```python
#  Healthcare Data Analysis and Prediction
# This script performs data preprocessing, exploratory data analysis
(EDA), and predictive modeling on a healthcare dataset.
import pandas as pd
import numpy as np
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from statsmodels.tsa.arima.model import ARIMA
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score, davies_bouldin_score
from sklearn.metrics import classification_report, accuracy_score,
roc_auc_score, confusion_matrix, mean_absolute_error,
mean_squared_error
from imblearn.over_sampling import SMOTE
import matplotlib.pyplot as plt
import seaborn as sns
import shap

# Load dataset
df = pd.read_csv('healthcare_ds.csv')

# Keep relevant columns only
df = df[['Age', 'Gender', 'Blood Type', 'Medical Condition',
'Admission Type',
         'Test Results', 'Medication', 'Billing Amount', 'Date of
Admission', 'Discharge Date']]

# Convert date columns
df['Date of Admission'] = pd.to_datetime(df['Date of Admission'],
errors='coerce')
df['Discharge Date'] = pd.to_datetime(df['Discharge Date'],
errors='coerce')
df['Length of Stay'] = (df['Discharge Date'] - df['Date of
Admission']).dt.days
df.dropna(inplace=True)

# Dividing diseases into  Critical and Non-Critical
def categorize_condition(condition):
    condition = str(condition).lower()
    critical_keywords = ['cancer', 'tumor', 'stroke', 'cardiac',
'arrest', 'heart', 'failure', 'trauma', 'critical', 'coma']
    for keyword in critical_keywords:
        if keyword in condition:
            return 1
    return 0

df['Medical Condition'] = df['Medical
```

