Raport projekt 1

Maciej Paczóski Bartosz Rożek

April 21, 2020

1 Wstęp

W ramach projektu zajęliśmy się modelowaniem zbioru *bank_marketing* zawierającego dane o klientach banku. Naszym celem było znalezienie najlepszego klasyfikatora zmiennej celu y, mówiącej czy dany klient będzie zainteresowany ofertą.

[1]:
```
from datetime import datetime
import math
import time
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import category_encoders as ce
import matplotlib.pyplot as plt
import seaborn as sns
from pandas_profiling import ProfileReport
```

[48]:
```
import warnings
warnings.filterwarnings('ignore')
```

[38]:
```
dataframe = pd.read_csv('bank_marketing_weka_dataset.csv')
continous_cols = ['balance', 'duration', 'pdays', 'previous']
```

Tak wyglądają dane w tabeli.

[39]:
```
dataframe.head()
```

[39]:
```
   age          job  marital  education default  balance housing loan  \
0   30   unemployed  married    primary      no     1787      no   no
1   33     services  married  secondary      no     4789     yes  yes
2   35   management   single   tertiary      no     1350     yes   no
3   30   management  married   tertiary      no     1476     yes  yes
4   59  blue-collar  married  secondary      no        0     yes   no

    contact  day month  duration  campaign  pdays  previous poutcome   y
0  cellular   19   oct      79.0         1   -1.0         0  unknown  no
1  cellular   11   may     220.0         1  339.0         4  failure  no
```

```
2  cellular   16   apr     185.0          1  330.0          1  failure  no
3   unknown    3   jun     199.0          4   -1.0          0  unknown  no
4   unknown    5   may     226.0          1   -1.0          0  unknown  no
```

Opis kolumn.

age -integer- **Age of client**: * numerical value

job -string- **Type of job**: * admin. * blue-collar * entrepreneur * housemaid * management * retired * self-employed * services * student * technician * unemployed * unknown

marital -string- **Marital status**: * divorced * married * single * unknown

education -string- **Level of education**: * primary * secondary * tertiary * unknown

default -string- **Has credit in default**: * no * yes * unknown

balance -integer- **Average yearly balance in Euro**: * numerical value

housing -string- **Has housing loan**: * no * yes * unknown

loan -string- **Has personal loan**: * no * yes * unknown

contact -string- **Communication type**: * unknown * telephone * cellular

day -integer- **Day of the month**: * numerical value between 1 and 31

month -string- **Month of the year**: * jan * feb * mar * apr * may * jun * jul * aug * sep * oct * nov * dec

duration -float- **Last contact duration**: * numerical value in seconds

campaign -integer- **Number of contacts made**: * numerical value

pdays -float- **Number of days passed since client was last contacted from a previous campaign**: * numerical value * -1 indicates client was not previously contacted

previous -integer- **Number of contacts performed before this campaign and for this client**: * numerical value

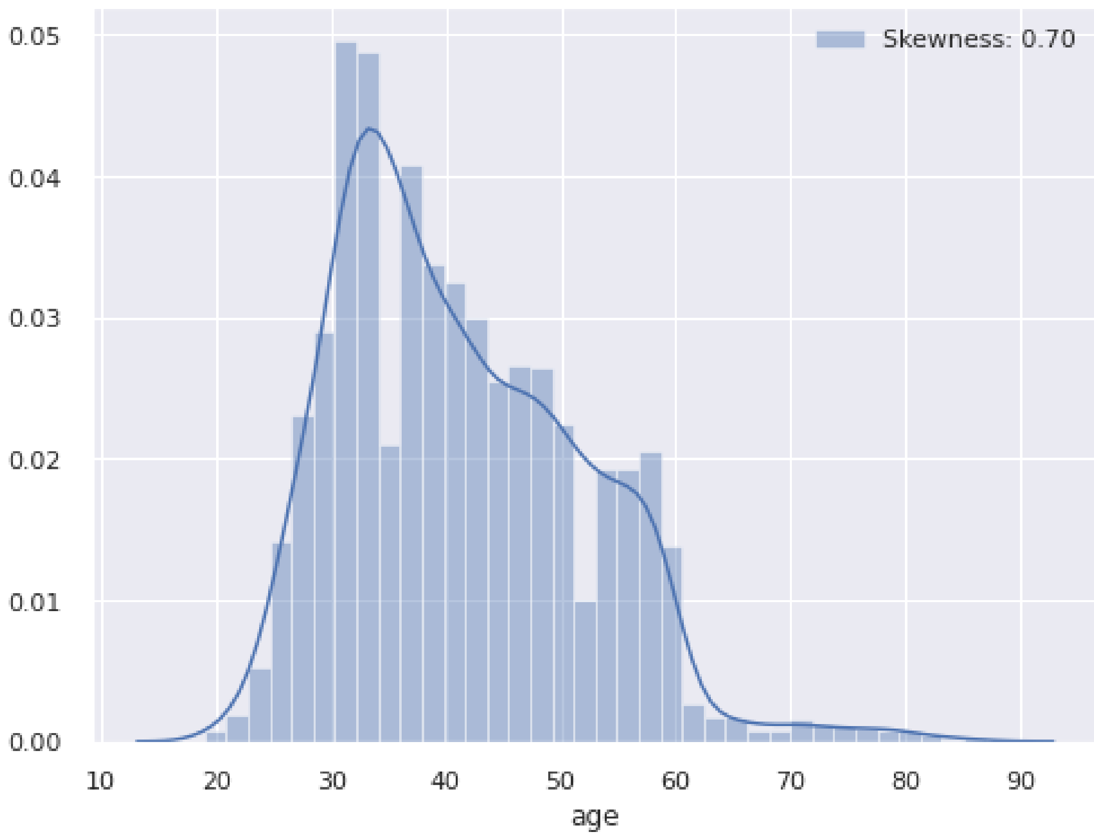
poutcome -string- **Outcome of previous marketing campaign**: * unknown * other * failure * success

