Hotel Reservations

Giới thiệu bộ dữ liệu:

- Hiện nay có rất nhiều kênh đặt phòng khách sạn trực tuyến, nó khiến cho việc đặt phòng trở nên dễ dàng hơn. Nhưng cũng chính vì thế việc hủy đặt phòng cũng dễ dàng. Những lý do chính cho việc hủy bỏ là thay đổi kế hoạch, xung đột lịch trình,...Việc hủy bỏ làm ảnh hưởng xấu đến doanh thu của khách sạn.
- Bộ dữ liệu bao gồm 19 đặc trưng thông tin chi tiết của khách đặt phòng:
  - **Booking_ID**: đinh danh cá nhân mỗi lần đặt phòng
  - **no_of_adults**: Số lượng người lớn
  - **no_of_children**: Số lượng trẻ em
  - **no_of_weekend_nights**: Số đêm cuối tuần (T7, CN), khách ở lại hoặc đã đặt chỗ.
  - **no_of_week_nights**: Số đêm trong tuần (T2 - T6), khách ở lại hoặc đã đặt chỗ.
  - **type_of_meal_plan**: Loại kế hoạch bữa ăn được đặt bởi khách hàng.
  - **required_car_parking_space**: Khách hàng có yêu cầu chỗ đặt xe hơi hay không? (0 - No, 1 - Yes)
  - **room_type_reserved**: Loại phòng khách hàng đặt
  - **lead_time**: Số ngày giữa ngày đặt phòng và ngày đến.
  - **arrival_year**: Năm đến.
  - **arrival_month**: Tháng đến.
  - **arrival_date**: Ngày đến.
  - **market_segment_type**: Loại phân khúc thị trường.
  - **repeated_guest**: Có phải là khách cũ hay không? (0 - No, 1 - Yes)
  - **no_of_previous_cancellations**: Số lượng đặt phòng trước đó đã bị khách hàng hủy bỏ trước khi đặt phòng hiện tại.
  - **no_of_previous_bookings_not_canceled**: Số lượng đặt phòng trước đó không bị khách hàng hủy bỏ trước khi đặt phòng hiện tại.
  - **avg_price_per_room**: Giá trung bình mỗi ngày đặt phòng, giá phòng là linh động.
  - **no_of_special_requests**: Số lượng yêu cầu đặc biệt của khách hàng (VD: tầng trên cao, view biển,...).
  - **booking_status**: Trạng thái cho biết phòng đó bị hủy hay không?
- Nguồn dữ liệu: **Kaggle**

Tiền xử lý dữ liệu:

```
# Thư viện sử dụng
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from yellowbrick.classifier import ConfusionMatrix
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import r2_score
from sklearn import metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.model_selection import GridSearchCV
# Đọc dữ liệu
df = pd.read_csv("Hotel Reservations.csv")
df.head()
```

	Booking_ID	no_of_adults	no_of_children	no_of_weekend_nights	no_of_week
0	INN00001	2	0	1	
1	INN00002	2	0	2	
2	INN00003	1	0	2	
3	INN00004	2	0	0	
4	INN00005	2	0	1	

```
# encode những cột categorical từ dạng chuỗi sang dạng số
from sklearn.preprocessing import LabelEncoder

label_encoder_type_of_meal_plan = LabelEncoder()
label_encoder_room_type_reserved = LabelEncoder()
label_encoder_market_segment_type = LabelEncoder()
label_encoder_booking_status = LabelEncoder()

df['type_of_meal_plan'] = label_encoder_type_of_meal_plan.fit_transform(df['type_of_meal_plan'])
df['room_type_reserved'] = label_encoder_room_type_reserved.fit_transform(df['room_type_reserved'])
df['market_segment_type'] = label_encoder_market_segment_type.fit_transform(df['market_segment_type'])
df['booking_status'] = label_encoder_booking_status.fit_transform(df['booking_status']) # 'Canceled': 0, 'Not_Canceled': 1


# Chia tập dữ liệu
X = df.drop(['booking_status','Booking_ID'], axis = 1)
X = X.values
y = df['booking_status']


df['booking_status'].value_counts()
```

```
1    24390
0    11885
Name: booking_status, dtype: int64
```

Vì dữ liệu bị lệch nên ta phải tiến hành cân bằng dữ liệu

```
from imblearn.over_sampling import RandomOverSampler
ros = RandomOverSampler(random_state=0)
X, y = ros.fit_resample(X, y)
y.value_counts()
```

```
1    24390
0    24390
Name: booking_status, dtype: int64
```

Bây giờ dữ liệu đã được cân bằng, tiếp theo ta sẽ chuẩn hóa dữ liệu
Lý do ta cần over sampling class bị thiếu là bởi vì ta không muốn cho các mô hình học máy thiên vị về bên class có nhiều data point hơn, đồng thời cân bằng cho máy học để có thể dự đoán chính xác hơn cho cả 2 class.