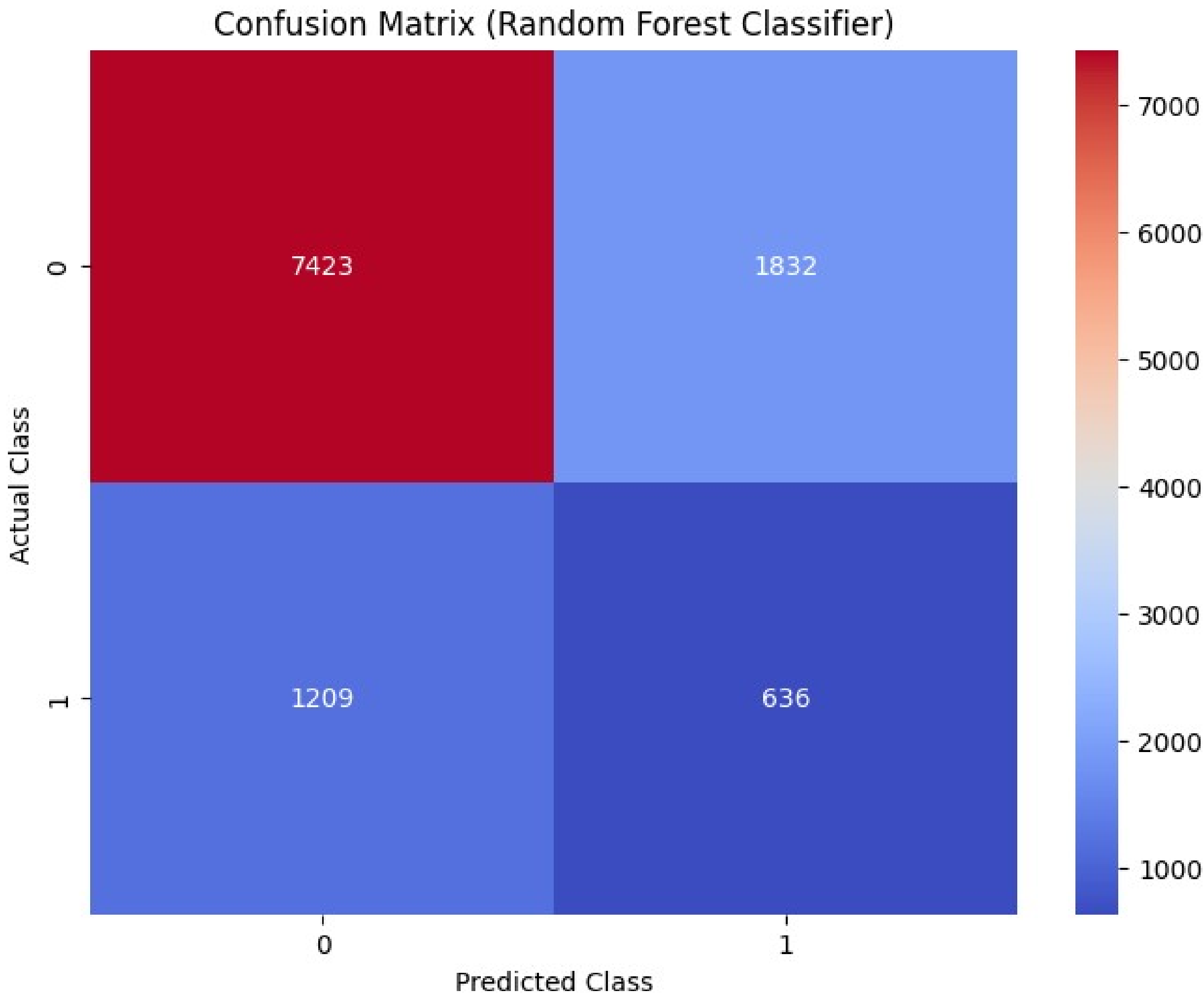
```python
Condition'].apply(categorize_condition)

# Label Encoding
le = LabelEncoder()
for col in ['Gender', 'Blood Type', 'Admission Type', 'Test Results',
'Medication']:
    df[col] = le.fit_transform(df[col].astype(str))

df['Billing_per_day'] = df['Billing Amount'] / (df['Length of Stay'] +
1)

# Scale numerical features
# scaler = StandardScaler()
# df[['Age', 'Billing Amount', 'Length of Stay', 'Billing_per_day']] =
scaler.fit_transform(
#     df[['Age', 'Billing Amount', 'Length of Stay',
'Billing_per_day']]
# )


# Split the dataset into features and target variable
# Prepare features and target
X = df.drop(['Medical Condition', 'Date of Admission', 'Discharge
Date'], axis=1)
y = df['Medical Condition']

# Stratified split to preserve class balance
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)

print("Class distribution before SMOTE:")
print(y_train.value_counts())

#Apply SMOTE to balance the dataset
sm = SMOTE(random_state=42)
X_res, y_res = sm.fit_resample(X_train, y_train)

print("Class distribution after SMOTE:")
print(pd.Series(y_res).value_counts())
```

```
Class distribution before SMOTE:
Medical Condition
0    37018
1     7382
Name: count, dtype: int64
Class distribution after SMOTE:
Medical Condition
0    37018
```

