```
#importing library
import pandas as pd
from pandas import read_csv, get_dummies, DataFrame,Series
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
from sklearn import metrics
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import f1_score


#loading the dataset
data = pd.read_csv('/content/healthcare_dataset.csv')


#to know the data info
data.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
```

```
#to check the null value
data.isnull().sum()
```

	0
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

to check value counts in colunms

```
data['Blood Type'].value_counts()
```

Blood Type	count
A-	6969
A+	6956
AB+	6947
AB-	6945
B+	6945
B-	6944
O+	6917
O-	6877

dtype: int64

```
data['Medical Condition'].value_counts()
```

Medical Condition	count
Arthritis	9308
Diabetes	9304
Hypertension	9245
Obesity	9231
Cancer	9227
Asthma	9185

dtype: int64

```
data['Doctor'].value_counts()
```

Doctor	count
Michael Smith	27
John Smith	22
Robert Smith	22
Michael Johnson	20
James Smith	20
...	...
Keith Ortiz	1
Timothy Chapman	1
Terri Collins	1
Jill Jacobson	1
Scott Coleman	1

40341 rows × 1 columns

dtype: int64

```
data['Hospital'].value_counts()
```

Hospital	count
LLC Smith	44
Ltd Smith	39
Johnson PLC	38
Smith Ltd	37
Smith PLC	36
...	...
Hammond Ltd	1
Moran Smith and Galloway,	1
Winters Martin, and Neal	1
James-Owen	1
Marks, and Vazquez Jenkins	1

39876 rows × 1 columns

dtype: int64

```
data['Insurance Provider'].value_counts()
```

Insurance Provider	count
Cigna	11249
Medicare	11154
UnitedHealthcare	11125
Blue Cross	11059
Aetna	10913

dtype: int64

```
data['Room Number'].value_counts()
```

Room Number	count
393	181
491	177
420	175
104	175
392	171
...	...
253	112
254	111
257	111
381	110
398	109

400 rows × 1 columns

dtype: int64

```
data['Admission Type'].value_counts()
```

Admission Type	count
Elective	18655
Urgent	18576
Emergency	18269

dtype: int64

```
data['Medication'].value_counts()
```

Medication	count
Lipitor	11140
Ibuprofen	11127
Aspirin	11094
Paracetamol	11071
Penicillin	11068

dtype: int64

```
data['Test Results'].value_counts()
```

Test Results	count
Abnormal	18627
Normal	18517
Inconclusive	18356

dtype: int64

to drop unwanted colunms

```
data_to_drop=['Name','Doctor',
              'Hospital','Date of Admission','Room Number',
              'Discharge Date']


data_cleaned=data.drop(data_to_drop,axis=1)


data_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55500 entries, 0 to 55499
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 55500 non-null  int64
 1   Gender              55500 non-null  object
 2   Blood Type          55500 non-null  object
 3   Medical Condition   55500 non-null  object
 4   Insurance Provider  55500 non-null  object
 5   Billing Amount      55500 non-null  float64
```

```
 6   Admission Type      55500 non-null  object
 7   Medication          55500 non-null  object
 8   Test Results        55500 non-null  object
dtypes: float64(1), int64(1), object(7)
memory usage: 3.8+ MB
```

converting gender colunm into 1 and 2 beacuse it is categorical values to numeric values

```
data_cleaned['Gender']=data_cleaned['Gender'].map({'Male':1,'Female':0})
```

```
data_cleaned['Gender']
```

	Gender
0	1
1	1
2	0
3	0
4	0
...	...
55495	0
55496	0
55497	0
55498	1
55499	0

55500 rows × 1 columns

dtype: int64

converting numberical values although it 3 values like normal,abnormal and Inconclusive we covert it into two normal and abnormal

```
data_cleaned['Test Results'] = data_cleaned['Test Results'].apply(lambda x: 0 if x == 'Norma
```

```
data_cleaned['Test Results']
```

	Test Results
0	0
1	1
2	0
3	1
4	1
...	...
55495	1
55496	0
55497	1
55498	1
55499	1

55500 rows × 1 columns

dtype: int64

```
data_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55500 entries, 0 to 55499
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   Age                 55500 non-null  int64  
 1   Gender              55500 non-null  int64  
 2   Blood Type          55500 non-null  object 
 3   Medical Condition   55500 non-null  object 
 4   Insurance Provider  55500 non-null  object 
 5   Billing Amount      55500 non-null  float64
 6   Admission Type      55500 non-null  object 
 7   Medication          55500 non-null  object 
 8   Test Results        55500 non-null  int64  
dtypes: float64(1), int64(3), object(5)
memory usage: 3.8+ MB
```

taking dummies of objects

```
data_cleaned=get_dummies(data_cleaned,['Blood Type','Medical Condition',
                                       'Insurance Provider','Admission Type','Medication'])
```

```
data_cleaned['Test Results'].value_counts()
```

Test Results	count
1	36983
0	18517

dtype: int64

dropping test result indo y data and taking balance colunms as X data

```
X=data_cleaned.drop('Test Results',axis=1)
Y=data_cleaned['Test Results']
```

```
Y.value_counts()
```

Test Results	count
1	36983
0	18517

dtype: int64

decsion tree

```
from sklearn import tree#importing tree
```

```
X_scaled=StandardScaler().fit_transform(X)
X_train, X_test, Y_train, Y_test = train_test_split(X_scaled, Y, test_size = 0.3, random_sta
X_train, Y_train =SMOTE(random_state=1).fit_resample(X_train, Y_train)#smoting the data beca
dec_tree= tree.DecisionTreeClassifier(criterion='entropy',max_depth=5)#giving tree depth as
dec_tree.fit(X_train,Y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=5)
```

```
Y_pred=dec_tree.predict(X_test)
```

```
Accuracy= metrics.accuracy_score(Y_test,Y_pred)
Recall= metrics.recall_score(Y_test,Y_pred)
Precision= metrics.precision_score(Y_test,Y_pred)
print('Accuracy:', Accuracy)
print('Recall:', Recall)
print('Precision:', Precision)
f1 = f1_score(Y_test, Y_pred)
print("F1-Score:", f1)
```

```
Accuracy: 0.6694894894894895
Recall: 0.9991929698708751
Precision: 0.669772194506221
F1-Score: 0.8019720033106625
```

```
conf_mat= metrics.confusion_matrix(Y_test,Y_pred)
```

```
conf_mat
```