`Lợi ích của việc xài over-sampling này là:` tăng performance cho máy học, bias correction, và đỡ tốn thời gian để thu thập thêm dữ liệu mới.

`Tác hại của việc xài over-sampling:` bị Overfiting (có thể data point mới bản gốc của nó), mất mát thông tin(loss of information - data point mới được sinh ra không giữ được những đặc trưng đặc biệt của nhóm ít data hơn)

```
# Chuẩn hóa dữ liệu để đưa dữ liệu về cùng tỉ lệ, tránh phụ thuộc vào 1 đặc trưng nào đó
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_standard = scaler.fit_transform(X)


# Chia dữ liệu
from sklearn.model_selection import train_test_split
X_train,X_test, y_train, y_test = train_test_split(X_standard, y, test_size = 0.2, random_state = 0)
```

Xây dựng Mô hình:

Mô hình 1: Logistic Regression

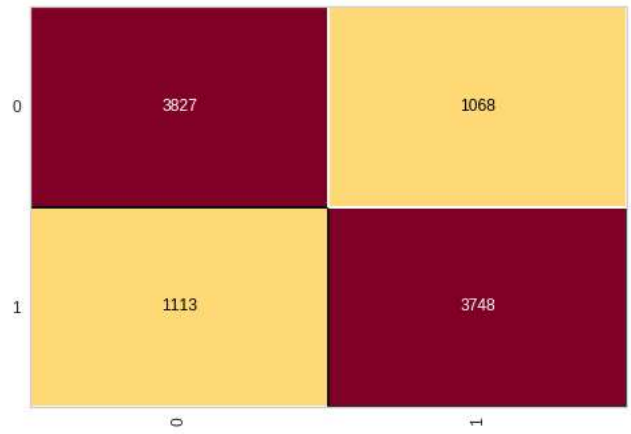
```
# Huấn luyện mô hình
from sklearn.linear_model import LogisticRegression
logistic = LogisticRegression(random_state = 1, max_iter=1000)
logistic.fit(X_train, y_train)
previsoes = logistic.predict(X_test)


# Kiểm tra Overfitting
print("Độ chính xác trên tập Train",accuracy_score(logistic.predict(X_train), y_train))
```

```
print("Độ chính xác trên tập Test",accuracy_score(logistic.predict(X_test), y_test))
```

```
Độ chính xác trên tập Train 0.7762658876588766
Độ chính xác trên tập Test 0.7764452644526445
```

```
# Đánh giá confusion matrix
cm = ConfusionMatrix(logistic)
cm.fit(X_train, y_train)

print("Accuracy =",cm.score(X_test, y_test))
```

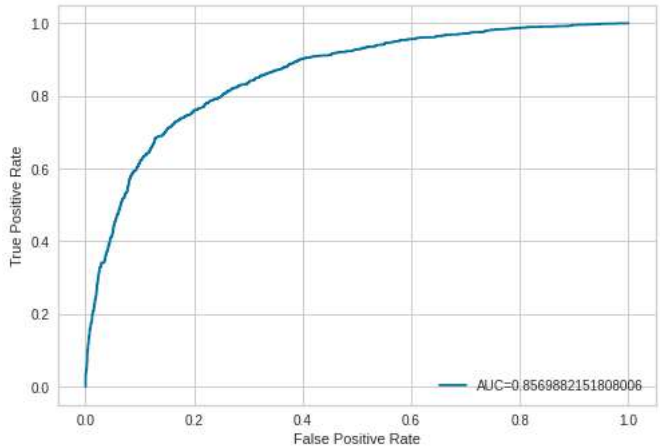
```
Accuracy = 0.7764452644526445
```




```
print(classification_report(y_test, previsoes))
```

```
                  precision    recall  f1-score   support

               0       0.77      0.78      0.78      4895
               1       0.78      0.77      0.77      4861

        accuracy                           0.78      9756
       macro avg       0.78      0.78      0.78      9756
    weighted avg       0.78      0.78      0.78      9756
```

```
#define metrics
y_pred_proba = logistic.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Nhận xét:

- Mô hình không bị overfitting
- Độ chính xác của mô hình là 0.776 => Mô hình dự đoán khá tốt
- Ta thấy chỉ số AUC (Are under curve) = 0.857 cho thấy model phân loại khá chính xác giữa 2 lớp với nhau

Mô hình 2: Decision Tree