y -string- **Predictor class**: * yes * no

2 Eksploracyjna analiza danych

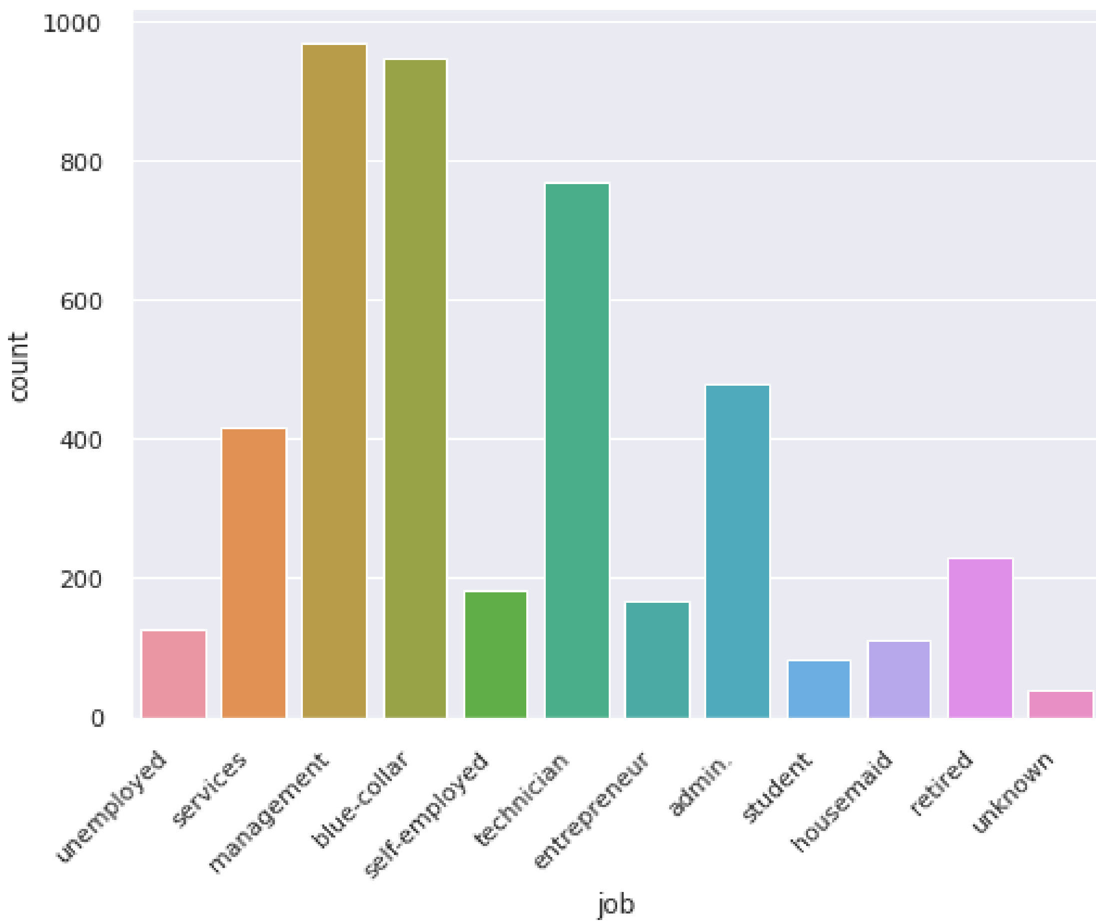
2.0.1 Rozkład wieku

[40]:
```
plt.figure(figsize=(8,6))
sns.distplot(dataframe.age, label='Skewness: %.2f' % (dataframe.age.skew()))
plt.legend()
plt.show()
```