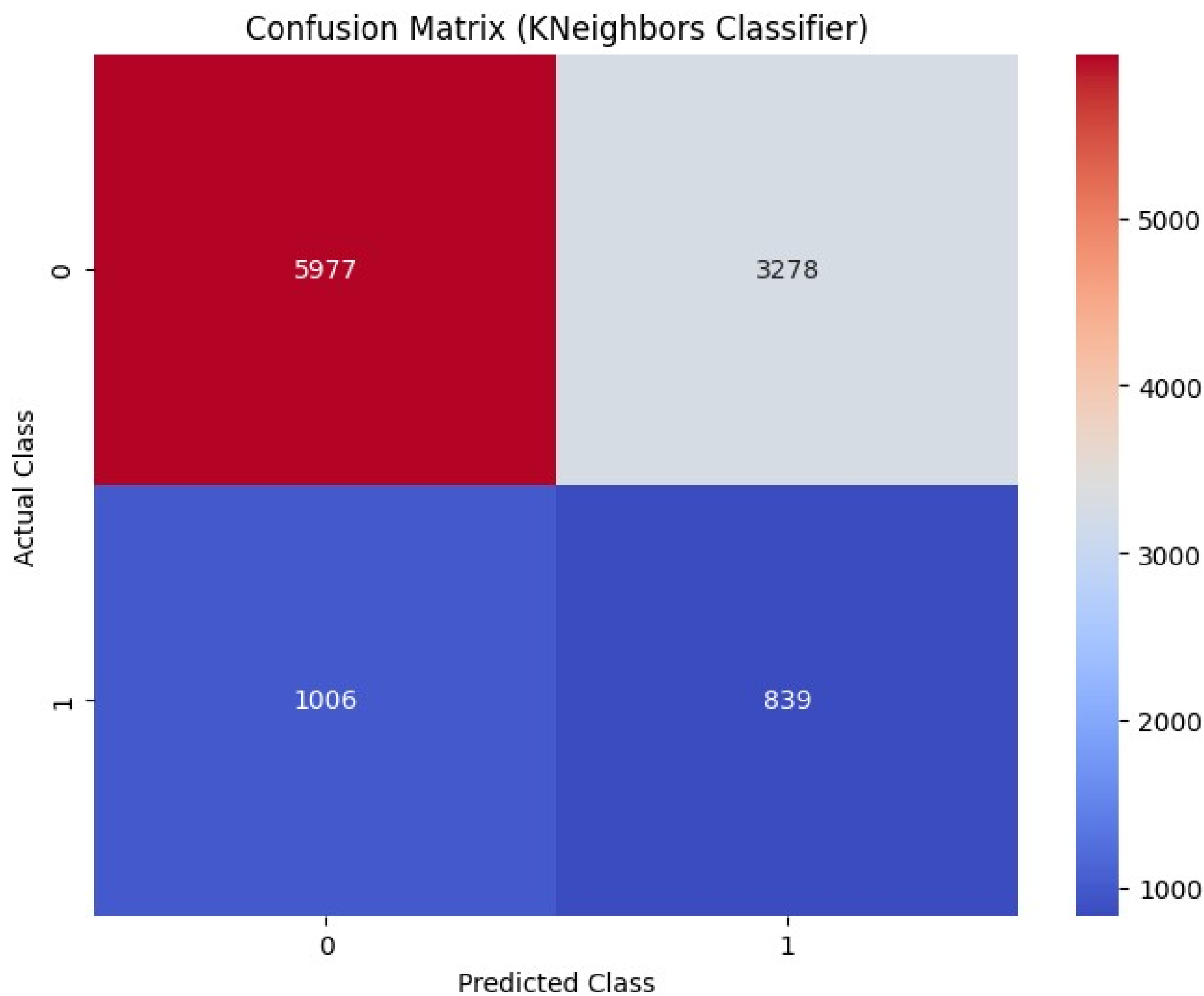
```
1    37018
Name: count, dtype: int64
```

```
#Train Random Forest Classifier
print("\n--- Training Random Forest Classifier ---")
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_res, y_res)
y_pred = rf.predict(X_test)

print("Accuracy Using RandomForestClassifier:", accuracy_score(y_test,
y_pred))
print(classification_report(y_test, y_pred))

# AUC-ROC Score
y_proba = rf.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, y_proba)
print("AUC-ROC Score:", roc_auc)

#Train KNeighbors Classifier
print("\n--- Training KNeighborsClassifier ---")
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_res, y_res)
knn_pred = knn.predict(X_test)
# knn_acc = accuracy_score(y_test, knn_pred)

print("Accuracy Using KNeighborsClassifier:", accuracy_score(y_test,
knn_pred))
print(classification_report(y_test, knn_pred))

# AUC-ROC Score
knn_proba = knn.predict_proba(X_test)[:, 1]
roc_auc = roc_auc_score(y_test, knn_proba)
print("AUC-ROC Score:", roc_auc)


cm_rfc = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm_rfc, annot=True, fmt="d", cmap="coolwarm")
plt.title('Confusion Matrix (Random Forest Classifier)')
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class')
plt.show()

cm_knc = confusion_matrix(y_test, knn_pred)
plt.figure(figsize=(8,6))
sns.heatmap(cm_knc, annot=True, fmt="d", cmap="coolwarm")
plt.title('Confusion Matrix (KNeighbors Classifier)')
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class')
plt.show()
```

```
--- Training Random Forest Classifier ---
Accuracy Using RandomForestClassifier: 0.7260360360360361
              precision    recall  f1-score   support

           0       0.86      0.80      0.83      9255
           1       0.26      0.34      0.29      1845

    accuracy                           0.73     11100
   macro avg       0.56      0.57      0.56     11100
weighted avg       0.76      0.73      0.74     11100

AUC-ROC Score: 0.6042315660325701

--- Training KNeighborsClassifier ---
Accuracy Using KNeighborsClassifier: 0.614054054054054
              precision    recall  f1-score   support

           0       0.86      0.65      0.74      9255
           1       0.20      0.45      0.28      1845

    accuracy                           0.61     11100
   macro avg       0.53      0.55      0.51     11100
weighted avg       0.75      0.61      0.66     11100

AUC-ROC Score: 0.5643441836903512
```