```
min_split = np.array([2, 3, 4, 5, 6, 7])
max_nvl = np.array([3, 4, 5, 6, 7, 9, 11])
alg = ['entropy', 'gini']
values_grid = {'min_samples_split': min_split, 'max_depth': max_nvl, 'criterion': alg}
# Tìm kiếm siêu tham số cho mô hình
model = DecisionTreeClassifier()
gridDecisionTree = GridSearchCV(estimator = model, param_grid = values_grid, cv = 5)
gridDecisionTree.fit(X_train, y_train)
# Các siêu tham số tối ưu của mô hình
print('Min Split: ', gridDecisionTree.best_estimator_.min_samples_split)
print('Max Nvl: ', gridDecisionTree.best_estimator_.max_depth)
print('Algorithm: ', gridDecisionTree.best_estimator_.criterion)
print('Score: ', gridDecisionTree.best_score_)
```

```
Min Split:  3
Max Nvl:  11
Algorithm:  gini
Score:  0.8646730745677818
```

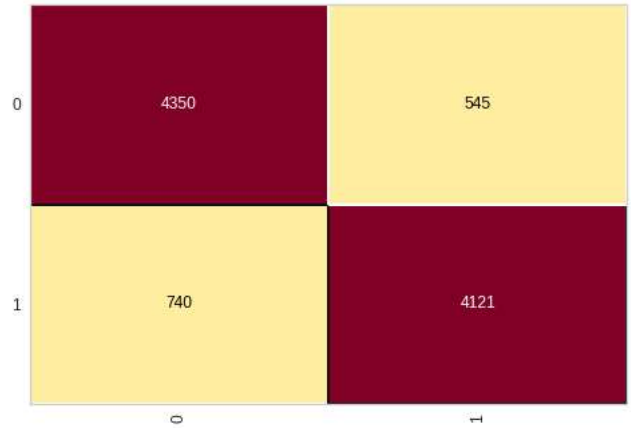
```
# Huấn luyện và dự đoán
decision_tree = DecisionTreeClassifier(criterion = 'gini', min_samples_split = 2, max_depth= 11, random_state=0)
decision_tree.fit(X_train, y_train)
previsoes = decision_tree.predict(X_test)
```

```
# Kiểm tra Overfitting
print("Độ chính xác trên tập Train",accuracy_score(decision_tree.predict(X_train), y_train))
print("Độ chính xác trên tập Test",accuracy_score(decision_tree.predict(X_test), y_test))
```

```
Độ chính xác trên tập Train 0.8814063140631406
Độ chính xác trên tập Test 0.8682861828618286
```

```
cm = ConfusionMatrix(decision_tree)
cm.fit(X_train, y_train)
print("Accuracy =",cm.score(X_test, y_test))
```

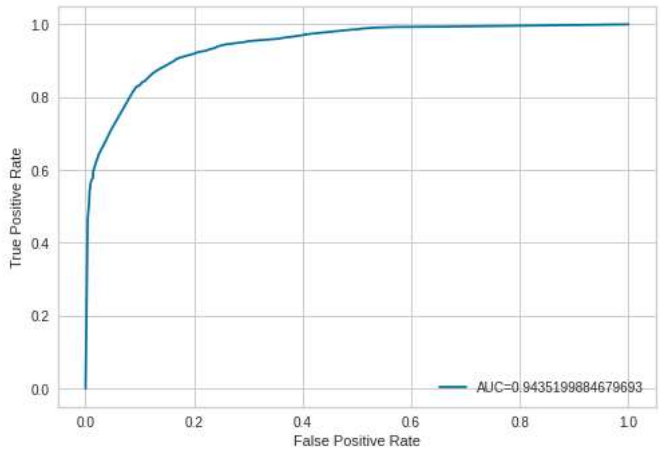
```
Accuracy = 0.8682861828618286
```




