import numpy as np   
import pandas as pd   
  
from sklearn import preprocessing  
import matplotlib.pyplot as plt   
plt.rc("font", size=14)  
import seaborn as sns  
sns.set(style="white") #white background style for seaborn plots  
sns.set(style="whitegrid", color\_codes=True)  
  
import warnings  
warnings.simplefilter(action='ignore')

* Khai báo thư viện
* Thay đổi nền cho đồ thị seaborn
* Khai báo thư viện warning(không có cảnh báo nào sẽ được in ra từ bây giờ)

train\_df = pd.read\_csv("input/train.csv")

* Đọc dữ liệu train trong DataFrame

test\_df = pd.read\_csv("input/test.csv")

* Đọc dữ liệu test trong DataFrame

train\_df.head()

PassengerId Survived Pclass ... Fare Cabin Embarked  
0 1 0 3 ... 7.2500 NaN S  
1 2 1 1 ... 71.2833 C85 C  
2 3 1 3 ... 7.9250 NaN S  
3 4 1 1 ... 53.1000 C123 S  
4 5 0 3 ... 8.0500 NaN S  
  
[5 rows x 12 columns]

* Xem trước dữ liệu train

print('The number of samples into the train data is {}.'.format(train\_df.shape[0]))

The number of samples into the train data is 891.

test\_df.head()

PassengerId Pclass ... Cabin Embarked  
0 892 3 ... NaN Q  
1 893 3 ... NaN S  
2 894 2 ... NaN Q  
3 895 3 ... NaN S  
4 896 3 ... NaN S  
  
[5 rows x 11 columns]

* Xem trước dữ liệu test

print('The number of samples into the test data is {}.'.format(test\_df.shape[0]))

The number of samples into the test data is 418.

train\_df.isnull().sum()

PassengerId 0  
Survived 0  
Pclass 0  
Name 0  
Sex 0  
Age 177  
SibSp 0  
Parch 0  
Ticket 0  
Fare 0  
Cabin 687  
Embarked 2  
dtype: int64

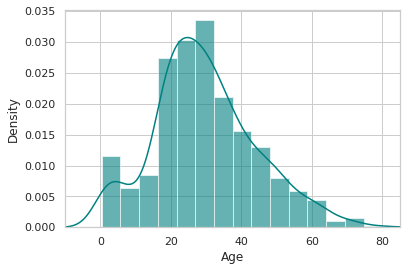
* Kiểm tra các giá trị bị thiếu trong dữ liệu train

print('Percent of missing "Age" records is %.2f%%' %((train\_df['Age'].isnull().sum()/train\_df.shape[0])\*100))

Percent of missing "Age" records is 19.87%

* Phần trăm "Age" bị thiếu

ax = train\_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)  
train\_df["Age"].plot(kind='density', color='teal')  
ax.set(xlabel='Age')  
plt.xlim(-10,85)  
plt.show()



* Hiện thị đồ thị với dữ liệu ở cột "Age"

# mean age  
print('The mean of "Age" is %.2f' %(train\_df["Age"].mean(skipna=True)))  
# median age  
print('The median of "Age" is %.2f' %(train\_df["Age"].median(skipna=True)))

The mean of "Age" is 29.70  
The median of "Age" is 28.00

* Giá trị trung bình của "Age" là 29.70
* Giá trị giữa của "Age" là 28.00

print('Percent of missing "Cabin" records is %.2f%%' %((train\_df['Cabin'].isnull().sum()/train\_df.shape[0])\*100))

Percent of missing "Cabin" records is 77.10%

* Phần trăm "Cabin" bị thiếu là 77.10%

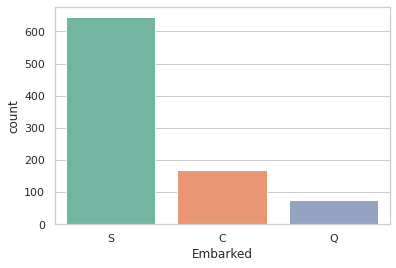
print('Percent of missing "Embarked" records is %.2f%%' %((train\_df['Embarked'].isnull().sum()/train\_df.shape[0])\*100))

Percent of missing "Embarked" records is 0.22%

* Phần trăm "Embarked" bị thiếu là 77.10%

print('Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton):')  
print(train\_df['Embarked'].value\_counts())  
sns.countplot(x='Embarked', data=train\_df, palette='Set2')  
plt.show()