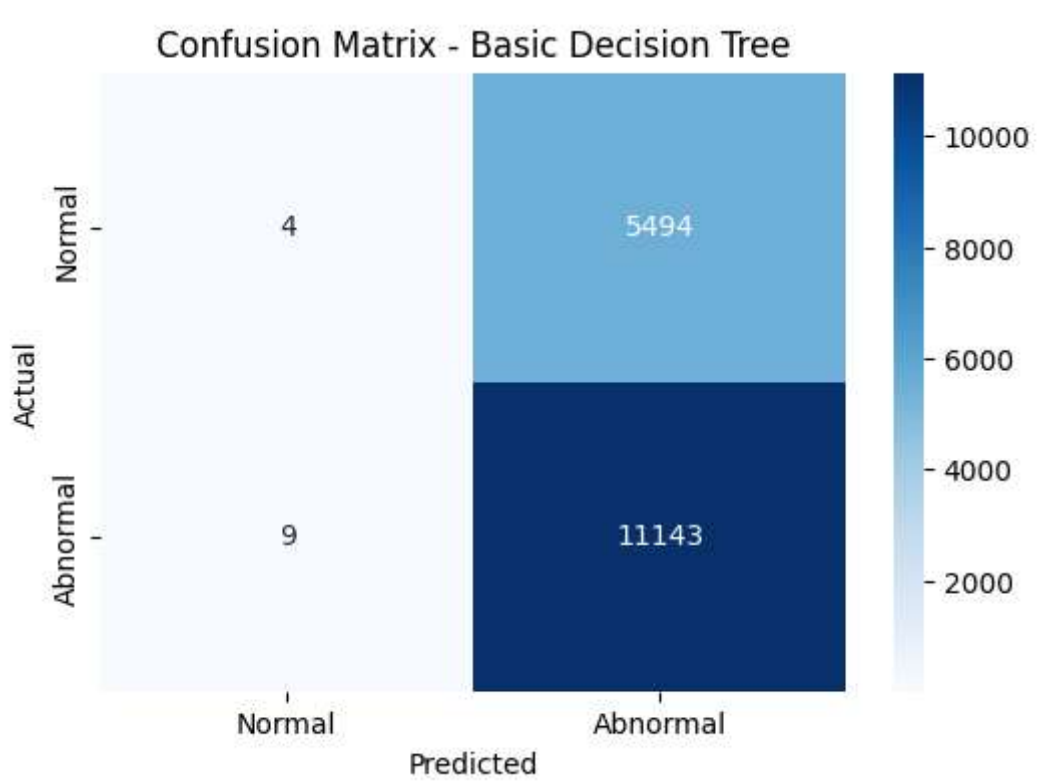
```
array([[    4,  5494],
       [    9, 11143]])
```

graph for basic dt

Confusion Matrix Heatmap

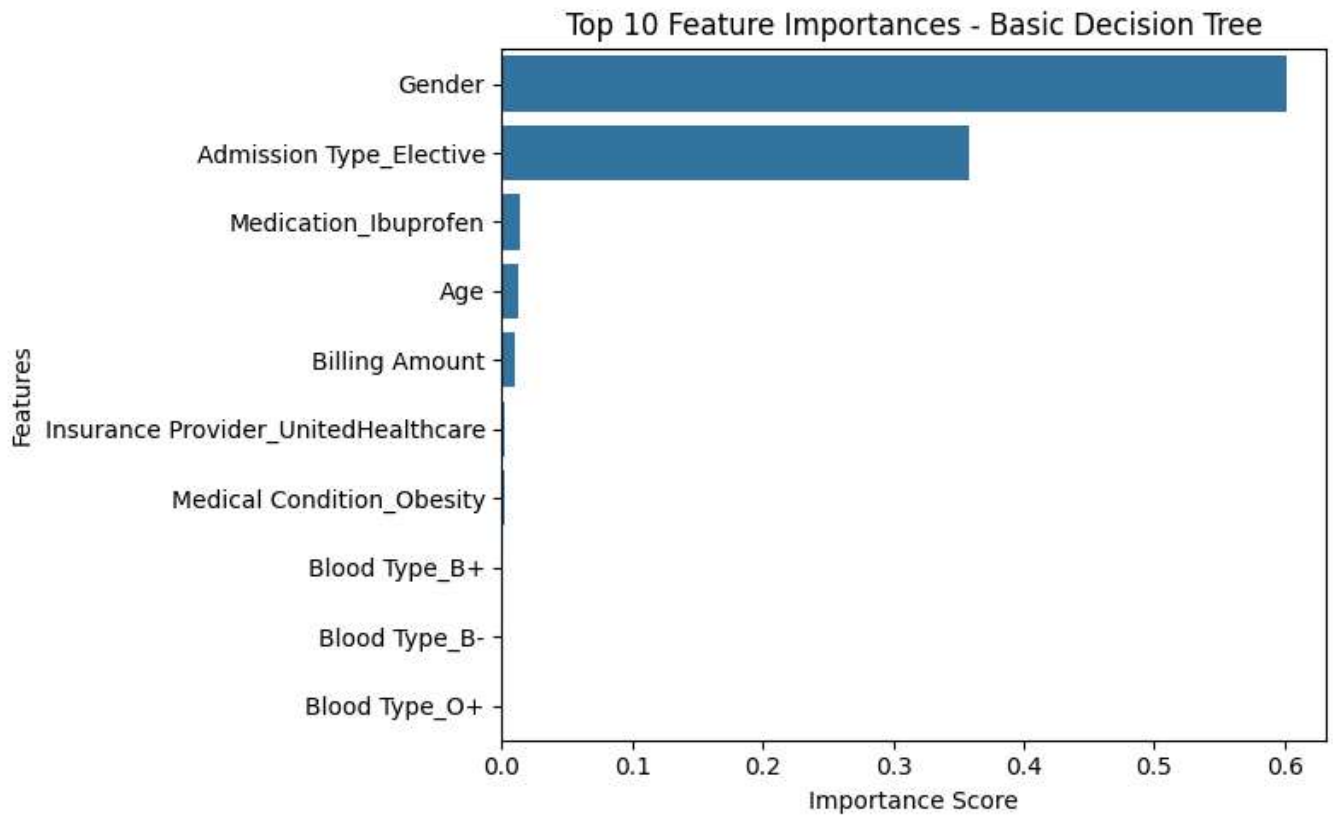
```
import seaborn as sns
import matplotlib.pyplot as plt
```

```
plt.figure(figsize=(6,4))
sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', xticklabels=["Normal", "Abnormal"],
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Basic Decision Tree')
plt.show()
```



Feature Importance Bar Plot

```
feat_importance = pd.Series(dec_tree.feature_importances_, index=X.columns)
feat_importance = feat_importance.sort_values(ascending=False).head(10)


plt.figure(figsize=(8,5))
sns.barplot(x=feat_importance.values, y=feat_importance.index)
plt.title("Top 10 Feature Importances - Basic Decision Tree")
plt.xlabel("Importance Score")
plt.ylabel("Features")
plt.tight_layout()
plt.show()
```



Tunned DT

```
from sklearn.model_selection import GridSearchCV


dec_tree_gs = tree.DecisionTreeClassifier(criterion='entropy')
depth_gs = {'max_depth':[5,10,15,20,25,30,35,40,45,46,47,48,49,50,51,52,53,54,55,60]}#to che
grid_search_gs = GridSearchCV(estimator=dec_tree_gs, param_grid=depth_gs, scoring='precisior
grid_search_gs.fit(X_train, Y_train)
best_depth_gs = grid_search_gs.best_params_
print(best_depth_gs)
```

```
{'max_depth': 53}
```

```
from sklearn.model_selection import GridSearchCV


dec_tree_gs_final = tree.DecisionTreeClassifier(criterion='entropy', max_depth=48)#got as 48
dec_tree_gs_final.fit(X_train, Y_train)
```