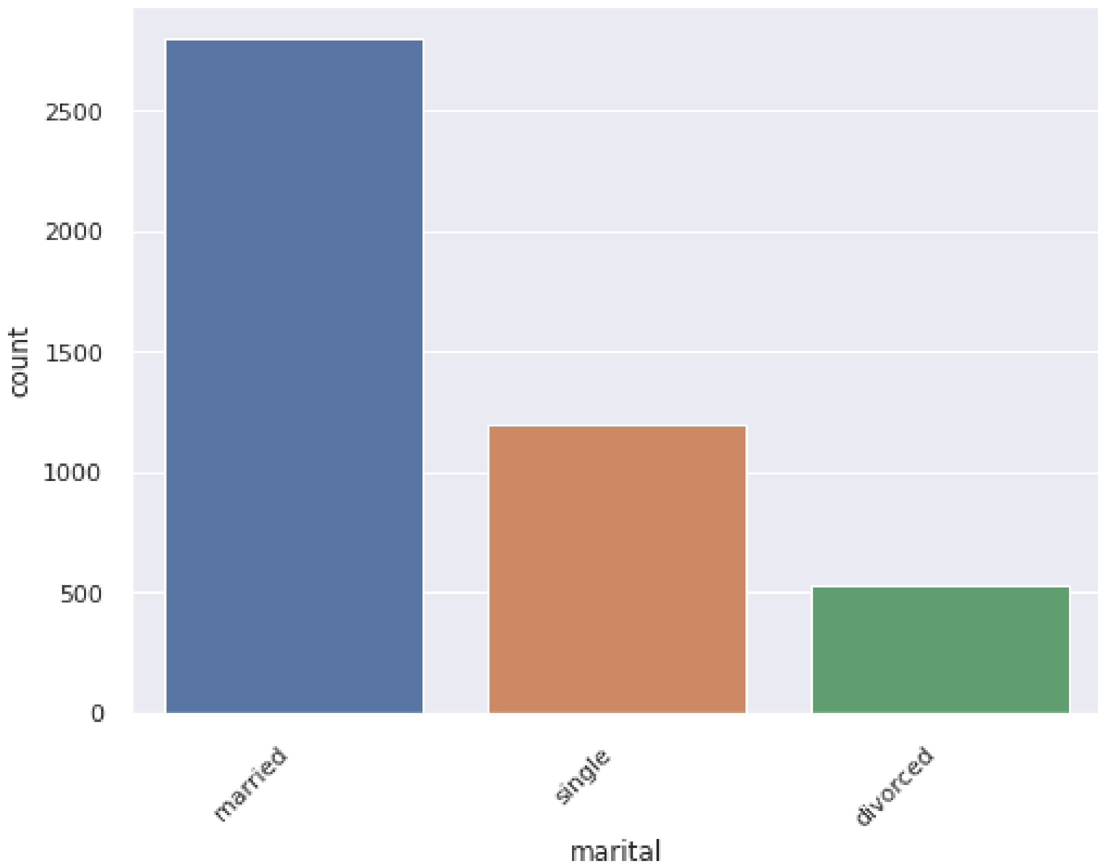
2.0.2 Zawody wykonywane

[41]:
```
sns.set(style="darkgrid")

fig = plt.figure(figsize=(8,6))
ax = sns.countplot(x='job', data=dataframe)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
plt.show()
```



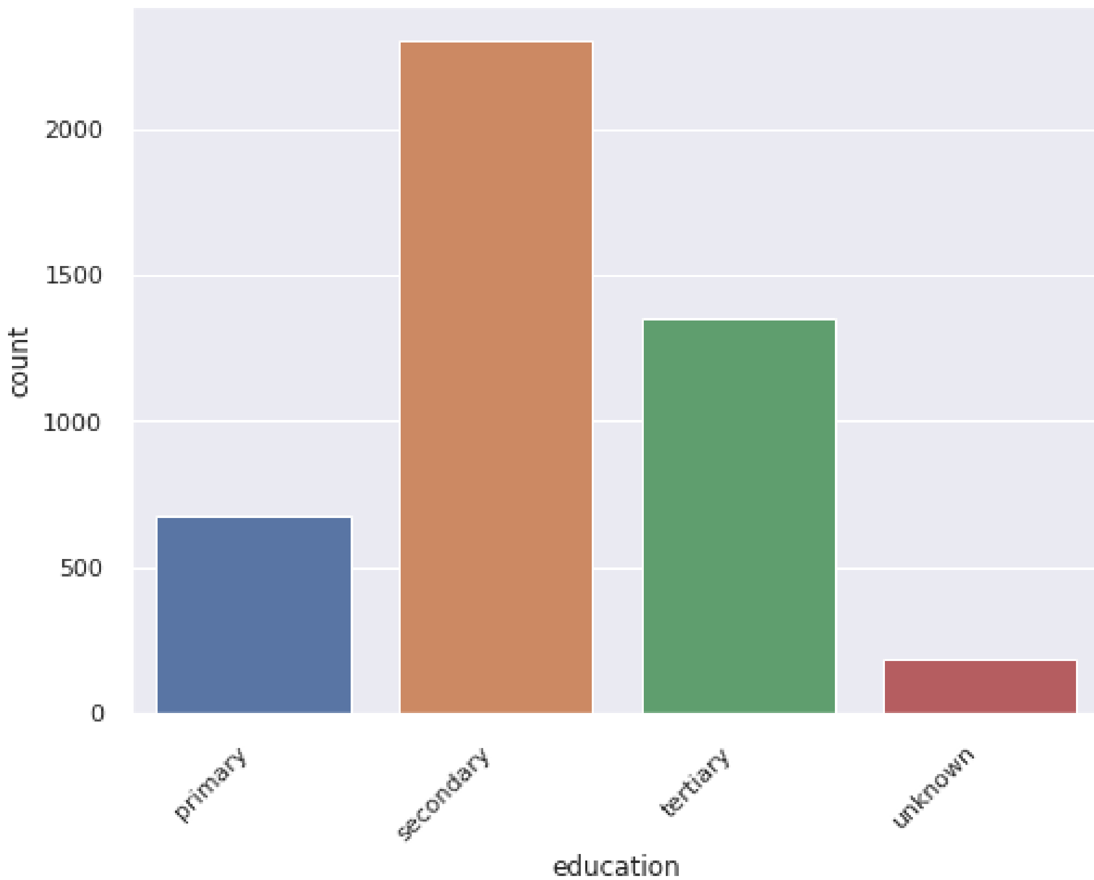
2.0.3 Rozkład stanu cywilnego

[42]:
```
sns.set(style="darkgrid")

fig = plt.figure(figsize=(8,6))
ax = sns.countplot(x='marital', data=dataframe)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
plt.show()
```