```
print(classification_report(y_test, previsoes))
```

```
                  precision    recall  f1-score   support

               0       0.85      0.89      0.87      4895
               1       0.88      0.85      0.87      4861

        accuracy                           0.87      9756
       macro avg       0.87      0.87      0.87      9756
    weighted avg       0.87      0.87      0.87      9756
```

```
#define metrics
y_pred_proba = decision_tree.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Nhận xét:

- Mô hình không bị overfitting
- Độ chính xác của mô hình là 0.868 => Mô hình dự đoán tốt
- Ta nhận thấy AUC ở decision tree cho ra dự đoán chính xác tốt hơn ở 2 lớp, cụ thể là AUC = 0.94%

Mô hình 3: Random Forest

```
from sklearn.ensemble import RandomForestClassifier
# Khởi tạo các giá trị siêu tham số
n_estimators = np.array([100])
alg = ['entropy', 'gini']
min_split = np.array([2, 3, 4, 5, 6, 7])
max_nvl = np.array([3, 4, 5, 6, 7, 9, 11])
values_grid = {'n_estimators': n_estimators, 'min_samples_split': min_split, 'max_depth': max_nvl, 'criterion': alg}
# Tìm siêu tham số tối ưu nhất cho mô hình
model = RandomForestClassifier()
gridRandomForest = GridSearchCV(estimator = model, param_grid = values_grid, cv = 5)
gridRandomForest.fit(X_train, y_train)
# Các siêu tham số tìm được
print('Algorithm: ', gridRandomForest.best_estimator_.criterion)
print('Score: ', gridRandomForest.best_score_)
print('Min Split: ', gridRandomForest.best_estimator_.min_samples_split)
print('Max Nvl: ', gridRandomForest.best_estimator_.max_depth)
```

```
Algorithm:  gini
Score:  0.8716686756344009
Min Split:  4
Max Nvl:  11
```

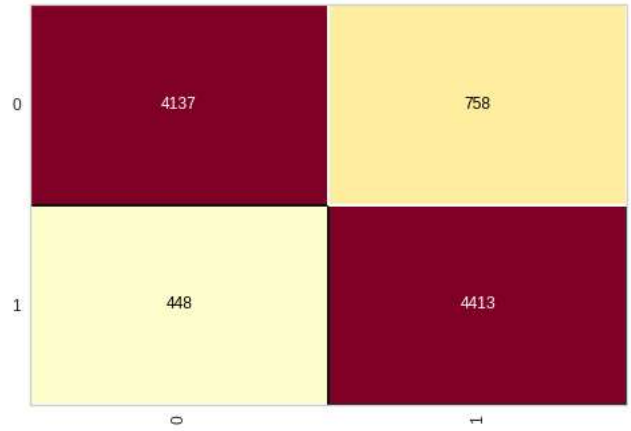
```
random_forest = RandomForestClassifier(n_estimators = 100, min_samples_split = 2, max_depth= 11,  criterion = 'gini', random_state = 0)
random_forest.fit(X_train, y_train)
previsoes = random_forest.predict(X_test)
```

```
# Kiểm tra Overfitting
print("Độ chính xác trên tập Train",accuracy_score(random_forest.predict(X_train), y_train))
print("Độ chính xác trên tập Test",accuracy_score(random_forest.predict(X_test), y_test))
```

```
Độ chính xác trên tập Train 0.8807913079130791
Độ chính xác trên tập Test 0.8763837638376384
```

```
cm = ConfusionMatrix(random_forest)
cm.fit(X_train, y_train)
print("Accuracy =",cm.score(X_test, y_test))
```