Boarded passengers grouped by port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton):  
S 644  
C 168  
Q 77  
Name: Embarked, dtype: int64



* Hiển thị đồ thị đếm của "Emabarked"

print('The most common boarding port of embarkation is %s.' %train\_df['Embarked'].value\_counts().idxmax())

The most common boarding port of embarkation is S.

train\_data = train\_df.copy()  
train\_data["Age"].fillna(train\_df["Age"].median(skipna=True), inplace=True)  
train\_data["Embarked"].fillna(train\_df['Embarked'].value\_counts().idxmax(), inplace=True)  
train\_data.drop('Cabin', axis=1, inplace=True)

* Điều chỉnh lại dữ liệu train

train\_data.isnull().sum()

PassengerId 0  
Survived 0  
Pclass 0  
Name 0  
Sex 0  
Age 0  
SibSp 0  
Parch 0  
Ticket 0  
Fare 0  
Embarked 0  
dtype: int64

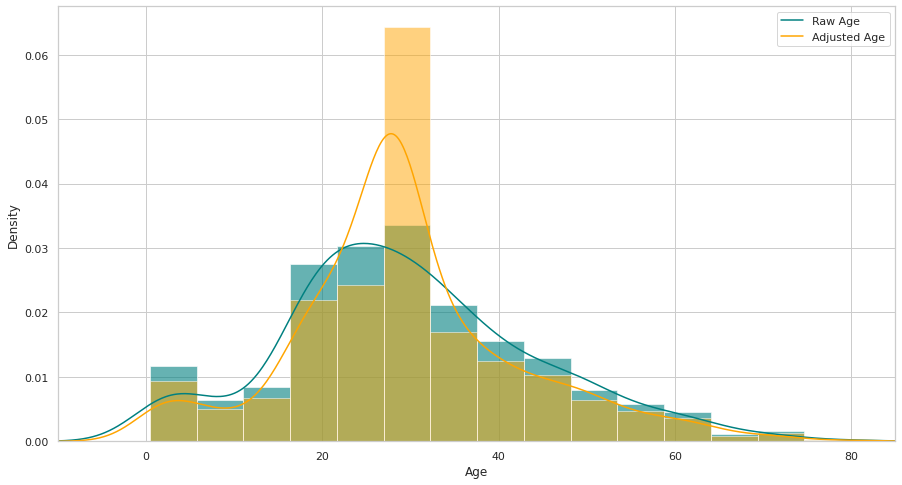
* Kiểm tra các giá trị còn thiếu trong dữ liệu train đã điều chỉnh

train\_data.head()

PassengerId Survived Pclass ... Ticket Fare Embarked  
0 1 0 3 ... A/5 21171 7.2500 S  
1 2 1 1 ... PC 17599 71.2833 C  
2 3 1 3 ... STON/O2. 3101282 7.9250 S  
3 4 1 1 ... 113803 53.1000 S  
4 5 0 3 ... 373450 8.0500 S  
  
[5 rows x 11 columns]

* Xem trước dữ liệu đã điều chỉnh

plt.figure(figsize=(15,8))  
ax = train\_df["Age"].hist(bins=15, density=True, stacked=True, color='teal', alpha=0.6)  
train\_df["Age"].plot(kind='density', color='teal')  
ax = train\_data["Age"].hist(bins=15, density=True, stacked=True, color='orange', alpha=0.5)  
train\_data["Age"].plot(kind='density', color='orange')  
ax.legend(['Raw Age', 'Adjusted Age'])  
ax.set(xlabel='Age')  
plt.xlim(-10,85)  
plt.show()



* Hiển thị đồ thị "Age" ban đầu và đồ thị "Age" vừa hiệu chỉnh

train\_data["TravelAlone"]=np.where((train\_data["SibSp"]+train\_data["Parch"])>0, 0, 1)  
train\_data.drop('SibSp', axis=1, inplace=True)  
train\_data.drop('Parch', axis=1, inplace=True)

* Tạo dữ liệu "TravelAlone"

training=pd.get\_dummies(train\_data, columns=["Pclass","Embarked","Sex"])  
training.drop('Sex\_female', axis=1, inplace=True)  
training.drop('PassengerId', axis=1, inplace=True)  
training.drop('Name', axis=1, inplace=True)  
training.drop('Ticket', axis=1, inplace=True)  
  
final\_train = training  
final\_train.head()