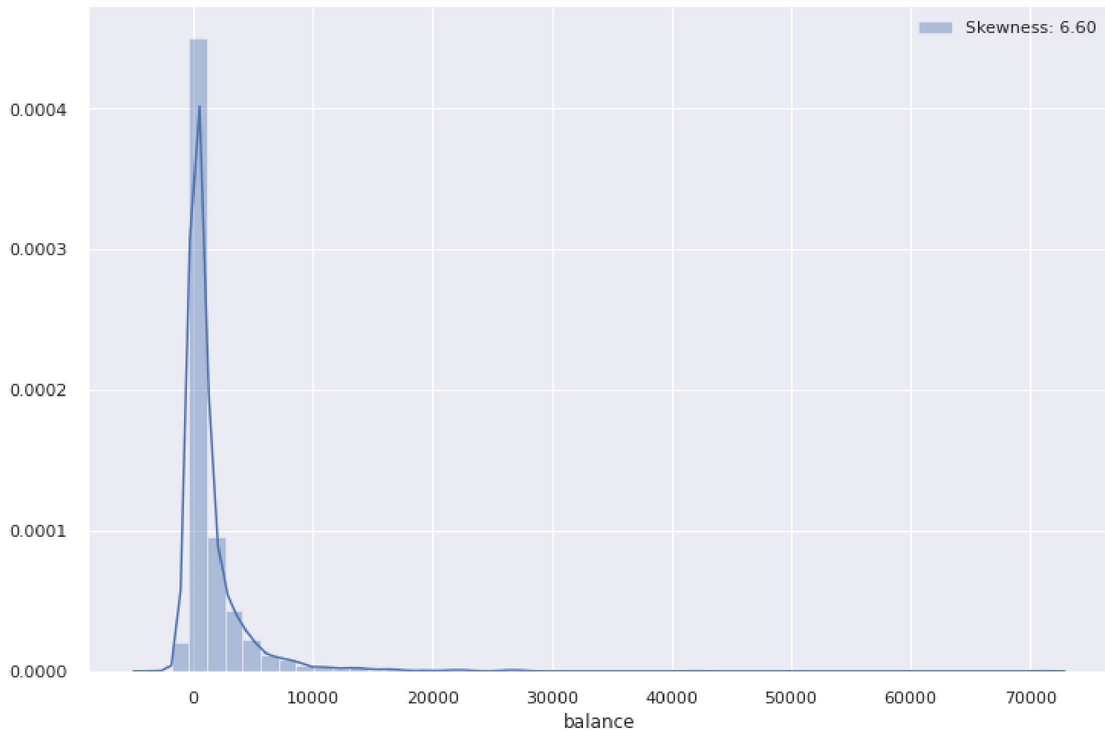
2.0.4 Rozkład wykształcenia

[29]:
```
sns.set(style="darkgrid")

fig = plt.figure(figsize=(8,6))
ax = sns.countplot(x='education', data=dataframe)
ax.set_xticklabels(ax.get_xticklabels(), rotation=45,␣
 ↪horizontalalignment='right')
plt.show()
```



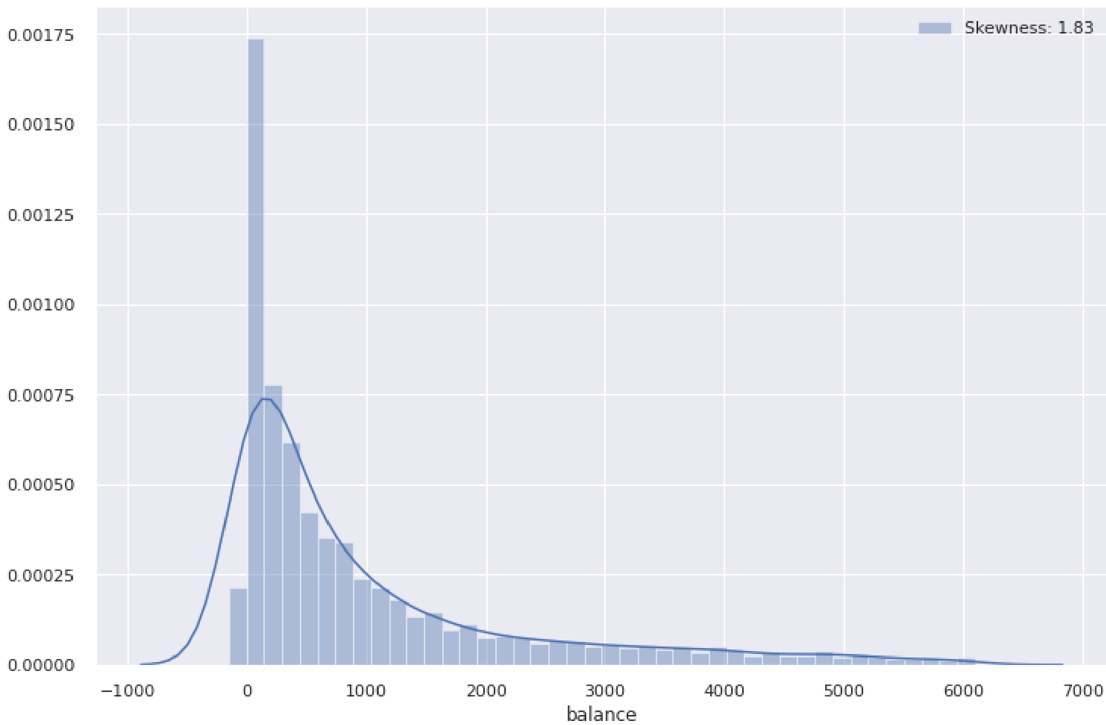
2.0.5 Rozkład bilansu konta

[13]:
```
plt.figure(figsize=(12,8))
sns.distplot(dataframe.balance, label='Skewness: %.2f' % (dataframe.balance.
 ↪skew()))
plt.legend()
plt.show()
```