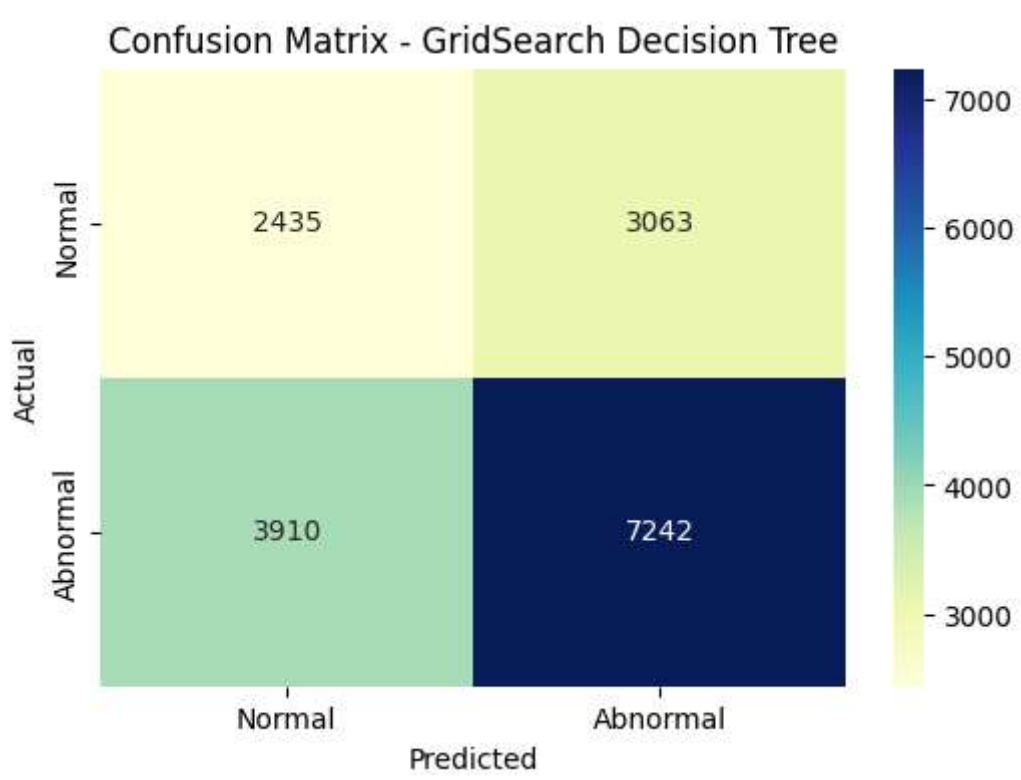
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=48)
```

```
Y_pred_gs = dec_tree_gs_final.predict(X_test)
Accuracy_gs = metrics.accuracy_score(Y_test, Y_pred_gs)
Recall_gs = metrics.recall_score(Y_test, Y_pred_gs)
Precision_gs = metrics.precision_score(Y_test, Y_pred_gs)
print('Accuracy:', Accuracy_gs)
print('Recall:', Recall_gs)
print('Precision:', Precision_gs)
f1_gs = f1_score(Y_test, Y_pred_gs)
print("F1-Score:", f1_gs)
conf_mat_gs = metrics.confusion_matrix(Y_test, Y_pred_gs)
print(conf_mat_gs)
```

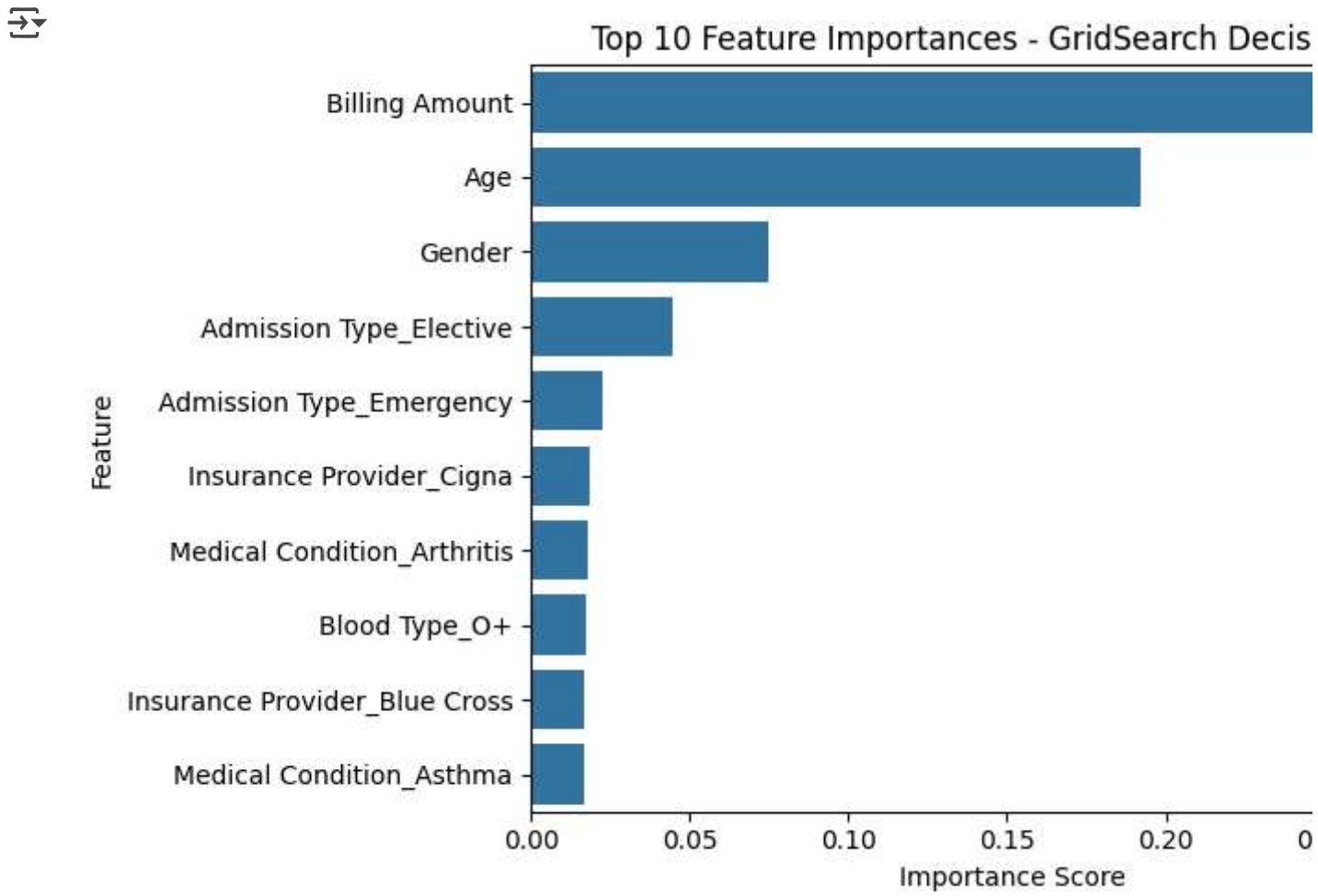
```
Accuracy: 0.5812012012012012
Recall: 0.649390243902439
Precision: 0.7027656477438137
F1-Score: 0.6750244675397307
[[2435 3063]
 [3910 7242]]
```

graph for Tunned DT

```
plt.figure(figsize=(6,4))
sns.heatmap(conf_mat_gs, annot=True, fmt='d', cmap='YlGnBu', xticklabels=["Normal", "Abnorma
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - GridSearch Decision Tree')
plt.show()
```



```
feat_importance_gs = pd.Series(dec_tree_gs_final.feature_importances_, index=X.columns)
feat_importance_gs = feat_importance_gs.sort_values(ascending=False).head(10)

plt.figure(figsize=(8,5))
sns.barplot(x=feat_importance_gs.values, y=feat_importance_gs.index)
plt.title("Top 10 Feature Importances - GridSearch Decision Tree")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```



random forest

```
from sklearn.ensemble import RandomForestClassifier


from sklearn import ensemble
rf_model = RandomForestClassifier(n_estimators=5, criterion='entropy', max_features=None, ra
rf_model.fit(X_train, Y_train)
```