Survived Age Fare ... Embarked\_Q Embarked\_S Sex\_male  
0 0 22.0 7.2500 ... 0 1 1  
1 1 38.0 71.2833 ... 0 0 0  
2 1 26.0 7.9250 ... 0 1 0  
3 1 35.0 53.1000 ... 0 1 0  
4 0 35.0 8.0500 ... 0 1 1  
  
[5 rows x 11 columns]

* Tạo biến phân loại và loại bỏ một số biến

test\_df.isnull().sum()

PassengerId 0  
Pclass 0  
Name 0  
Sex 0  
Age 86  
SibSp 0  
Parch 0  
Ticket 0  
Fare 1  
Cabin 327  
Embarked 0  
dtype: int64

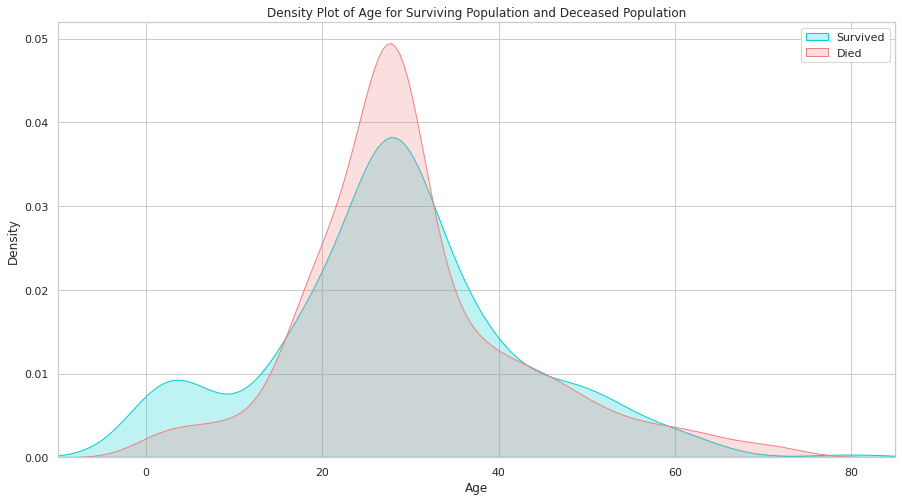
* Kiểm tra dữ liệu rỗng

test\_data = test\_df.copy()  
test\_data['Age'].fillna(train\_df["Age"].median(skipna=True), inplace=True)  
test\_data["Fare"].fillna(train\_df["Fare"].median(skipna=True), inplace=True)  
test\_data.drop('Cabin', axis=1, inplace=True)  
  
test\_data['TravelAlone']=np.where((test\_data["SibSp"]+test\_data["Parch"])>0, 0, 1)  
  
test\_data.drop('SibSp', axis=1, inplace=True)  
test\_data.drop('Parch', axis=1, inplace=True)  
  
testing = pd.get\_dummies(test\_data, columns=["Pclass","Embarked","Sex"])  
testing.drop('Sex\_female', axis=1, inplace=True)  
testing.drop('PassengerId', axis=1, inplace=True)  
testing.drop('Name', axis=1, inplace=True)  
testing.drop('Ticket', axis=1, inplace=True)  
  
final\_test = testing  
final\_test.head()

Age Fare TravelAlone ... Embarked\_Q Embarked\_S Sex\_male  
0 34.5 7.8292 1 ... 1 0 1  
1 47.0 7.0000 0 ... 0 1 0  
2 62.0 9.6875 1 ... 1 0 1  
3 27.0 8.6625 1 ... 0 1 1  
4 22.0 12.2875 0 ... 0 1 0  
  
[5 rows x 10 columns]