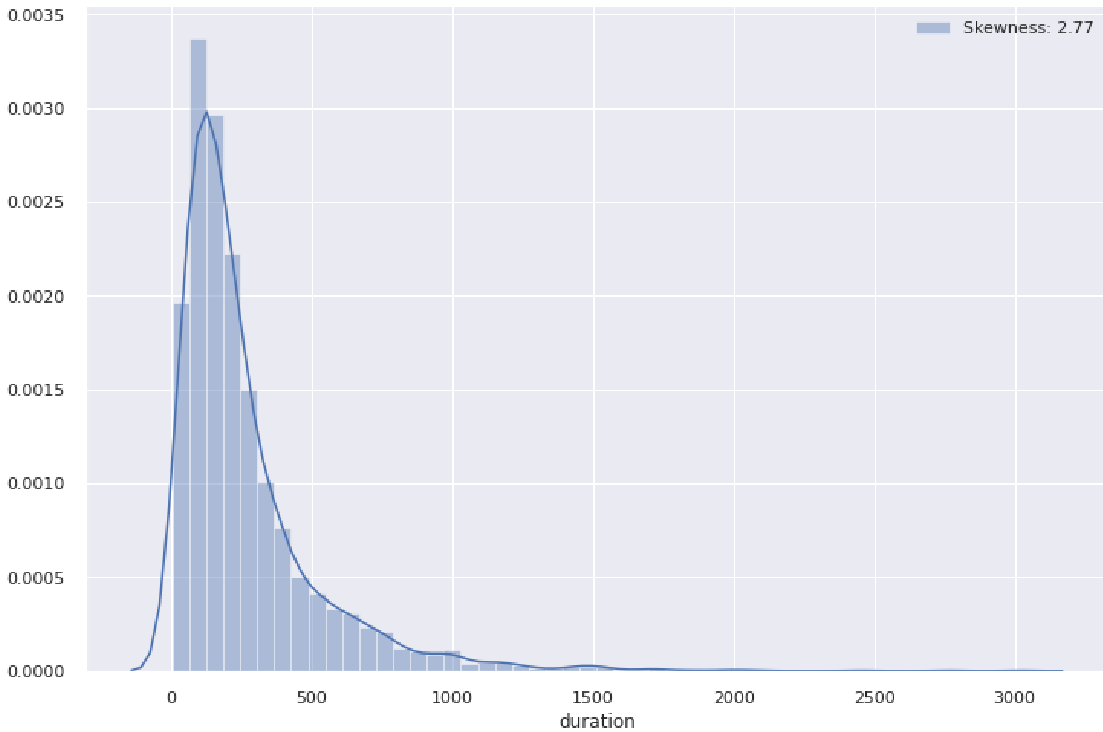
2.0.6 Rozkład po usunięciu outlierów

[14]:
```
plt.figure(figsize=(12,8))
sns.distplot(dataframe[(dataframe.balance.quantile(0.05) < dataframe.balance) &␣
 ↪(dataframe.balance < dataframe.balance.quantile(0.95))].balance,␣
 ↪label='Skewness: %.2f' % (dataframe[(dataframe.balance.quantile(0.05) <␣
 ↪dataframe.balance) & (dataframe.balance < dataframe.balance.quantile(0.95))].
 ↪balance.skew()))
plt.legend()
plt.show()
```



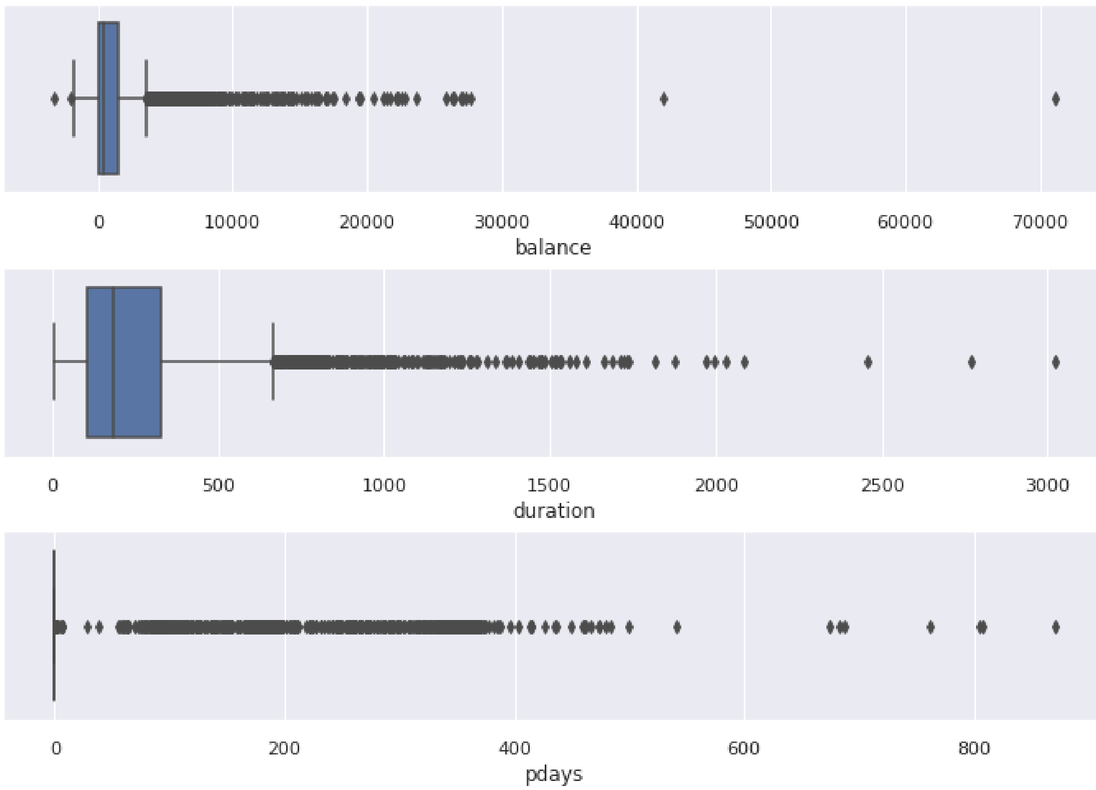
2.0.7 Rozkład czasu trwania

[15]:
```
plt.figure(figsize=(12,8))
sns.distplot(dataframe.duration, label='Skewness: %.2f' % (dataframe.duration.
 ↪skew()))
plt.legend()
plt.show()
```