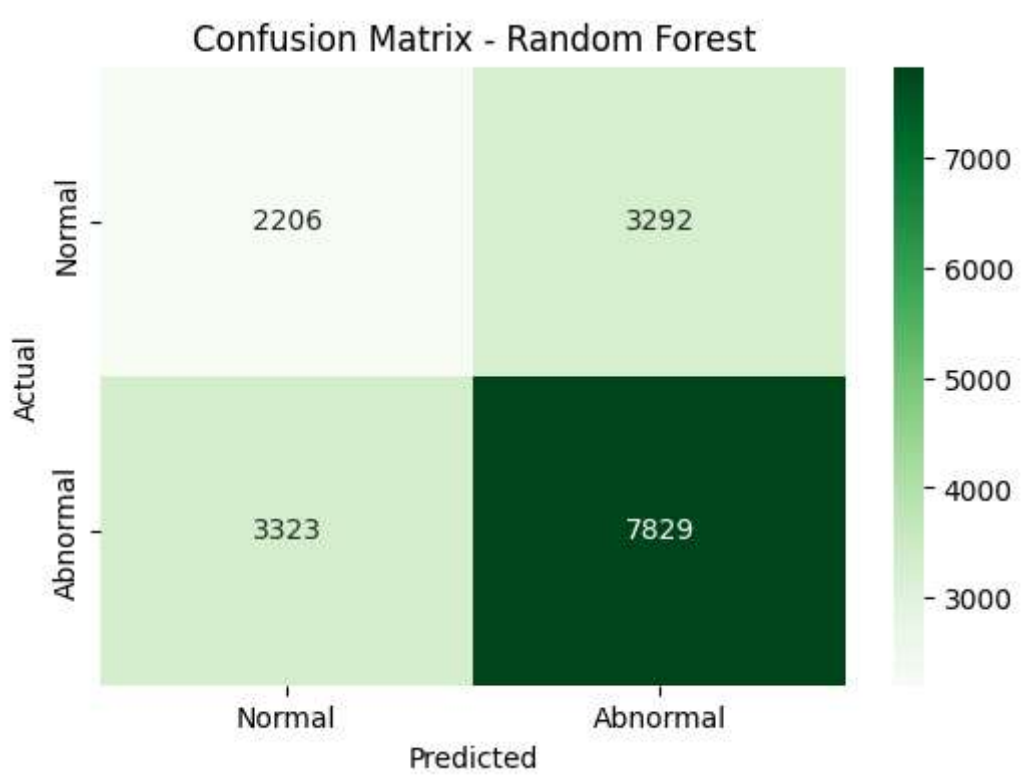
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_features=None, n_estimators=5,
                       random_state=1)
```

```
Y_pred_rf = rf_model.predict(X_test)


Accuracy_rf = accuracy_score(Y_test, Y_pred_rf)
Recall_rf = recall_score(Y_test, Y_pred_rf)
Precision_rf = precision_score(Y_test, Y_pred_rf)
f1_rf = f1_score(Y_test, Y_pred_rf)

print("Random Forest Results")
print("Accuracy:", Accuracy_rf)
print("Recall:", Recall_rf)
print("Precision:", Precision_rf)
print("F1-Score:", f1_rf)
conf_mat_rf = confusion_matrix(Y_test, Y_pred_rf)
print("Confusion Matrix:\n", conf_mat_rf)
```

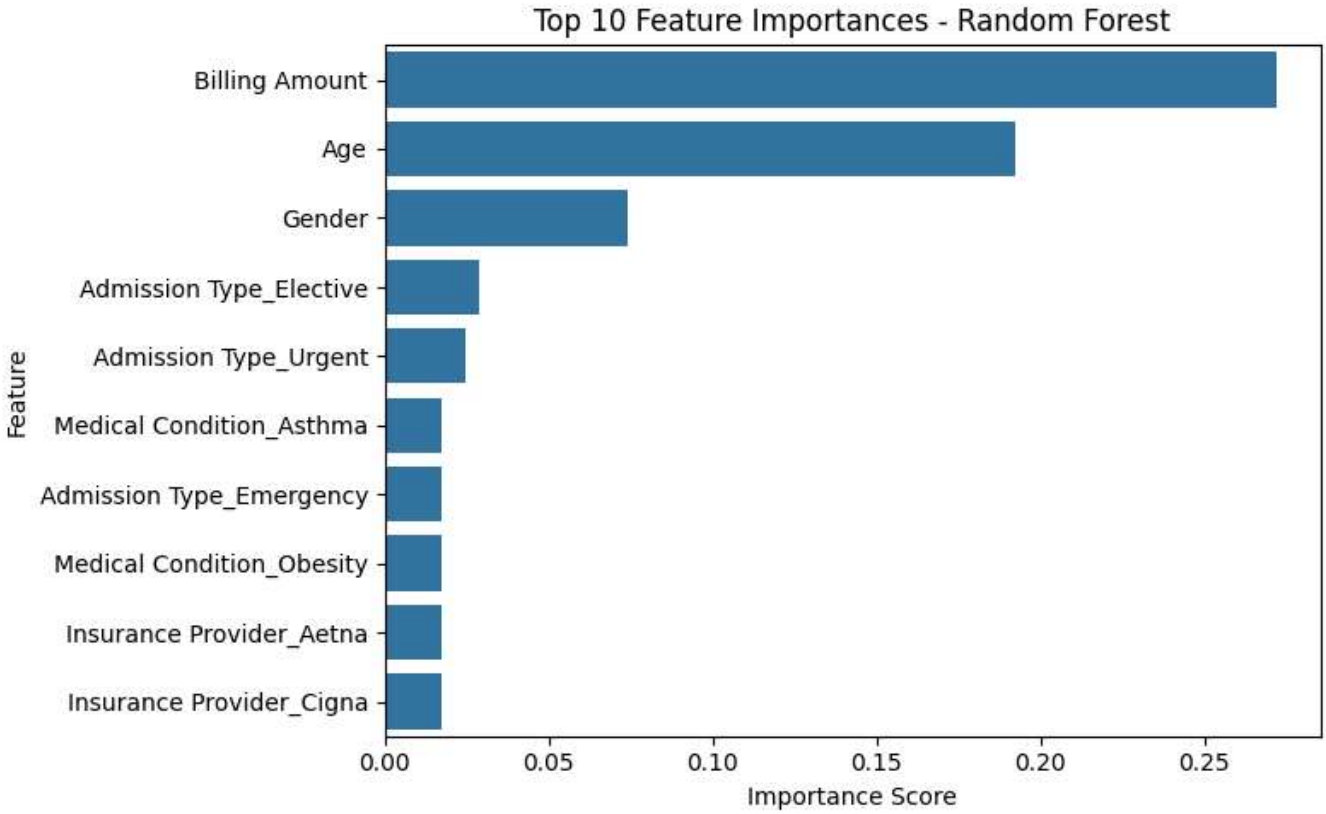
```
Random Forest Results
Accuracy: 0.6027027027027027
Recall: 0.7020265423242468
Precision: 0.7039834547252944
F1-Score: 0.703003636690163
Confusion Matrix:
 [[2206 3292]
 [3323 7829]]
```

graphs for RF