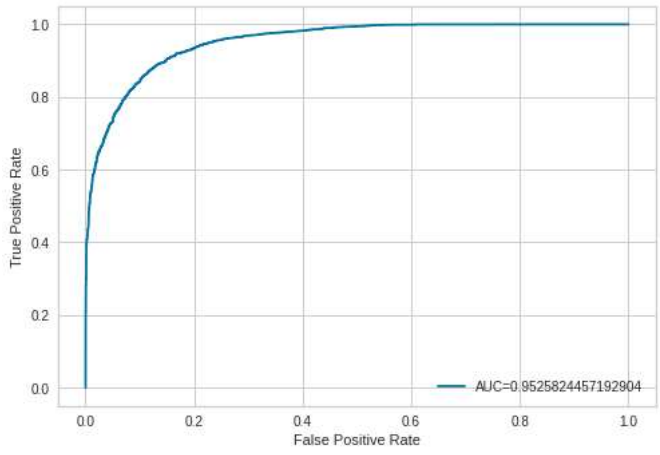
```
Accuracy = 0.8763837638376384
```




```
print(classification_report(y_test, previsoes))
```

```
                  precision    recall  f1-score   support

               0       0.90      0.85      0.87      4895
               1       0.85      0.91      0.88      4861

        accuracy                           0.88      9756
       macro avg       0.88      0.88      0.88      9756
    weighted avg       0.88      0.88      0.88      9756
```

```
#define metrics
y_pred_proba = random_forest.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Nhận xét:

- Mô hình không bị overfitting
- Độ chính xác của mô hình là 0.876 => Mô hình hoạt động tốt
- AUC = 95.2%, là model có chỉ số cao nhất cho thấy khi model dự đoán là positive thì khả năng là true positive lên tận 95.2%, chỉ 4.8% còn lại là khả năng sẽ bị False positive khi mô hình dự đoán là positive

Mô hình 4: KNN

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()

k_list = list(range(1,10))
k_values = dict(n_neighbors = k_list)
grid = GridSearchCV(knn, k_values, cv = 2, scoring = 'accuracy', n_jobs = -1)
grid.fit(X_train, y_train)

print("n_neighbors tốt nhất:",grid.best_params_)
```

```
n_neighbors tốt nhất: {'n_neighbors': 1}
```

```
knn = KNeighborsClassifier(n_neighbors = 1, metric = 'minkowski', p = 2)
knn.fit(X_train, y_train)
previsoes = knn.predict(X_test)


# Kiểm tra Overfitting
print("Độ chính xác trên tập Train",accuracy_score(knn.predict(X_train), y_train))
print("Độ chính xác trên tập Test",accuracy_score(knn.predict(X_test), y_test))
```

```
Độ chính xác trên tập Train 0.997360598605986
Độ chính xác trên tập Test 0.9074415744157441
```

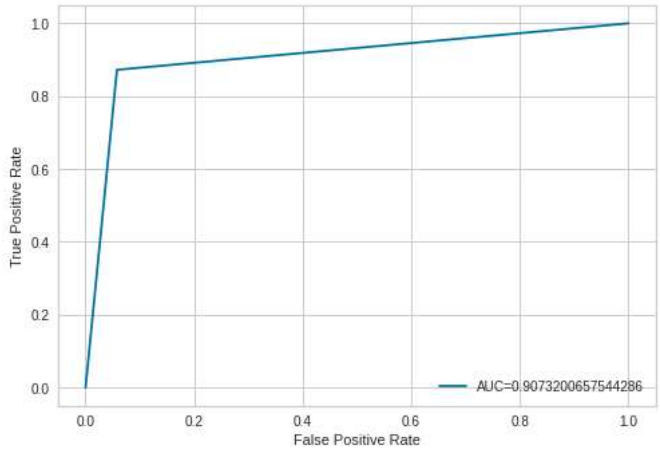
```
cm = ConfusionMatrix(knn)
cm.fit(X_train, y_train)
print("Accuracy =",cm.score(X_test, y_test))
```

```
Accuracy = 0.9074415744157441
```


```
print(classification_report(y_test, previsoes))
```

```
                  precision    recall  f1-score   support

               0       0.88      0.94      0.91      4895
               1       0.94      0.87      0.90      4861

        accuracy                           0.91      9756
       macro avg       0.91      0.91      0.91      9756
    weighted avg       0.91      0.91      0.91      9756
```

```
#define metrics
y_pred_proba = knn.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Nhận xét:

- Mô hình không bị overfitting
- Độ chính xác của mô hình là 0.907 => Mô hình dự đoán rất tốt
- Với AUC=98% cũng khá cao cho mô hình cơ bản như KNN

Mô hình 5: Support Vector Machines (SVMs):

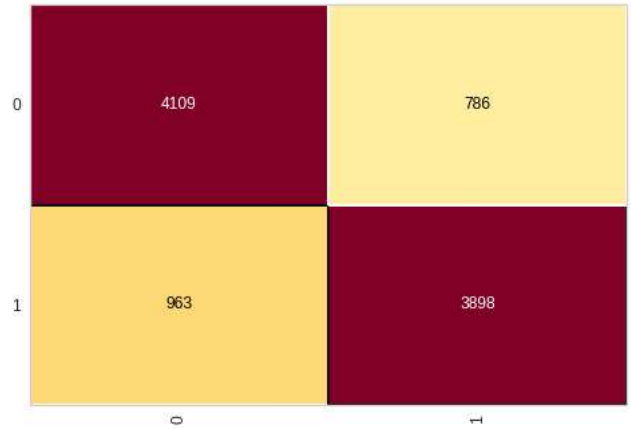
```
from sklearn import svm