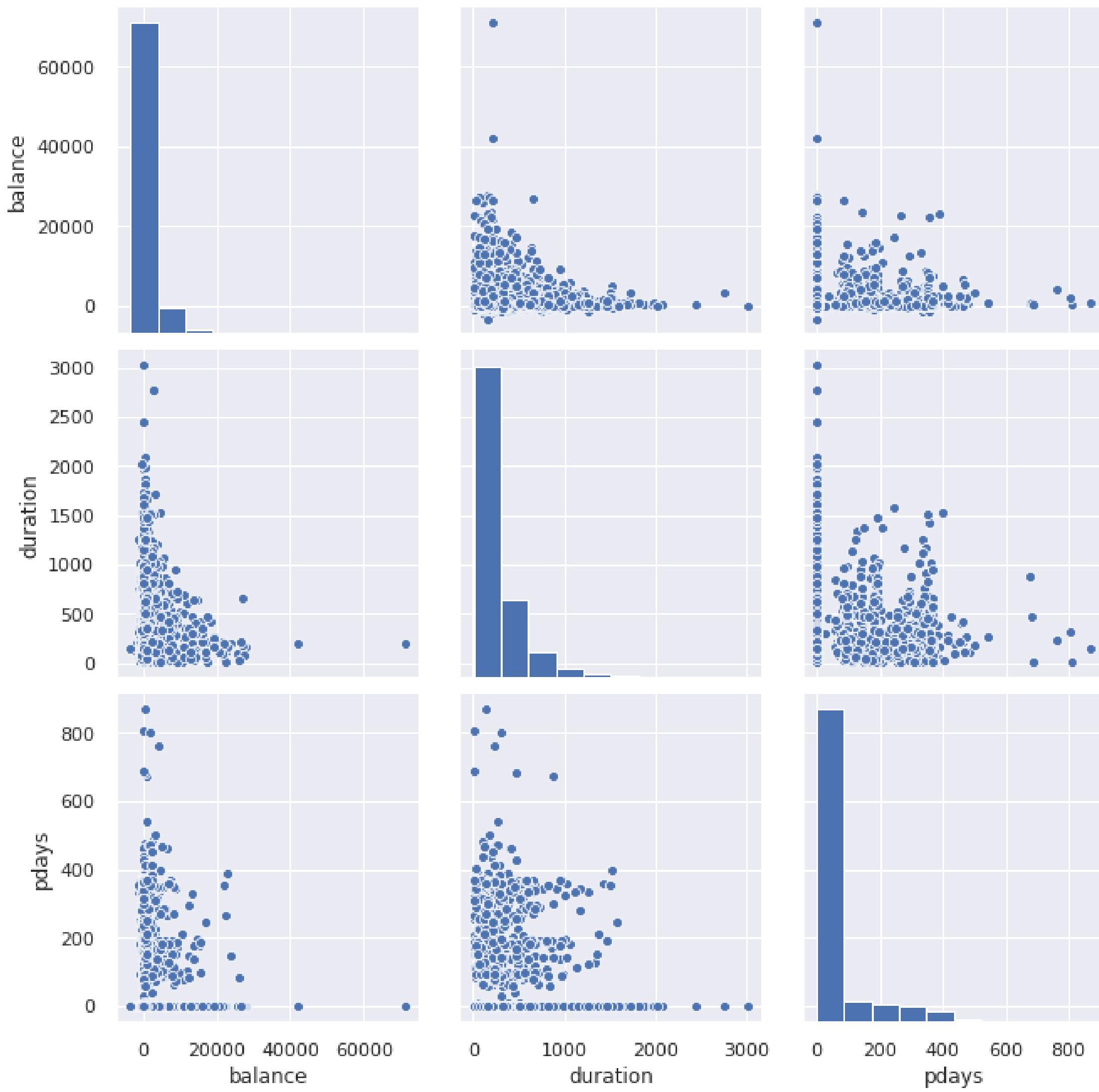
2.0.8 Boxploty kolumn ciągłych

[22]:
```
sns.set(style="darkgrid")

fig = plt.figure(figsize=(12,8))
fig.subplots_adjust(hspace=0.4, wspace=0.4)
for i in range(1, 4):
    ax = fig.add_subplot(3, 1, i),
    ax = sns.boxplot(x=dataframe[continous_cols[i-1]])
```