```
plt.figure(figsize=(6,4))
sns.heatmap(conf_mat_rf, annot=True, fmt='d', cmap='Greens', xticklabels=["Normal", "Abnorma
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Random Forest')
plt.show()
```

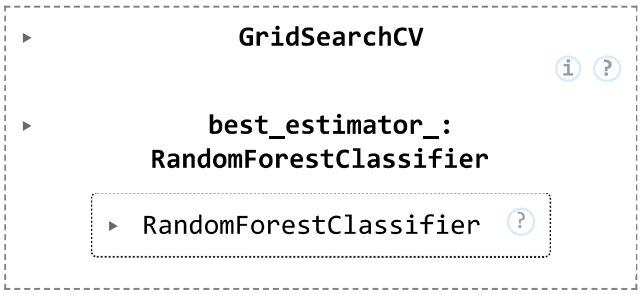



```
feat_importance_rf = pd.Series(rf_model.feature_importances_, index=X.columns)
feat_importance_rf = feat_importance_rf.sort_values(ascending=False).head(10)


plt.figure(figsize=(8,5))
sns.barplot(x=feat_importance_rf.values, y=feat_importance_rf.index)
plt.title("Top 10 Feature Importances - Random Forest")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```



Tunned RF

```
rf_gs = RandomForestClassifier(criterion='entropy', max_features=None, random_state=1)
param_grid_rf = {'n_estimators': [200,300]}#to check the best estimators for that we have ch
grid_search_rf = GridSearchCV(estimator=rf_gs, param_grid=param_grid_rf, scoring='precision'
grid_search_rf.fit(X_train, Y_train)
```




```
best_rf_params = grid_search_rf.best_params_
print("Best n_estimators:", best_rf_params)
```

```
Best n_estimators: {'n_estimators': 200}
```

```
rf_model_gs = RandomForestClassifier(n_estimators=best_rf_params['n_estimators'], criterion=
#giving the best estimator as 200 but it is automattically updated from the above code
rf_model_gs.fit(X_train, Y_train)
```

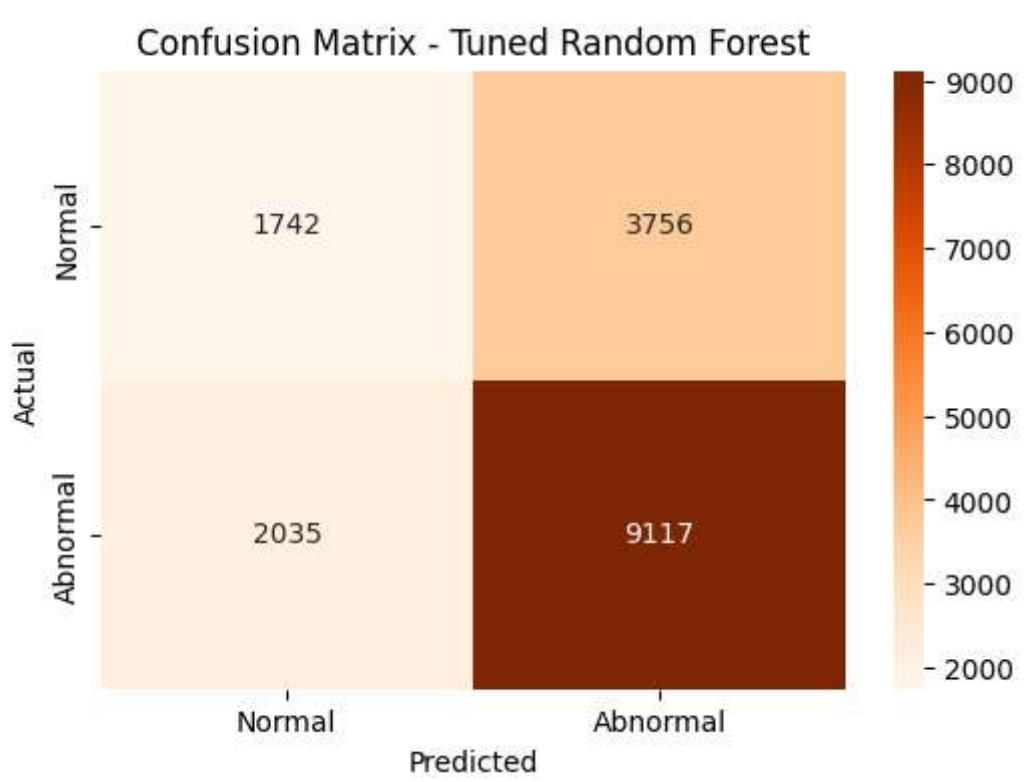
```
RandomForestClassifier
RandomForestClassifier(criterion='entropy', max_features=None, n_estimators=200,
                       random_state=1)
```

```
from sklearn.metrics import f1_score


Y_pred_rf_gs = rf_model_gs.predict(X_test)
Accuracy_rf_gs = accuracy_score(Y_test, Y_pred_rf_gs)
Recall_rf_gs = recall_score(Y_test, Y_pred_rf_gs)
Precision_rf_gs = precision_score(Y_test, Y_pred_rf_gs)
f1_rf_gs = f1_score(Y_test, Y_pred_rf_gs)


print("Tuned Random Forest Results")
print("Accuracy:", Accuracy_rf_gs)
print("Recall:", Recall_rf_gs)
print("Precision:", Precision_rf_gs)
print("F1-Score:", f1_rf_gs)
conf_mat_rf_gs = confusion_matrix(Y_test, Y_pred_rf_gs)
print("Confusion Matrix:\n", conf_mat_rf_gs)
```