SVM = svm.SVC(kernel='poly', C=1.0, probability=True)
SVM.fit(X_train, y_train)
previsoes = SVM.predict(X_test)
```

```
# Kiểm tra Overfitting
print("Độ chính xác trên tập Train",accuracy_score(SVM.predict(X_train), y_train))
print("Độ chính xác trên tập Test",accuracy_score(SVM.predict(X_test), y_test))
```

```
Độ chính xác trên tập Train 0.8194188191881919
Độ chính xác trên tập Test 0.8207257072570726
```

```
cm = ConfusionMatrix(SVM)
cm.fit(X_train, y_train)
print("Accuracy =",cm.score(X_test, y_test))
```

```
Accuracy = 0.8207257072570726
```

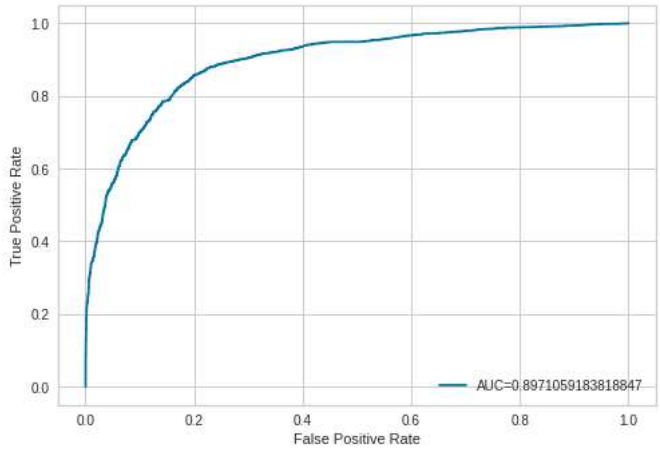



```
print(classification_report(y_test, previsoes))
```

```
              precision    recall  f1-score   support

           0       0.81      0.84      0.82      4895
           1       0.83      0.80      0.82      4861
```

```
              accuracy                           0.82      9756
             macro avg       0.82      0.82      0.82      9756
          weighted avg       0.82      0.82      0.82      9756
```

```
#define metrics
y_pred_proba = SVM.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Nhận xét:

- Mô hình không bị overfitting
- Độ chính xác của mô hình là 0.820 => Mô hình dự đoán tốt
- AUC=89.7%, ta thấy thấp hơn so với 1 số mô hình trên.

Mô hình 6: Naive Bayes

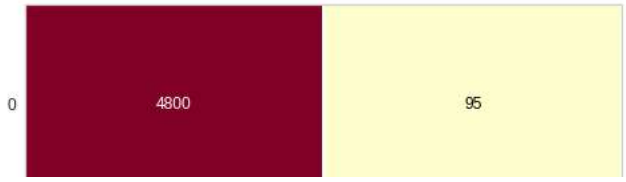
```
from sklearn.naive_bayes import GaussianNB
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
previsoes = naive_bayes.predict(X_test)
```

```
# Kiểm tra Overfitting
print("Độ chính xác trên tập Train",accuracy_score(naive_bayes.predict(X_train), y_train))
print("Độ chính xác trên tập Test",accuracy_score(naive_bayes.predict(X_test), y_test))
```

```
Độ chính xác trên tập Train 0.5670612956129562
Độ chính xác trên tập Test 0.5712382123821238
```

```
cm = ConfusionMatrix(naive_bayes)
cm.fit(X_train, y_train)
print("Accuracy =",cm.score(X_test, y_test))
```