Confusion Matrix (Random Forest Classifier)
7423
1832
1209
636
0
1
Actual Class
Predicted Class
7000
6000
5000
4000
3000
2000
1000




```
# Model Explainability (SHAP)
print("\n--- SHAP Explainability ---")
# Sample 2000 patients to reduce memory usage
sample_idx = np.random.choice(X_test.index, 2000, replace=False)
X_sample = X_test.loc[sample_idx]

# Generate SHAP values
# Initialize SHAP explainer for Random Forest
explainer = shap.TreeExplainer(rf)
shap_values = explainer.shap_values(X_sample)

# Plot with explicit rendering
plt.figure(figsize=(10, 6))
shap.summary_plot(
    shap_values,
    X_sample,
    plot_type="bar",
    feature_names=X_test.columns.tolist(),
```

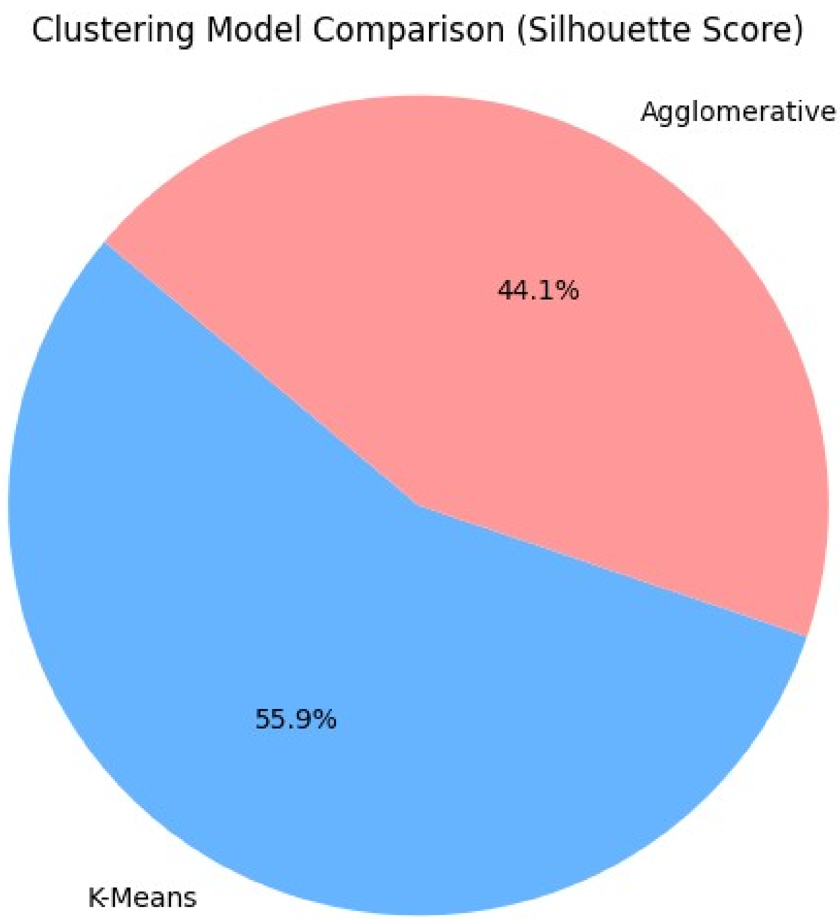
```
    show=False
)
plt.title("SHAP Feature Impact (2000 Patients)")
plt.tight_layout()
plt.show()
```