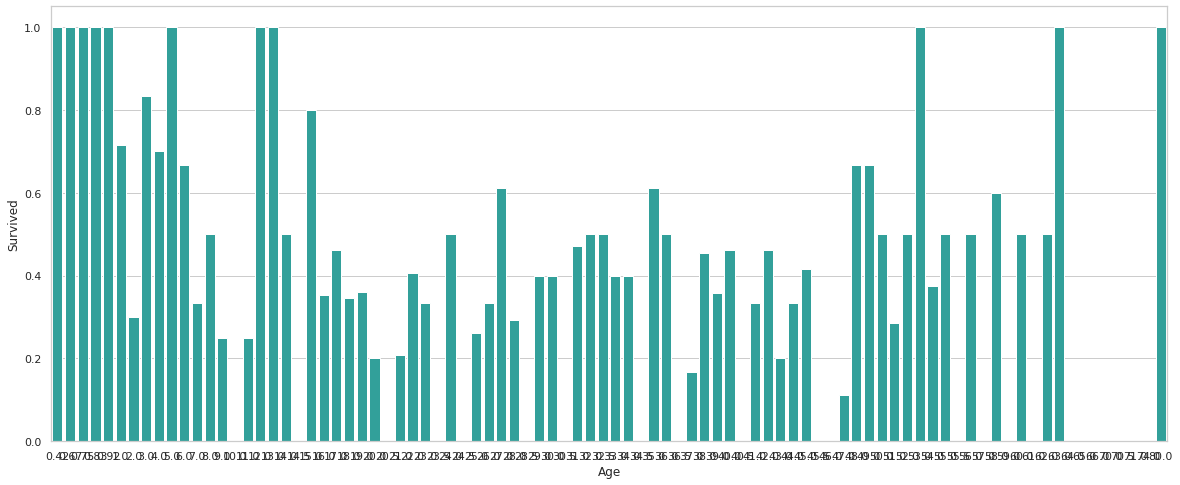
* Kiểm tra dữ liệu

plt.figure(figsize=(15,8))  
ax = sns.kdeplot(final\_train["Age"][final\_train.Survived == 1], color="darkturquoise", shade=True)  
sns.kdeplot(final\_train["Age"][final\_train.Survived == 0], color="lightcoral", shade=True)  
plt.legend(['Survived', 'Died'])  
plt.title('Density Plot of Age for Surviving Population and Deceased Population')  
ax.set(xlabel='Age')  
plt.xlim(-10,85)  
plt.show()



* Hiển thị đồ thị của 2 dòng dữ liệu "Survived" và "Died"

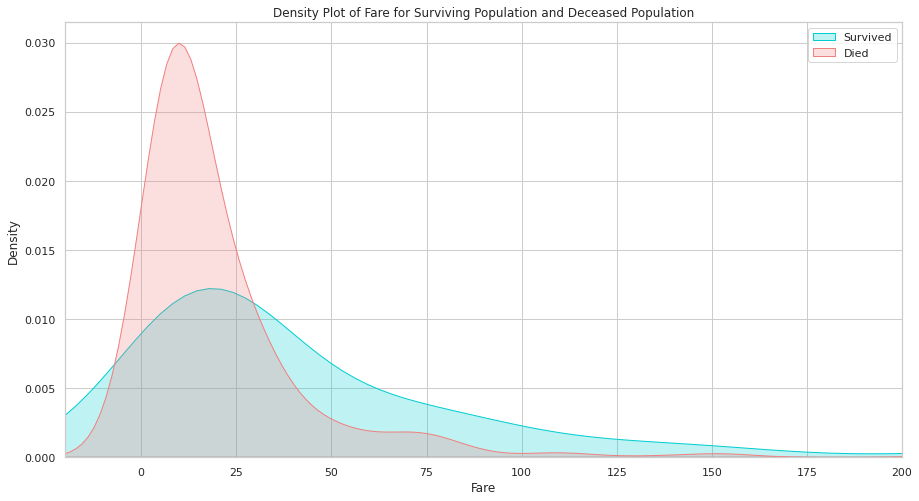
plt.figure(figsize=(20,8))  
avg\_survival\_byage = final\_train[["Age", "Survived"]].groupby(['Age'], as\_index=False).mean()  
g = sns.barplot(x='Age', y='Survived', data=avg\_survival\_byage, color="LightSeaGreen")  
plt.show()



* Hiển thị đồ thị giá trị trung bình với x là Age và Y là Survived, gom nhóm "Age" lại

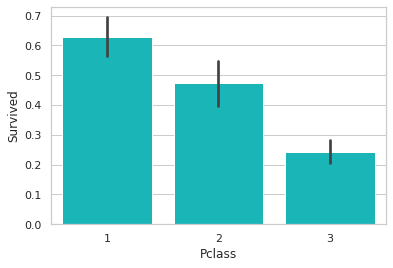
final\_train['IsMinor']=np.where(final\_train['Age']<=16, 1, 0)  
  
final\_test['IsMinor']=np.where(final\_test['Age']<=16, 1, 0)

plt.figure(figsize=(15,8))  
ax = sns.kdeplot(final\_train["Fare"][final\_train.Survived == 1], color="darkturquoise", shade=True)  
sns.kdeplot(final\_train["Fare"][final\_train.Survived == 0], color="lightcoral", shade=True)  
plt.legend(['Survived', 'Died'])  
plt.title('Density Plot of Fare for Surviving Population and Deceased Population')  
ax.set(xlabel='Fare')  
plt.xlim(-20,200)  
plt.show()