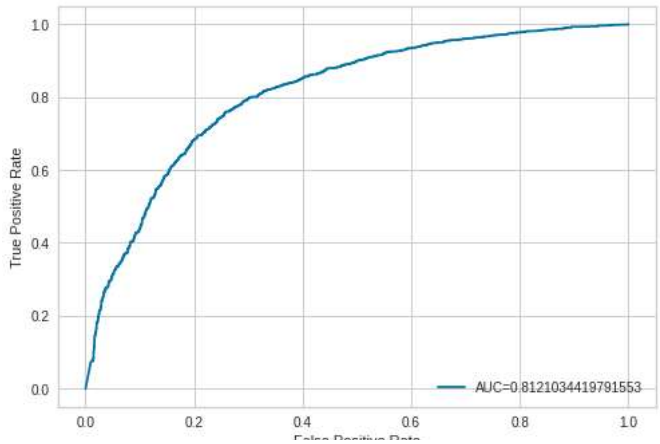
```
Accuracy = 0.5712382123821238
```




```
print(classification_report(y_test, previsoes))
```

```
                  precision    recall  f1-score   support

               0       0.54      0.98      0.70      4895
               1       0.89      0.16      0.27      4861

        accuracy                           0.57      9756
       macro avg       0.72      0.57      0.48      9756
    weighted avg       0.71      0.57      0.48      9756
```

```
#define metrics
y_pred_proba = naive_bayes.predict_proba(X_test)[::,1]
fpr, tpr, _ = metrics.roc_curve(y_test,  y_pred_proba)
auc = metrics.roc_auc_score(y_test, y_pred_proba)

#create ROC curve
plt.plot(fpr,tpr,label="AUC="+str(auc))
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.legend(loc=4)
plt.show()
```



Nhận xét:

- Mô hình không bị overfitting
- Độ chính xác của mô hình là 0.571 => Mô hình dự đoán không tốt
- Ở model này AUC=0.812, thấp nhất trong số các model trên