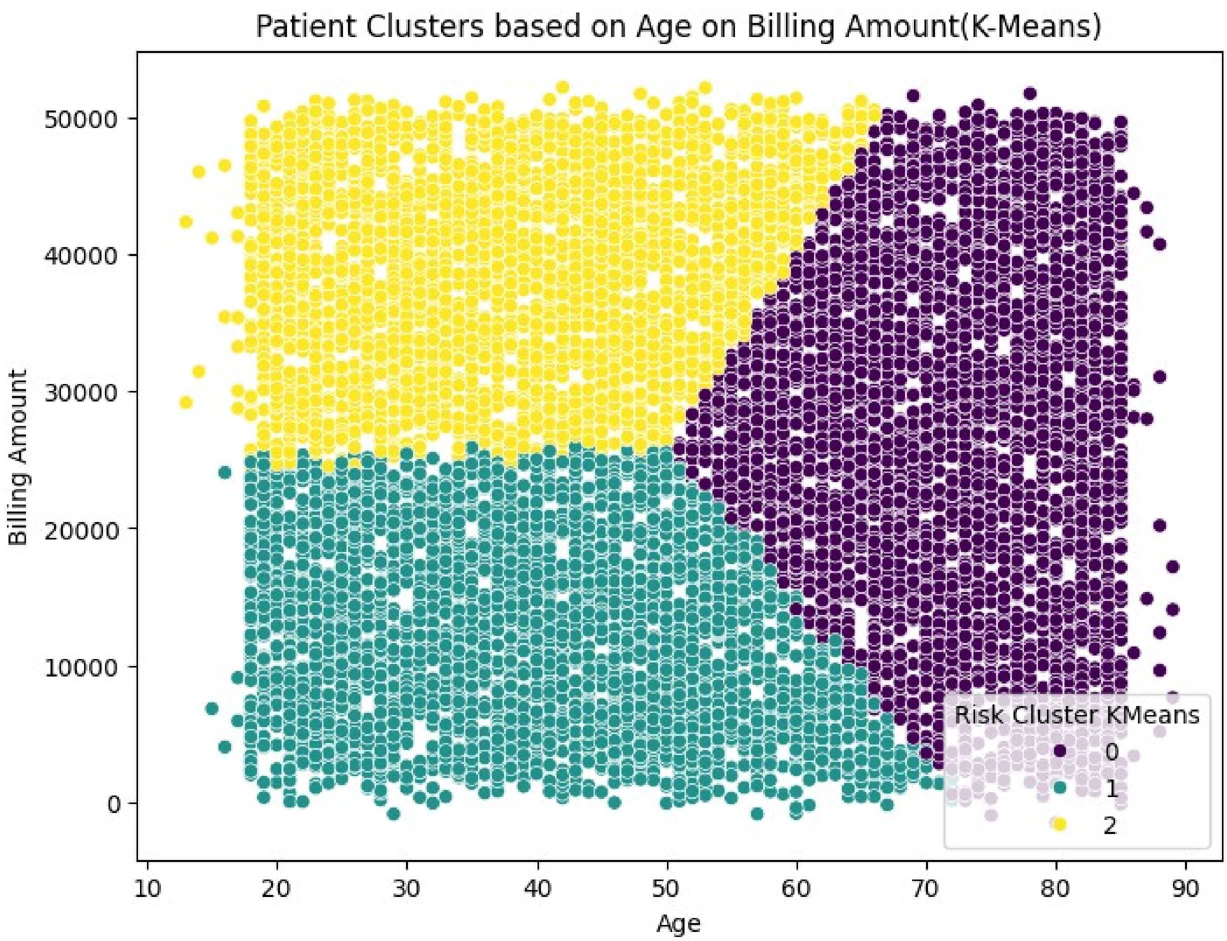
```
--- SHAP Explainability ---

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