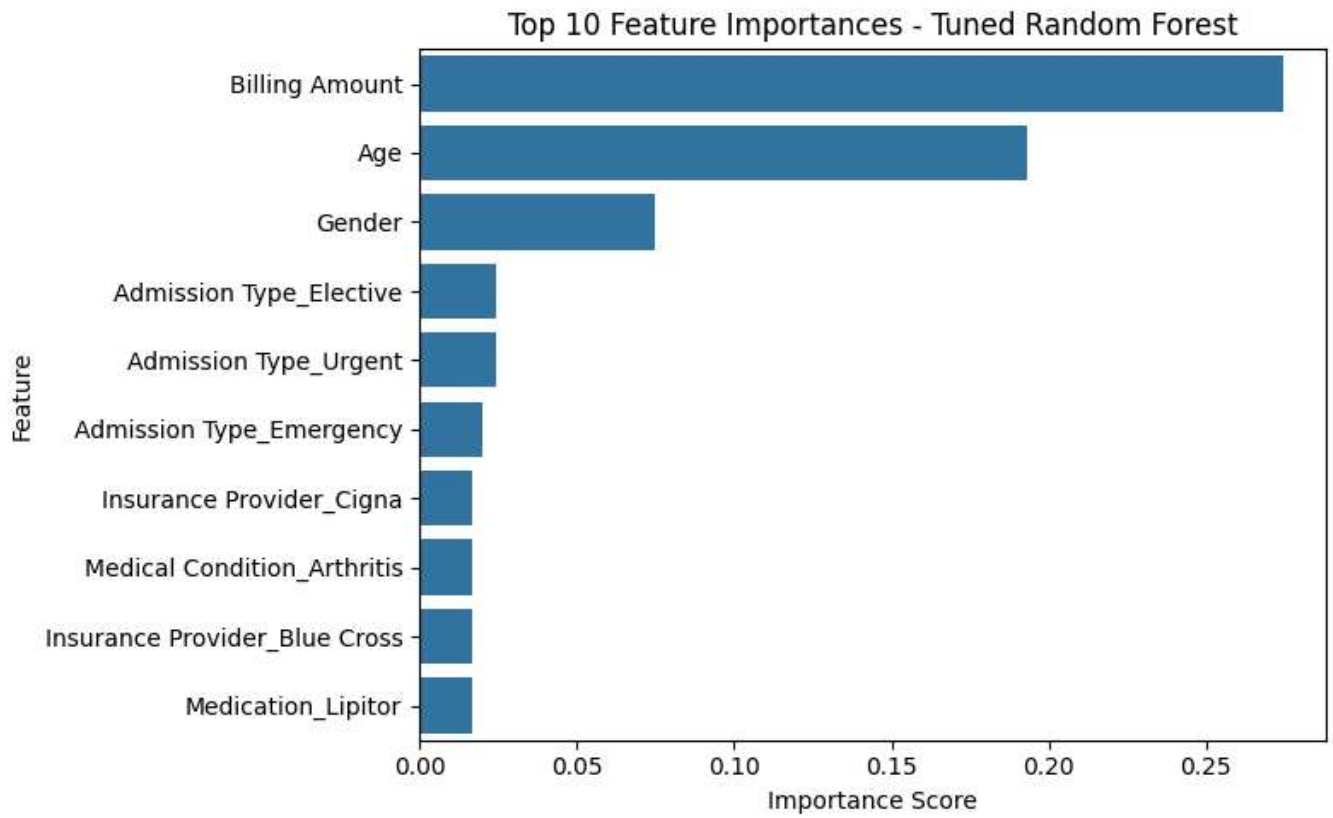
```
Tuned Random Forest Results
Accuracy: 0.6521921921921922
Recall: 0.8175215208034433
Precision: 0.708226520624563
F1-Score: 0.7589594172736732
Confusion Matrix:
 [[1742 3756]
 [2035 9117]]
```

Tunned RF graph

```
plt.figure(figsize=(6,4))
sns.heatmap(conf_mat_rf_gs, annot=True, fmt='d', cmap='Oranges', xticklabels=["Normal", "Abr
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Tuned Random Forest')
plt.show()
```




```
feat_importance_rf_gs = pd.Series(rf_model_gs.feature_importances_, index=X.columns)
feat_importance_rf_gs = feat_importance_rf_gs.sort_values(ascending=False).head(10)


plt.figure(figsize=(8,5))
sns.barplot(x=feat_importance_rf_gs.values, y=feat_importance_rf_gs.index)
plt.title("Top 10 Feature Importances - Tuned Random Forest")
plt.xlabel("Importance Score")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```



logistic Regression

```
from sklearn.linear_model import LogisticRegression
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusi
```

```
logreg_model = LogisticRegression(max_iter=1000)#giving intial value as 1000
logreg_model.fit(X_train, Y_train)
```

```
LogisticRegression
LogisticRegression(max_iter=1000)
```

```
Y_pred_logreg = logreg_model.predict(X_test)
```

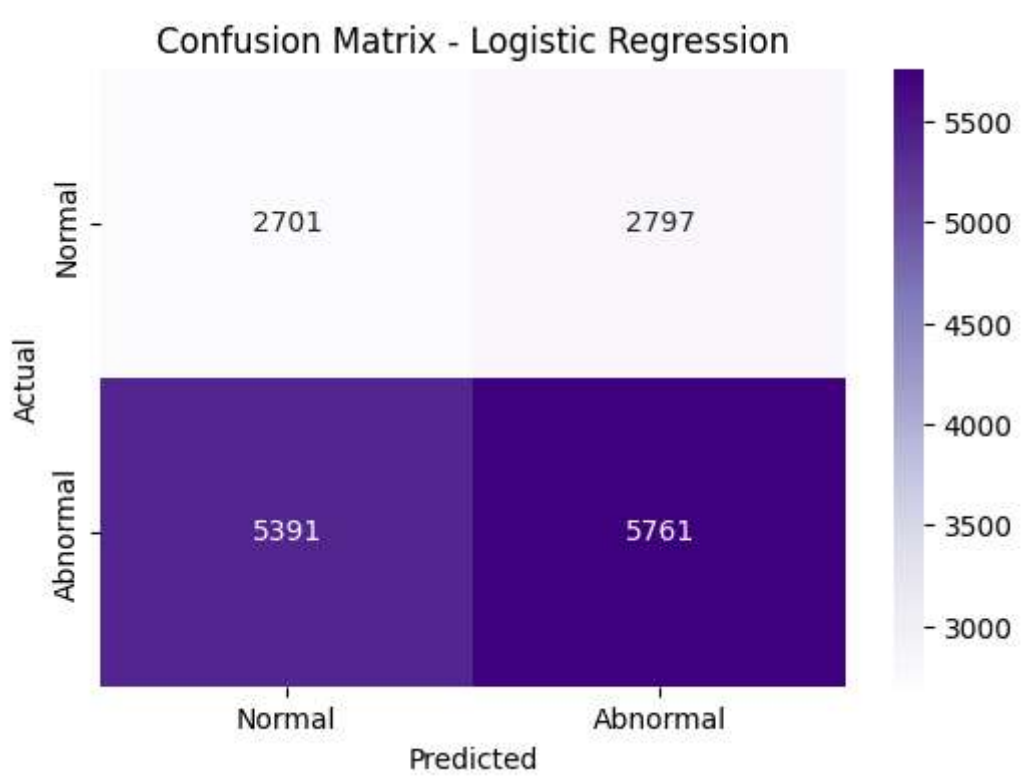
```
Accuracy_logreg = accuracy_score(Y_test, Y_pred_logreg)
Recall_logreg = recall_score(Y_test, Y_pred_logreg)
Precision_logreg = precision_score(Y_test, Y_pred_logreg)
f1_logreg = f1_score(Y_test, Y_pred_logreg)
```