Mô hình 7: Neural Network

```
from keras.utils import np_utils
y_convert = np_utils.to_categorical(y)
X_train, X_test, y_train, y_test = train_test_split(X_standard, y_convert, test_size = 0.3, random_state = 0)
```

```
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras import regularizers
# Tạo mô hình neural networks rỗng
model = Sequential()
```

```
# Thêm các lớp Layer
model.add(Dense(200, input_dim = 17, kernel_initializer = 'normal', activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(100, kernel_initializer = 'normal', activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(2, kernel_initializer = 'normal', activation = 'softmax'))


from tensorflow.keras.optimizers import Adam
optimitzer = Adam(amsgrad=True)

# Tổng hợp mô hình Neural networks để huấn luyện
model.compile(loss='categorical_crossentropy', optimizer = optimitzer, metrics=['acc'])

# Huấn luyện mô hình
model.fit(X_train, y_train, epochs = 100, batch_size = 400, validation_data = (X_test, y_test), verbose = 1)
```

```
86/86 [==============================] - 1s 7ms/step - loss: 0.3443 - acc: 0.8449 - val_loss: 0.3429 - val_acc: 0.8478
Epoch 23/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3415 - acc: 0.8449 - val_loss: 0.3412 - val_acc: 0.8491
Epoch 24/100
86/86 [==============================] - 1s 8ms/step - loss: 0.3411 - acc: 0.8472 - val_loss: 0.3376 - val_acc: 0.8498
Epoch 25/100
86/86 [==============================] - 1s 8ms/step - loss: 0.3389 - acc: 0.8474 - val_loss: 0.3335 - val_acc: 0.8529
Epoch 26/100
86/86 [==============================] - 1s 8ms/step - loss: 0.3380 - acc: 0.8476 - val_loss: 0.3336 - val_acc: 0.8525
Epoch 27/100
86/86 [==============================] - 1s 7ms/step - loss: 0.3358 - acc: 0.8486 - val_loss: 0.3329 - val_acc: 0.8526
Epoch 28/100
86/86 [==============================] - 1s 7ms/step - loss: 0.3331 - acc: 0.8498 - val_loss: 0.3289 - val_acc: 0.8548
Epoch 29/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3315 - acc: 0.8518 - val_loss: 0.3307 - val_acc: 0.8517
Epoch 30/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3312 - acc: 0.8513 - val_loss: 0.3258 - val_acc: 0.8564
Epoch 31/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3295 - acc: 0.8534 - val_loss: 0.3251 - val_acc: 0.8586
Epoch 32/100
86/86 [==============================] - 1s 7ms/step - loss: 0.3271 - acc: 0.8510 - val_loss: 0.3244 - val_acc: 0.8579
Epoch 33/100
86/86 [==============================] - 1s 7ms/step - loss: 0.3266 - acc: 0.8527 - val_loss: 0.3265 - val_acc: 0.8544
Epoch 34/100
86/86 [==============================] - 1s 8ms/step - loss: 0.3237 - acc: 0.8544 - val_loss: 0.3230 - val_acc: 0.8567
Epoch 35/100
86/86 [==============================] - 1s 8ms/step - loss: 0.3234 - acc: 0.8554 - val_loss: 0.3226 - val_acc: 0.8605
Epoch 36/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3232 - acc: 0.8546 - val_loss: 0.3220 - val_acc: 0.8581
Epoch 37/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3222 - acc: 0.8554 - val_loss: 0.3199 - val_acc: 0.8592
Epoch 38/100
86/86 [==============================] - 1s 9ms/step - loss: 0.3212 - acc: 0.8567 - val_loss: 0.3178 - val_acc: 0.8601
Epoch 39/100
86/86 [==============================] - 1s 9ms/step - loss: 0.3183 - acc: 0.8585 - val_loss: 0.3165 - val_acc: 0.8615
Epoch 40/100
86/86 [==============================] - 1s 8ms/step - loss: 0.3164 - acc: 0.8579 - val_loss: 0.3145 - val_acc: 0.8611
Epoch 41/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3165 - acc: 0.8581 - val_loss: 0.3162 - val_acc: 0.8627
Epoch 42/100
86/86 [==============================] - 1s 7ms/step - loss: 0.3147 - acc: 0.8592 - val_loss: 0.3184 - val_acc: 0.8603
Epoch 43/100
86/86 [==============================] - 1s 8ms/step - loss: 0.3137 - acc: 0.8583 - val_loss: 0.3140 - val_acc: 0.8607
Epoch 44/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3126 - acc: 0.8611 - val_loss: 0.3137 - val_acc: 0.8617
Epoch 45/100
86/86 [==============================] - 1s 7ms/step - loss: 0.3126 - acc: 0.8607 - val_loss: 0.3108 - val_acc: 0.8605
Epoch 46/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3120 - acc: 0.8622 - val_loss: 0.3116 - val_acc: 0.8622
Epoch 47/100
86/86 [==============================] - 1s 7ms/step - loss: 0.3080 - acc: 0.8636 - val_loss: 0.3101 - val_acc: 0.8614
Epoch 48/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3112 - acc: 0.8626 - val_loss: 0.3110 - val_acc: 0.8615
Epoch 49/100
86/86 [==============================] - 1s 6ms/step - loss: 0.3089 - acc: 0.8611 - val_loss: 0.3079 - val_acc: 0.8629
Epoch 50/100
86/86 [==============================] - 1s 8ms/step - loss: 0.3074 - acc: 0.8630 - val_loss: 0.3092 - val_acc: 0.8618
Epoch 51/100
```

Nhận xét:

- Mô hình không bị overfitting
- Độ chính xác của mô hình là 0.875 => Mô hình dự đoán tốt

Phần kết luận:

No.	Model	Type	Precision
1	Logistic Regression	-	0.776
2	Decision Tree	Gini	0.868
3	Random Forest	Gini	0.871
4	KNN	-	0.907
5	SVM	-	0.820
6	Naive Bayes	Gaussian	0.571
7	Neural Network	-	0.875

Nhận xét:

- Mô hình không bị overfitting
- Mặc dù mô hình Logistic Regression có ưu điểm để phân loại dữ liệu nhị phân nhưng kết quả lại không tốt hơn các mô hình khác.
- Mô hình có kết quả tốt nhất là mô hình KNN
- Mô hình có kết quả tệ nhất là mô hình Naive Bayes