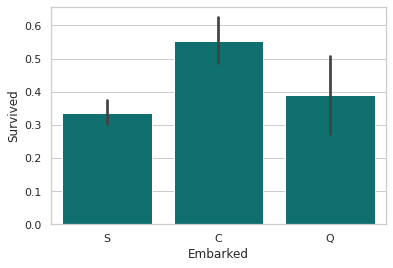
* Hiển thị đồ thị "Survived" và "Died"

sns.barplot('Pclass', 'Survived', data=train\_df, color="darkturquoise")  
plt.show()



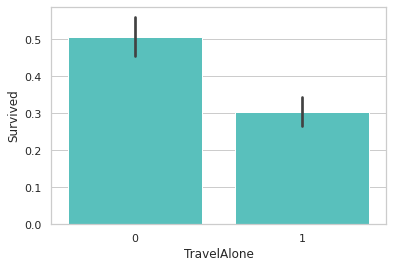
* Hiển thị đồ thị dạng cột với trục tung là "Survived", trục hoành là "Pclass"

sns.barplot('Embarked', 'Survived', data=train\_df, color="teal")  
plt.show()



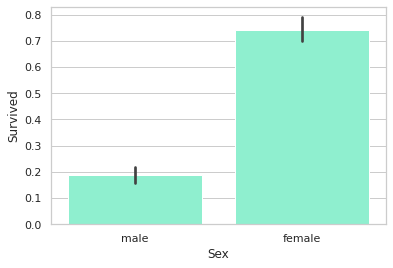
* Hiển thị đồ thị dạng cột với trục tung là "Survived", trục hoành là "Embarked"

sns.barplot('TravelAlone', 'Survived', data=final\_train, color="mediumturquoise")  
plt.show()



* Hiển thị đồ thị dạng cột với trục tung là "Survived", trục hoành là "TravelAlone"

sns.barplot('Sex', 'Survived', data=train\_df, color="aquamarine")  
plt.show()



* Hiển thị đồ thị dạng cột với trục tung là "Survived", trục hoành là "Sex"

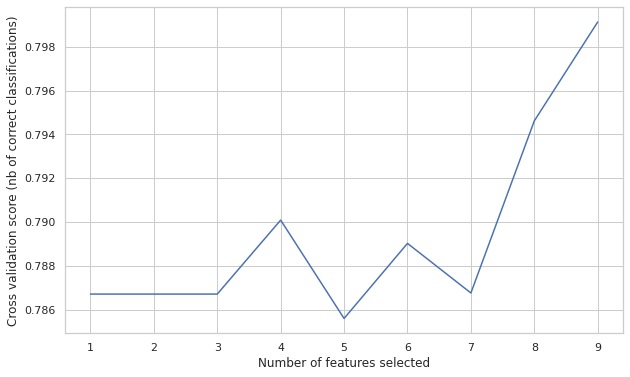
from sklearn.linear\_model import LogisticRegression  
from sklearn.feature\_selection import RFE  
  
cols = ["Age","Fare","TravelAlone","Pclass\_1","Pclass\_2","Embarked\_C","Embarked\_S","Sex\_male","IsMinor"]   
X = final\_train[cols]  
y = final\_train['Survived']  
# Build a logreg and compute the feature importances  
model = LogisticRegression()  
# create the RFE model and select 8 attributes  
rfe = RFE(model, 8)  
rfe = rfe.fit(X, y)  
# summarize the selection of the attributes  
print('Selected features: %s' % list(X.columns[rfe.support\_]))

Selected features: ['Age', 'TravelAlone', 'Pclass\_1', 'Pclass\_2', 'Embarked\_C', 'Embarked\_S', 'Sex\_male', 'IsMinor']

* Xây dựng nhật ký và tính toán các lần nhập tính năng
* Tạo mô hình RFE và chọn 8 thuộc tính
* Tóm tắt lựa chọn các thuộc tính