```
# Select and scale features
# print(len(df))
# Sample only 10000 patients for clustering
df_ssf = df.sample(n=10000,random_state=42)
cluster_data = df_ssf[['Age', 'Billing Amount', 'Length of Stay']]
scaler = StandardScaler()
scaled_data = scaler.fit_transform(cluster_data)

# Cluster patients using KMeans
print("\n--- Patient Risk Clustering (K-Means) ---")
kmeans = KMeans(n_clusters=3, random_state=42)
df_ssf['Risk Cluster KMeans'] = kmeans.fit_predict(scaled_data)
kmeans_sil = silhouette_score(scaled_data, df_ssf['Risk Cluster
KMeans'])
db_score = davies_bouldin_score(scaled_data, df_ssf['Risk Cluster
KMeans'])
```

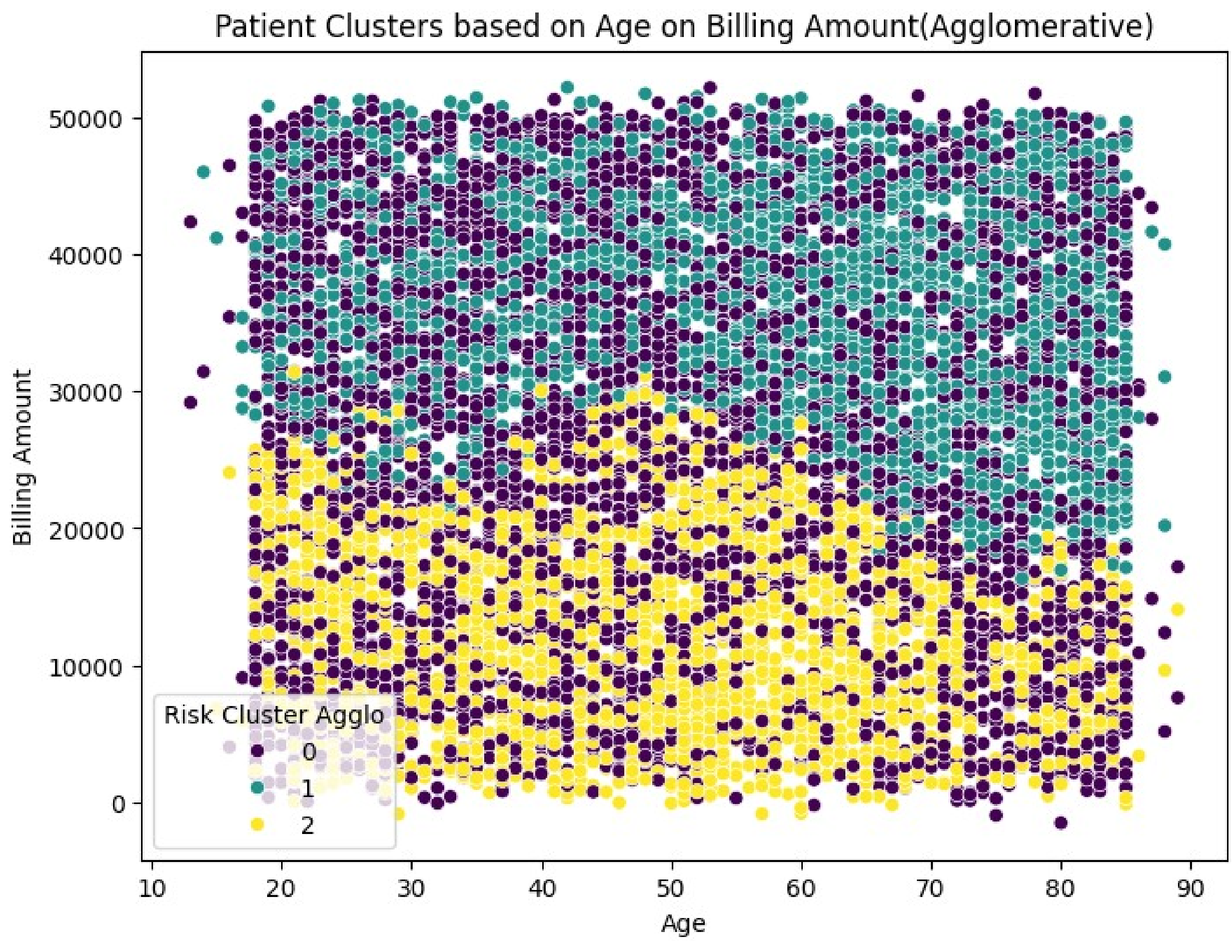
```python
print(f"Silhouette Score: {kmeans_sil:.3f}")
print(f"Davies-Bouldin Index: {db_score:.3f}")