```
print("Logistic Regression Results")
print("Accuracy:", Accuracy_logreg)
print("Recall:", Recall_logreg)
print("Precision:", Precision_logreg)
print("F1-Score:", f1_logreg)
conf_mat_logreg = confusion_matrix(Y_test, Y_pred_logreg)
print("Confusion Matrix:\n", conf_mat_logreg)
```

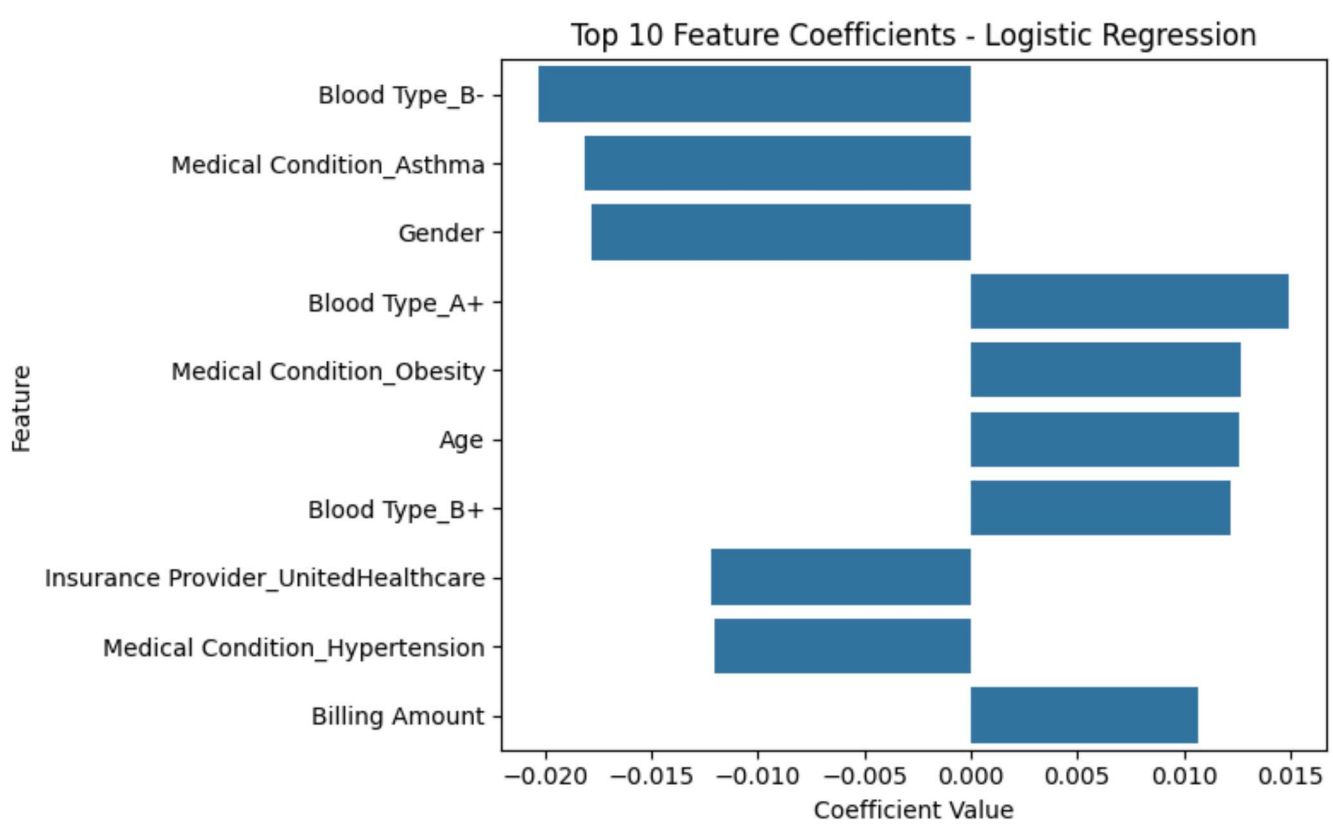
```
Logistic Regression Results
Accuracy: 0.5082282282282282
Recall: 0.5165889526542324
Precision: 0.673171301706006
F1-Score: 0.584576357179097
Confusion Matrix:
 [[2701 2797]
 [5391 5761]]
```

graph for logistic

```
plt.figure(figsize=(6,4))
sns.heatmap(conf_mat_logreg, annot=True, fmt='d', cmap='Purples', xticklabels=["Normal", "Ab
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Logistic Regression')
plt.show()
```



```
coefficients = pd.Series(logreg_model.coef_[0], index=X.columns)
coefficients = coefficients.sort_values(key=abs, ascending=False).head(10)


plt.figure(figsize=(8,5))
sns.barplot(x=coefficients.values, y=coefficients.index)
plt.title("Top 10 Feature Coefficients - Logistic Regression")
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```

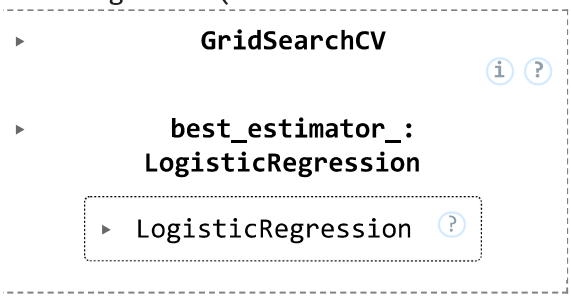


Tunned LogisticRegression

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_scc


param_grid_logreg = {
    'C': [0.1, 1],
    'penalty': ['l1', 'l2']
}
#tuning the logistic for better perfomance
```

```
logreg_gs = LogisticRegression(max_iter=1000)
grid_search_logreg = GridSearchCV(estimator=logreg_gs, param_grid=param_grid_logreg, scoring
grid_search_logreg.fit(X_train, Y_train)
```

```
usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py:528: FitF
0 fits failed out of a total of 20.
he score on these train-test partitions for these parameters will be set to nan.
f these failures are not expected, you can try to debug them by setting error_score='ra

elow are more details about the failures:
--------------------------------------------------------------------------------
0 fits failed with the following error:
raceback (most recent call last):
 File "/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_validation.py",
   estimator.fit(X_train, y_train, **fit_params)
 File "/usr/local/lib/python3.11/dist-packages/sklearn/base.py", line 1389, in wrapper
   return fit_method(estimator, *args, **kwargs)
          ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
 File "/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py", line
   solver = _check_solver(self.solver, self.penalty, self.dual)
            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
 File "/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py", line
   raise ValueError(
alueError: Solver lbfgs supports only 'l2' or None penalties, got l1 penalty.

 warnings.warn(some_fits_failed_message, FitFailedWarning)
usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWar
 warnings.warn(
```




```
print("Best Parameters (LogReg):", grid_search_logreg.best_params_)
```

```
Best Parameters (LogReg): {'C': 1, 'penalty': 'l2'}
```

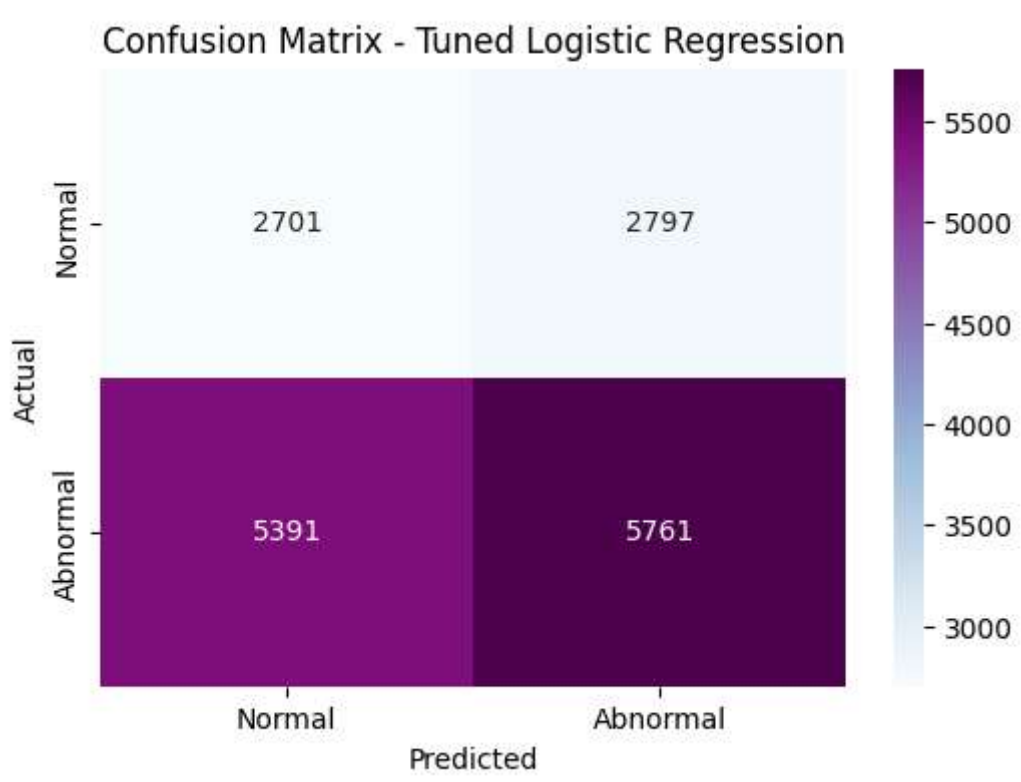
```
best_logreg_model = grid_search_logreg.best_estimator_
Y_pred_logreg_gs = best_logreg_model.predict(X_test)#giving the best Best Parameters (LogReg