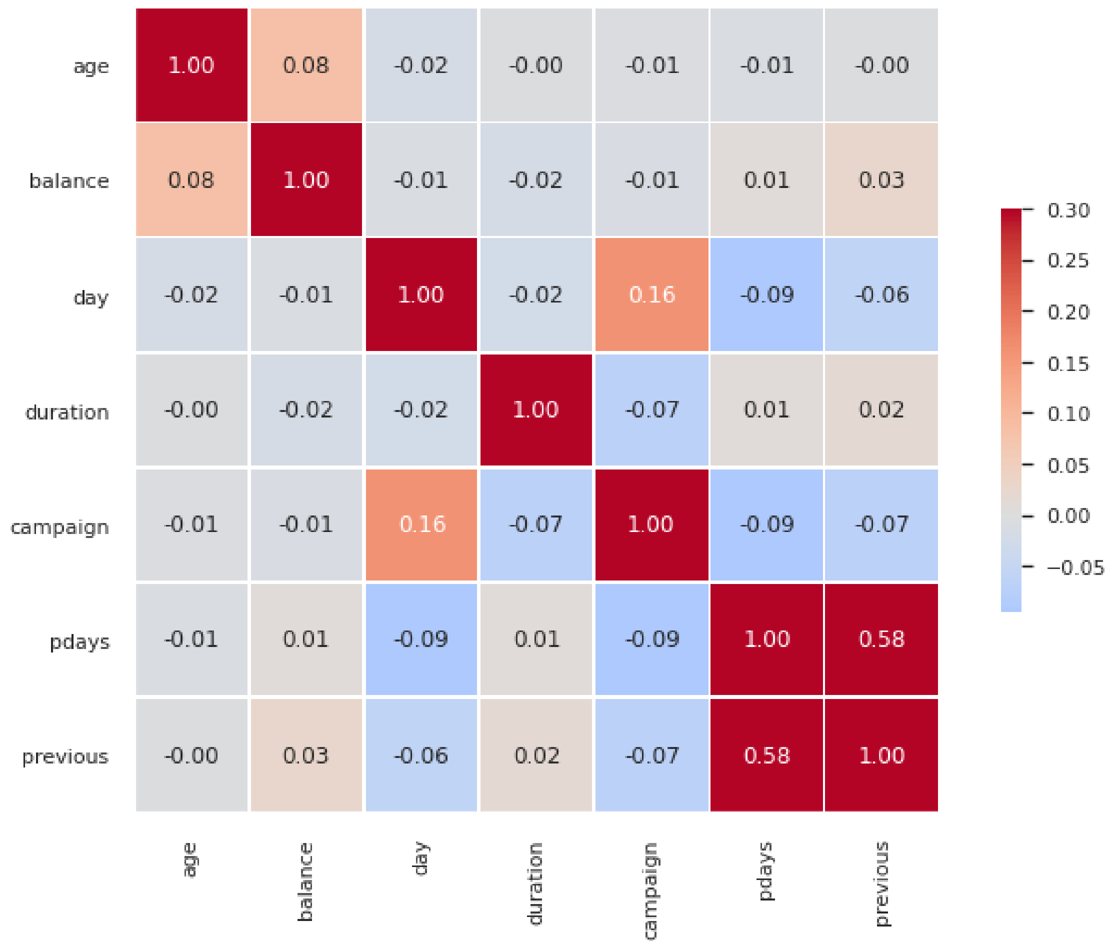
2.0.9 Pairplot kolumn ciągłych

[36]:
```
sns.pairplot(dataframe[continous_cols], height=3)
plt.show()
```



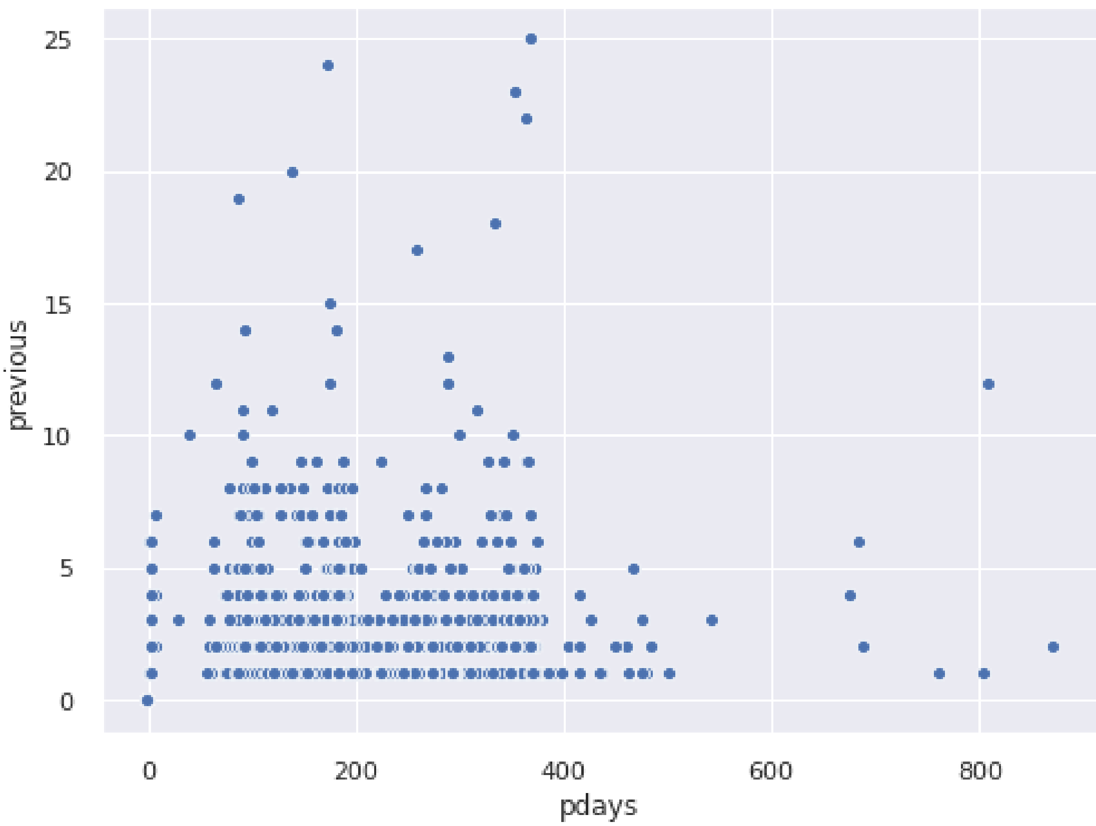
2.0.10 Mapa ciepła korelacji

[17]:
```
corr = dataframe.corr()
g = sns.heatmap(corr,  vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True,␣
 ↪fmt='.2f', cmap='coolwarm')
sns.despine()
g.figure.set_size_inches(12,8)

plt.show()
```



2.0.11 Scatterplot między dwoma najbardziej skorelowanymi kolumnami

[22]:
```
fig = plt.figure(figsize=(8,6))
ax = sns.scatterplot(data=dataframe, x = 'pdays', y = 'previous')
```



3 Feature engineering

3.1 Sprawdzenie braków danych

[5]:
```
dataframe.isna().any()
```

[5]:
```
age          False
job          False
marital      False
education    False
default      False
balance      False
housing      False
loan         False
contact      False
day          False
month        False
duration     False
campaign     False
pdays        False
```

```
previous     False
poutcome     False
y            False
dtype: bool
```

[6]:
```python
dataframe.isin(['unknown']).any()
```

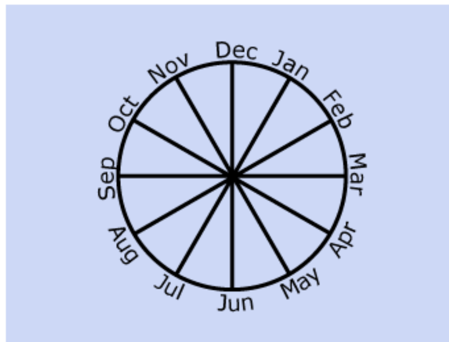
[6]:
```
age           False
job            True
marital       False
education      True
default       False
balance       False
housing       False
loan          False
contact        True
day           False
month         False
duration      False
campaign      False
pdays         False
previous      False
poutcome       True
y             False
dtype: bool
```

Dane pozbawione są klasycznich braków NA. Zastąpione są wartościami unknown, pojawiającymi się w kilu kolumnach.