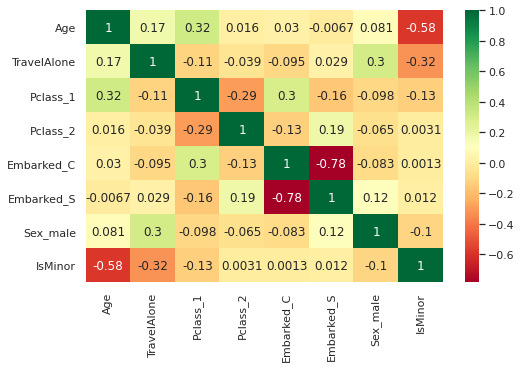
from sklearn.feature\_selection import RFECV  
rfecv = RFECV(estimator=LogisticRegression(), step=1, cv=10, scoring='accuracy')  
rfecv.fit(X, y)  
  
print("Optimal number of features: %d" % rfecv.n\_features\_)  
print('Selected features: %s' % list(X.columns[rfecv.support\_]))  
  
plt.figure(figsize=(10,6))  
plt.xlabel("Number of features selected")  
plt.ylabel("Cross validation score (nb of correct classifications)")  
plt.plot(range(1, len(rfecv.grid\_scores\_) + 1), rfecv.grid\_scores\_)  
plt.show()

Optimal number of features: 9  
Selected features: ['Age', 'Fare', 'TravelAlone', 'Pclass\_1', 'Pclass\_2', 'Embarked\_C', 'Embarked\_S', 'Sex\_male', 'IsMinor']



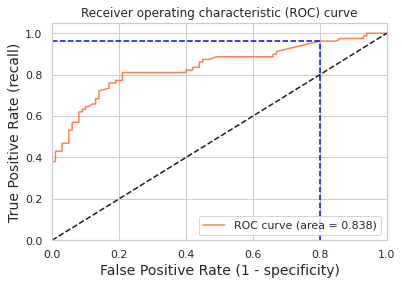
* Tạo RFE object
* Điểm "accuracy" tỷ lệ thuận với số lượng phân loại đúng
* Hiển thị đồ thị

Selected\_features = ['Age', 'TravelAlone', 'Pclass\_1', 'Pclass\_2', 'Embarked\_C',   
 'Embarked\_S', 'Sex\_male', 'IsMinor']  
X = final\_train[Selected\_features]  
  
plt.subplots(figsize=(8, 5))  
sns.heatmap(X.corr(), annot=True, cmap="RdYlGn")  
plt.show()



from sklearn.model\_selection import train\_test\_split, cross\_val\_score  
from sklearn.metrics import accuracy\_score, classification\_report, precision\_score, recall\_score   
from sklearn.metrics import confusion\_matrix, precision\_recall\_curve, roc\_curve, auc, log\_loss  
  
# create X (features) and y (response)  
X = final\_train[Selected\_features]  
y = final\_train['Survived']  
  
# use train/test split with different random\_state values  
# we can change the random\_state values that changes the accuracy scores  
# the scores change a lot, this is why testing scores is a high-variance estimate  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2)  
  
# check classification scores of logistic regression  
logreg = LogisticRegression()  
logreg.fit(X\_train, y\_train)  
y\_pred = logreg.predict(X\_test)  
y\_pred\_proba = logreg.predict\_proba(X\_test)[:, 1]  
[fpr, tpr, thr] = roc\_curve(y\_test, y\_pred\_proba)  
print('Train/Test split results:')  
print(logreg.\_\_class\_\_.\_\_name\_\_+" accuracy is %2.3f" % accuracy\_score(y\_test, y\_pred))  
print(logreg.\_\_class\_\_.\_\_name\_\_+" log\_loss is %2.3f" % log\_loss(y\_test, y\_pred\_proba))  
print(logreg.\_\_class\_\_.\_\_name\_\_+" auc is %2.3f" % auc(fpr, tpr))  
  
idx = np.min(np.where(tpr > 0.95)) # index of the first threshold for which the sensibility > 0.95  
  
plt.figure()  
plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, tpr))  
plt.plot([0, 1], [0, 1], 'k--')  
plt.plot([0,fpr[idx]], [tpr[idx],tpr[idx]], 'k--', color='blue')  
plt.plot([fpr[idx],fpr[idx]], [0,tpr[idx]], 'k--', color='blue')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('False Positive Rate (1 - specificity)', fontsize=14)  
plt.ylabel('True Positive Rate (recall)', fontsize=14)  
plt.title('Receiver operating characteristic (ROC) curve')  
plt.legend(loc="lower right")  
plt.show()  
  
print("Using a threshold of %.3f " % thr[idx] + "guarantees a sensitivity of %.3f " % tpr[idx] +   
 "and a specificity of %.3f" % (1-fpr[idx]) +   
 ", i.e. a false positive rate of %.2f%%." % (np.array(fpr[idx])\*100))