Accuracy_logreg_gs = accuracy_score(Y_test, Y_pred_logreg_gs)
Recall_logreg_gs = recall_score(Y_test, Y_pred_logreg_gs)
Precision_logreg_gs = precision_score(Y_test, Y_pred_logreg_gs)
f1_logreg_gs = f1_score(Y_test, Y_pred_logreg_gs)
```

```
print("GridSearch Logistic Regression Results")
print("Accuracy:", Accuracy_logreg_gs)
print("Recall:", Recall_logreg_gs)
print("Precision:", Precision_logreg_gs)
print("F1-Score:", f1_logreg_gs)
conf_mat_logreg_gs = confusion_matrix(Y_test, Y_pred_logreg_gs)
print("Confusion Matrix:\n", conf_mat_logreg_gs)
```

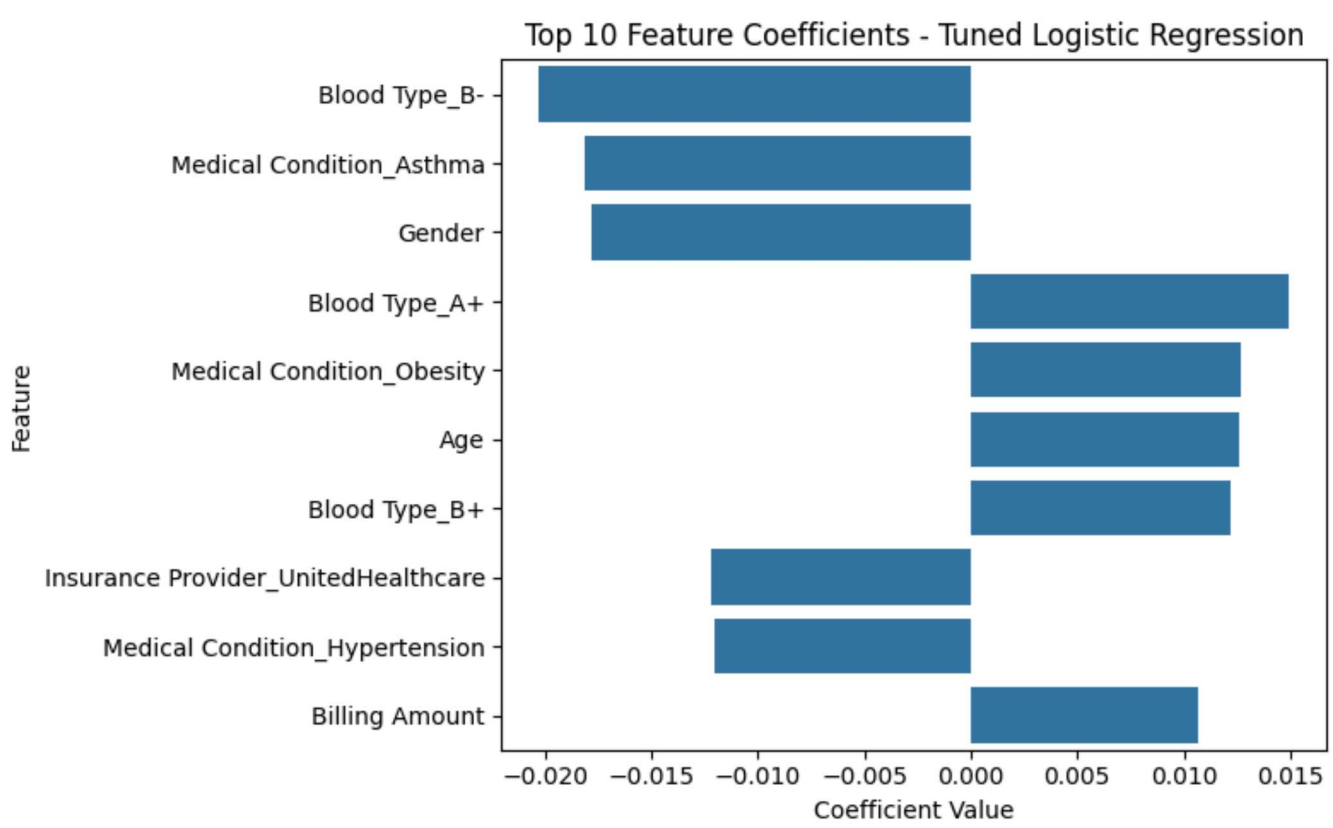
```
GridSearch Logistic Regression Results
Accuracy: 0.5082282282282282
Recall: 0.5165889526542324
Precision: 0.673171301706006
F1-Score: 0.584576357179097
Confusion Matrix:
 [[2701 2797]
 [5391 5761]]
```

graph for loistic

```
plt.figure(figsize=(6,4))
sns.heatmap(conf_mat_logreg_gs, annot=True, fmt='d', cmap='BuPu', xticklabels=["Normal", "Ab
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix - Tuned Logistic Regression')
plt.show()
```



```
coefficients_logreg_gs = pd.Series(best_logreg_model.coef_[0], index=X.columns)
coefficients_logreg_gs = coefficients_logreg_gs.sort_values(key=abs, ascending=False).head(1


plt.figure(figsize=(8,5))
sns.barplot(x=coefficients_logreg_gs.values, y=coefficients_logreg_gs.index)
plt.title("Top 10 Feature Coefficients - Tuned Logistic Regression")
plt.xlabel("Coefficient Value")
plt.ylabel("Feature")
plt.tight_layout()
plt.show()
```



final model comparison bar chart

```
import matplotlib.pyplot as plt
import numpy as np

# Define the models
models = ['Basic DT', 'Tuned DT', 'Random Forest', 'Tuned RF', 'LogReg', 'Tuned L
x = np.arange(len(models))
width = 0.2

# Create lists to store the evaluation metrics for each model
accuracy_vals = [Accuracy, Accuracy_gs, Accuracy_rf, Accuracy_rf_gs, Accuracy_log
precision_vals = [Precision, Precision_gs, Precision_rf, Precision_rf_gs, Precisi
recall_vals = [Recall, Recall_gs, Recall_rf, Recall_rf_gs, Recall_logreg, Recall_
f1_vals = [f1, f1_gs, f1_rf, f1_rf_gs, f1_logreg, f1_logreg_gs]
```

```
# Plotting the bar chart
plt.figure(figsize=(12, 6))
plt.bar(x - 1.5*width, accuracy_vals, width, label='Accuracy', color='steelblue')
```