model.predict(X_test_cls)
    acc = accuracy_score(y_test_cls, preds)
    acc_scores[name] = acc
    print(f"{name}: Accuracy = {acc:.4f}")

best_cls_model = max(acc_scores, key=acc_scores.get)
print(f"\nBest Classification Model: {best_cls_model} with Accuracy = {acc_s

# Regressors
reg1 = XGBRegressor(random_state=42)
reg2 = GradientBoostingRegressor(random_state=42)
reg3 = RandomForestRegressor(random_state=42)

# Train regressors
reg1.fit(X_train_reg, y_train_reg)
reg2.fit(X_train_reg, y_train_reg)
reg3.fit(X_train_reg, y_train_reg)

# Predict and evaluate regressors
models_reg = {'XGBoost': reg1, 'GradientBoosting': reg2, 'RandomForest': reg
rmse_scores = {}

print("\n--- Regression RMSE Scores ---")
for name, model in models_reg.items():
    preds = model.predict(X_test_reg)
    rmse = np.sqrt(mean_squared_error(y_test_reg, preds))
    rmse_scores[name] = rmse
    print(f"{name}: RMSE = {rmse:.2f}")

# Average ensemble prediction
y_pred_reg_avg = (reg1.predict(X_test_reg) + reg2.predict(X_test_reg) + reg3
rmse_avg = np.sqrt(mean_squared_error(y_test_reg, y_pred_reg_avg))
rmse_scores['AverageEnsemble'] = rmse_avg
print(f"Average Ensemble RMSE = {rmse_avg:.2f}")

best_reg_model = min(rmse_scores, key=rmse_scores.get)
print(f"\nBest Regression Model: {best_reg_model} with RMSE = {rmse_scores[b

# Optional: Plot scores
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.bar(acc_scores.keys(), acc_scores.values(), color='skyblue')
```

```python
plt.title("Classification Accuracy")
plt.ylabel("Accuracy")
plt.xticks(rotation=45)

plt.subplot(1, 2, 2)
plt.bar(rmse_scores.keys(), rmse_scores.values(), color='lightgreen')
plt.title("Regression RMSE")
plt.ylabel("RMSE")
plt.xticks(rotation=45)

plt.tight_layout()
plt.show()
```

```
--- Classification Accuracy Scores ---
XGBoost: Accuracy = 0.3436
GradientBoosting: Accuracy = 0.3421
RandomForest: Accuracy = 0.3990
VotingEnsemble: Accuracy = 0.3897

Best Classification Model: RandomForest with Accuracy = 0.3990

--- Regression RMSE Scores ---
XGBoost: RMSE = 14554.68
GradientBoosting: RMSE = 14286.19
RandomForest: RMSE = 14908.73
Average Ensemble RMSE = 14367.03

Best Regression Model: GradientBoosting with RMSE = 14286.19
```



In [33]:

```python
# Prediction Code (User Input + Output)
# 🔮 Prediction section

print('--- Enter Patient Details ---')

age = int(input('Enter the Age of the Patient: '))
gender = int(input('Enter the Gender (0: Female, 1: Male): '))
blood_type = int(input('Enter Blood Type Code (0: A-, 1: A+, 2: B+, 3: AB+,
medical_condition = int(input('Enter Medical Condition Code (0: Arthritis, 1
insurance_provider = int(input('Enter Insurance Provider Code (0: Cigna, 1:
medication = int(input('Enter Medication Code (0: Lipitor, 1: Ibuprofen, 2:
test_results = int(input('Enter Test Results Code (0: Abnormal, 1: Normal, 2
```

```python
# Prepare input for prediction
input_point = np.array([[age, gender, blood_type, medical_condition, insuran

# Predictions
admission_prediction = clf3.predict(input_point)
billing_prediction = reg2.predict(input_point)

# Mapping Admission Type
label_mapping = {0: 'Elective', 1: 'Urgent', 2: 'Emergency'}
admission_type = label_mapping.get(admission_prediction[0], "Unknown")

# Final Output
print('\n🔍 Predicted Results:')
print(f'🩺 Admission Type: {admission_type}')
print(f'💰 Estimated Billing Amount: ₹{billing_prediction[0]:.2f}')
```

```
--- Enter Patient Details ---
Enter the Age of the Patient: 30
Enter the Gender (0: Female, 1: Male): 1
Enter Blood Type Code (0: A-, 1: A+, 2: B+, 3: AB+, 4: AB-, 5: B-, 6: O+, 7:
O-): 5
Enter Medical Condition Code (0: Arthritis, 1: Diabetes, 2: Hypertension, 3:
Obesity, 4: Cancer, 5: Asthma): 4
Enter Insurance Provider Code (0: Cigna, 1: Medicare, 2: UnitedHealthcare, 3:
Blue Cross, 4: Aetna): 3
Enter Medication Code (0: Lipitor, 1: Ibuprofen, 2: Aspirin, 3: Paracetamol,
4: Penicillin): 3
Enter Test Results Code (0: Abnormal, 1: Normal, 2: Inconclusive): 1

🔍 Predicted Results:
🩺 Admission Type: Urgent
💰 Estimated Billing Amount: ₹26061.43
```

In [ ]:

```python
# More Advance Working Sonn..!
# Notebook Project By : PRASAD JADHAV (ML-ENG)
# LinkedIn: linkedin.com/in/prasadmjadhav2 | Github: github.com/prasadmjadha
```

Thank You!