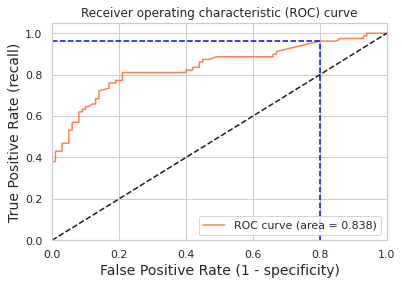
Train/Test split results:  
LogisticRegression accuracy is 0.782  
LogisticRegression log\_loss is 0.504  
LogisticRegression auc is 0.838



Using a threshold of 0.070 guarantees a sensitivity of 0.962 and a specificity of 0.200, i.e. a false positive rate of 80.00%.

from sklearn.metrics import accuracy\_score, classification\_report, precision\_score, recall\_score   
from sklearn.metrics import confusion\_matrix, precision\_recall\_curve, roc\_curve, auc, log\_loss  
  
# create X (features) and y (response)  
X = final\_train[Selected\_features]  
y = final\_train['Survived']  
  
# use train/test split with different random\_state values  
# we can change the random\_state values that changes the accuracy scores  
# the scores change a lot, this is why testing scores is a high-variance estimate  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=2)  
  
# check classification scores of logistic regression  
logreg = LogisticRegression()  
logreg.fit(X\_train, y\_train)  
y\_pred = logreg.predict(X\_test)  
y\_pred\_proba = logreg.predict\_proba(X\_test)[:, 1]  
[fpr, tpr, thr] = roc\_curve(y\_test, y\_pred\_proba)  
print('Train/Test split results:')  
print(logreg.\_\_class\_\_.\_\_name\_\_+" accuracy is %2.3f" % accuracy\_score(y\_test, y\_pred))  
print(logreg.\_\_class\_\_.\_\_name\_\_+" log\_loss is %2.3f" % log\_loss(y\_test, y\_pred\_proba))  
print(logreg.\_\_class\_\_.\_\_name\_\_+" auc is %2.3f" % auc(fpr, tpr))  
  
idx = np.min(np.where(tpr > 0.95)) # index of the first threshold for which the sensibility > 0.95  
  
plt.figure()  
plt.plot(fpr, tpr, color='coral', label='ROC curve (area = %0.3f)' % auc(fpr, tpr))  
plt.plot([0, 1], [0, 1], 'k--')  
plt.plot([0,fpr[idx]], [tpr[idx],tpr[idx]], 'k--', color='blue')  
plt.plot([fpr[idx],fpr[idx]], [0,tpr[idx]], 'k--', color='blue')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('False Positive Rate (1 - specificity)', fontsize=14)  
plt.ylabel('True Positive Rate (recall)', fontsize=14)  
plt.title('Receiver operating characteristic (ROC) curve')  
plt.legend(loc="lower right")  
plt.show()  
  
print("Using a threshold of %.3f " % thr[idx] + "guarantees a sensitivity of %.3f " % tpr[idx] +   
 "and a specificity of %.3f" % (1-fpr[idx]) +   
 ", i.e. a false positive rate of %.2f%%." % (np.array(fpr[idx])\*100))

Train/Test split results:  
LogisticRegression accuracy is 0.782  
LogisticRegression log\_loss is 0.504  
LogisticRegression auc is 0.838



Using a threshold of 0.070 guarantees a sensitivity of 0.962 and a specificity of 0.200, i.e. a false positive rate of 80.00%.

from sklearn.model\_selection import cross\_validate  
  
scoring = {'accuracy': 'accuracy', 'log\_loss': 'neg\_log\_loss', 'auc': 'roc\_auc'}  
  
modelCV = LogisticRegression()  
  
results = cross\_validate(modelCV, X, y, cv=10, scoring=list(scoring.values()),   
 return\_train\_score=False)  
  
print('K-fold cross-validation results:')  
for sc in range(len(scoring)):  
 print(modelCV.\_\_class\_\_.\_\_name\_\_+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['test\_%s' % list(scoring.values())[sc]].mean()  
 if list(scoring.values())[sc]=='neg\_log\_loss'   
 else results['test\_%s' % list(scoring.values())[sc]].mean(),   
 results['test\_%s' % list(scoring.values())[sc]].std()))

K-fold cross-validation results:  
LogisticRegression average accuracy: 0.795 (+/-0.025)  
LogisticRegression average log\_loss: 0.454 (+/-0.037)  
LogisticRegression average auc: 0.850 (+/-0.028)