print("\n--- Patient Risk Clustering (AgglomerativeClustering) ---")
agg = AgglomerativeClustering(n_clusters=3)
agg_labels = agg.fit_predict(scaled_data)
df_ssf['Risk Cluster Agglo'] = agg_labels
agg_sil = silhouette_score(scaled_data, agg_labels)
agg_db = davies_bouldin_score(scaled_data, agg_labels)
print(f"Silhouette Score: {agg_sil:.3f}")
print(f"Davies-Bouldin Index: {agg_db:.3f}")


# print(df_ssf)
# Plotting the silhouette scores in pie chart
labels = ['K-Means', 'Agglomerative']
silhouette_scores = [kmeans_sil, agg_sil]
colors = ['#66b3ff', '#ff9999']
plt.figure(figsize=(6,6))
plt.pie(silhouette_scores, labels=labels, autopct='%1.1f%%',
colors=colors, startangle=140)
plt.title('Clustering Model Comparison (Silhouette Score)')
plt.axis('equal')
plt.show()

# Visualize KMeans clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Age', y='Billing Amount', hue='Risk Cluster
KMeans', data=df_ssf, palette='viridis')
plt.title('Patient Clusters based on Age on Billing Amount(K-Means)')
plt.show()

# Visualize Agglomerative clusters
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Age', y='Billing Amount', hue='Risk Cluster Agglo',
data=df_ssf, palette='viridis')
plt.title('Patient Clusters based on Age on Billing
Amount(Agglomerative)')
plt.show()
```

```
--- Patient Risk Clustering (K-Means) ---
Silhouette Score: 0.243
Davies-Bouldin Index: 1.322

--- Patient Risk Clustering (AgglomerativeClustering) ---
Silhouette Score: 0.192
Davies-Bouldin Index: 1.433
```


Clustering Model Comparison (Silhouette Score)
Agglomerative
44.1%
55.9%
K-Means

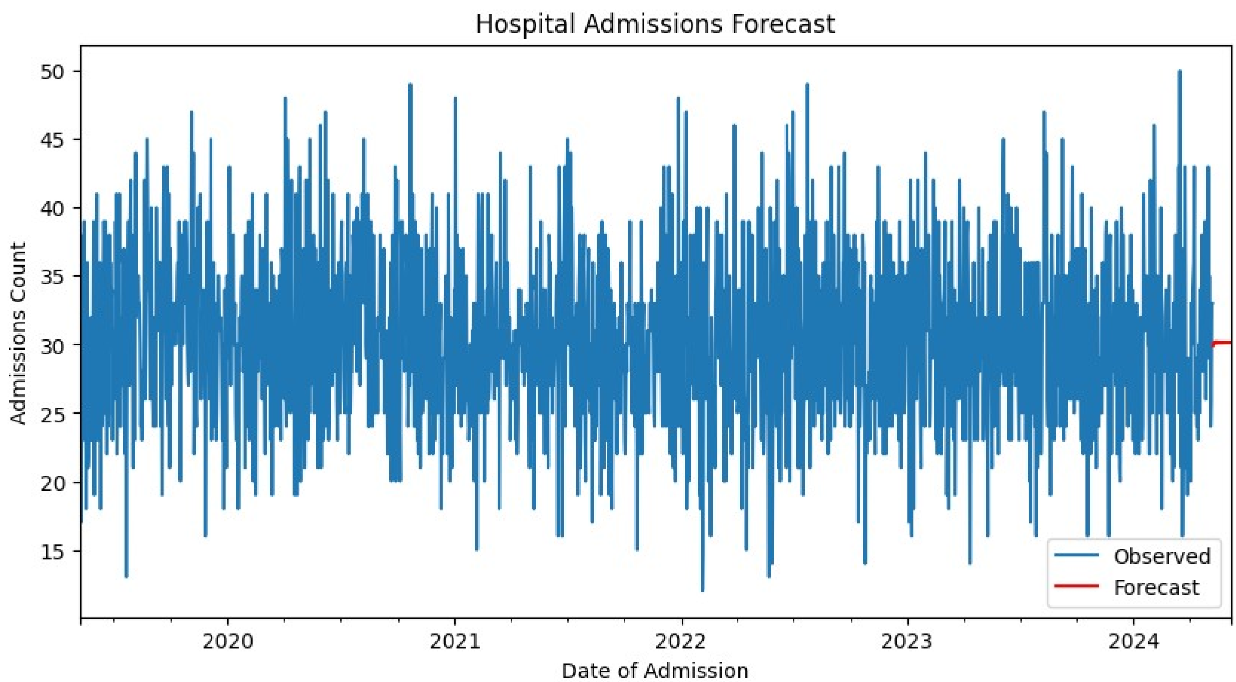


Patient Clusters based on Age on Billing Amount(K-Means)
Billing Amount
Age
Risk Cluster KMeans
0
1
2
0
10000
20000
30000
40000
50000
10
20
30
40
50
60
70
80
90




```
print("\n--- Hospital Resource Forecasting (ARIMA) ---")
# Resample the data to daily frequency and fill missing dates with 0
admissions
admissions = df.groupby('Date of
Admission').size().asfreq('D').fillna(0)
model = ARIMA(admissions, order=(3, 1, 2))
model_fit = model.fit()
forecast = model_fit.forecast(steps=30)

plt.figure(figsize=(10, 5))
admissions.plot(label='Observed')
forecast.plot(label='Forecast', color='red')
plt.legend()
plt.title('Hospital Admissions Forecast')
plt.ylabel('Admissions Count')
plt.show()


--- Hospital Resource Forecasting (ARIMA) ---
```

```
c:\Users\roshan\Desktop\Dissertation\.venv\Lib\site-packages\
statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Non-stationary
starting autoregressive parameters found. Using zeros as starting
parameters.
  warn('Non-stationary starting autoregressive parameters'
c:\Users\roshan\Desktop\Dissertation\.venv\Lib\site-packages\
statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Non-invertible
starting MA parameters found. Using zeros as starting parameters.
  warn('Non-invertible starting MA parameters found.'
```




```
# Evaluate the model
predicted = model_fit.predict(start=len(admissions)-30,
end=len(admissions)-1, dynamic=False)
true = admissions[-30:]
print("MAE:", mean_absolute_error(true, predicted))
print("RMSE:", np.sqrt(mean_squared_error(true, predicted)))

MAE: 4.467349574327432
RMSE: 5.662922822234962
```