* Tính trung bình accuracy, log\_loss, auc

cols = ["Age","Fare","TravelAlone","Pclass\_1","Pclass\_2","Embarked\_C","Embarked\_S","Sex\_male","IsMinor"]  
X = final\_train[cols]  
  
scoring = {'accuracy': 'accuracy', 'log\_loss': 'neg\_log\_loss', 'auc': 'roc\_auc'}  
  
modelCV = LogisticRegression()  
  
results = cross\_validate(modelCV, final\_train[cols], y, cv=10, scoring=list(scoring.values()),   
 return\_train\_score=False)  
  
print('K-fold cross-validation results:')  
for sc in range(len(scoring)):  
 print(modelCV.\_\_class\_\_.\_\_name\_\_+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['test\_%s' % list(scoring.values())[sc]].mean()  
 if list(scoring.values())[sc]=='neg\_log\_loss'   
 else results['test\_%s' % list(scoring.values())[sc]].mean(),   
 results['test\_%s' % list(scoring.values())[sc]].std()))

K-fold cross-validation results:  
LogisticRegression average accuracy: 0.799 (+/-0.028)  
LogisticRegression average log\_loss: 0.455 (+/-0.037)  
LogisticRegression average auc: 0.849 (+/-0.028)

cols = ["Age","Fare","TravelAlone","Pclass\_1","Pclass\_2","Embarked\_C","Embarked\_S","Sex\_male","IsMinor"]  
X = final\_train[cols]  
  
scoring = {'accuracy': 'accuracy', 'log\_loss': 'neg\_log\_loss', 'auc': 'roc\_auc'}  
  
modelCV = LogisticRegression()  
  
results = cross\_validate(modelCV, final\_train[cols], y, cv=10, scoring=list(scoring.values()),   
 return\_train\_score=False)  
  
print('K-fold cross-validation results:')  
for sc in range(len(scoring)):  
 print(modelCV.\_\_class\_\_.\_\_name\_\_+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['test\_%s' % list(scoring.values())[sc]].mean()  
 if list(scoring.values())[sc]=='neg\_log\_loss'   
 else results['test\_%s' % list(scoring.values())[sc]].mean(),   
 results['test\_%s' % list(scoring.values())[sc]].std()))

K-fold cross-validation results:  
LogisticRegression average accuracy: 0.799 (+/-0.028)  
LogisticRegression average log\_loss: 0.455 (+/-0.037)  
LogisticRegression average auc: 0.849 (+/-0.028)

cols = ["Age","Fare","TravelAlone","Pclass\_1","Pclass\_2","Embarked\_C","Embarked\_S","Sex\_male","IsMinor"]  
X = final\_train[cols]  
  
scoring = {'accuracy': 'accuracy', 'log\_loss': 'neg\_log\_loss', 'auc': 'roc\_auc'}  
  
modelCV = LogisticRegression()  
  
results = cross\_validate(modelCV, final\_train[cols], y, cv=10, scoring=list(scoring.values()),   
 return\_train\_score=False)  
  
print('K-fold cross-validation results:')  
for sc in range(len(scoring)):  
 print(modelCV.\_\_class\_\_.\_\_name\_\_+" average %s: %.3f (+/-%.3f)" % (list(scoring.keys())[sc], -results['test\_%s' % list(scoring.values())[sc]].mean()  
 if list(scoring.values())[sc]=='neg\_log\_loss'   
 else results['test\_%s' % list(scoring.values())[sc]].mean(),   
 results['test\_%s' % list(scoring.values())[sc]].std()))

K-fold cross-validation results:  
LogisticRegression average accuracy: 0.799 (+/-0.028)  
LogisticRegression average log\_loss: 0.455 (+/-0.037)  
LogisticRegression average auc: 0.849 (+/-0.028)

final\_test['Survived'] = log\_clf.predict(final\_test[Selected\_features])  
final\_test['PassengerId'] = test\_df['PassengerId']  
  
submission = final\_test[['PassengerId','Survived']]  
  
submission.to\_csv("submission.csv", index=False)  
  
submission.tail()

---------------------------------------------------------------------------  
NameError Traceback (most recent call last)  
<ipython-input-112-82758ca4cd1d> in <module>()  
----> 1 final\_test['Survived'] = log\_clf.predict(final\_test[Selected\_features])  
 2 final\_test['PassengerId'] = test\_df['PassengerId']  
 3   
 4 submission = final\_test[['PassengerId','Survived']]  
 5   
  
NameError: name 'log\_clf' is not defined