3.2 Zmiana daty na reprezentacje kątem

Cyclical Temporal Modeling

One approach is through cyclical representations. This is to model the days of the year as their points on a yearly circle.



By using the X, Y locations on the above circle as inputs to our model, our model is allowed to learn times of year that have high sales and opposite times of the year with lower sales, so that it doesn't mistakenly attribute seasonal sales trends to promotions.

date	sin	cos
2017-01-01	0.017213	0.999852
2017-01-02	0.034422	0.999407
2017-01-03	0.051620	0.998667
2017-01-04	0.068802	0.997630
2017-01-05	0.085965	0.996298
2017-01-06	0.103102	0.994671
2017-01-07	0.120208	0.992749
2017-01-08	0.137279	0.990532

Cyclical Representation For Seasonality

[7]:
```
date = list(dataframe.day)
date = [str(item) for item in date]
date = [date[i] + '/' + list(dataframe.month)[i] for i in range(len(dataframe.
 ↪day))]

date = [int(datetime.strptime(item, '%d/%b').strftime('%j')) for item in date]
date_sin = [math.sin(date[i]/360) for i in range (len(date))]
```

```python
date_cos = [math.cos(date[i]/360) for i in range (len(date))]
dataframe['date_sin'] = date_sin
dataframe['date_cos'] = date_cos

to_drop = ['month', 'day']
dataframe = dataframe.drop(to_drop, axis=1)

dataframe.head()
```

[7]:

```
   age          job  marital  education default  balance housing loan  \
0   30   unemployed  married    primary      no     1787      no   no
1   33     services  married  secondary      no     4789     yes  yes
2   35   management   single   tertiary      no     1350     yes   no
3   30   management  married   tertiary      no     1476     yes  yes
4   59  blue-collar  married  secondary      no        0     yes   no

    contact  duration  campaign  pdays  previous poutcome   y  date_sin  \
0  cellular      79.0         1   -1.0         0  unknown  no  0.725053
1  cellular     220.0         1  339.0         4  failure  no  0.355911
2  cellular     185.0         1  330.0         1  failure  no  0.290208
3   unknown     199.0         4   -1.0         0  unknown  no  0.414850
4   unknown     226.0         1   -1.0         0  unknown  no  0.340287

   date_cos
0  0.688693
1  0.934520
2  0.956964
3  0.909890
4  0.940322
```

3.3 Kodowanie zmiennych kategorycznych

[8]:

```python
dataframe[["marital","education","default","housing","loan","contact","poutcome",
 ↪"y"]] =
 ↪dataframe[["marital","education","default","housing","loan","contact","poutcome",
 ↪"y"]].apply(LabelEncoder().fit_transform)
a = ce.BaseNEncoder(base=2).fit_transform(dataframe["job"])
dataframe = pd.concat([dataframe, a], axis=1).drop(["job"], axis=1)
dataframe.head()
```

[8]:

```
   age  marital  education  default  balance  housing  loan  contact  \
0   30        1          0        0     1787        0     0        0
1   33        1          1        0     4789        1     1        0
2   35        2          2        0     1350        1     0        0
3   30        1          2        0     1476        1     1        2
4   59        1          1        0        0        1     0        2
```

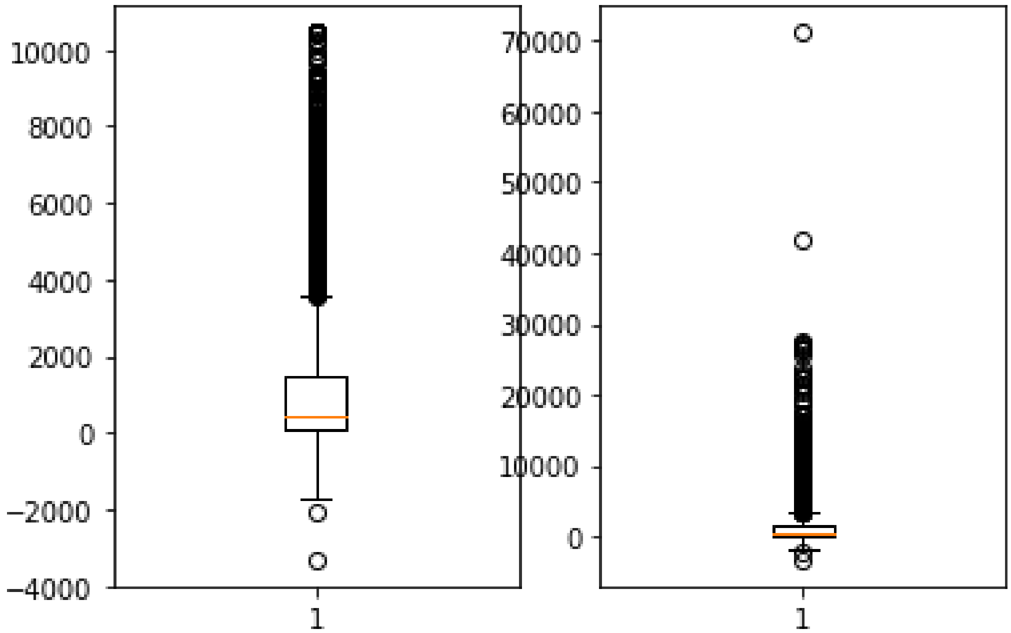
```
   duration  campaign  ...  previous  poutcome  y  date_sin  date_cos  job_0  \
0      79.0         1  ...         0         3  0  0.725053  0.688693      0
1     220.0         1  ...         4         0  0  0.355911  0.934520      0
2     185.0         1  ...         1         0  0  0.290208  0.956964      0
3     199.0         4  ...         0         3  0  0.414850  0.909890      0
4     226.0         1  ...         0         3  0  0.340287  0.940322      0

   job_1  job_2  job_3  job_4
0      0      0      0      1
1      0      0      1      0
2      0      0      1      1
3      0      0      1      1
4      0      1      0      0

[5 rows x 21 columns]
```

3.4 Outliers

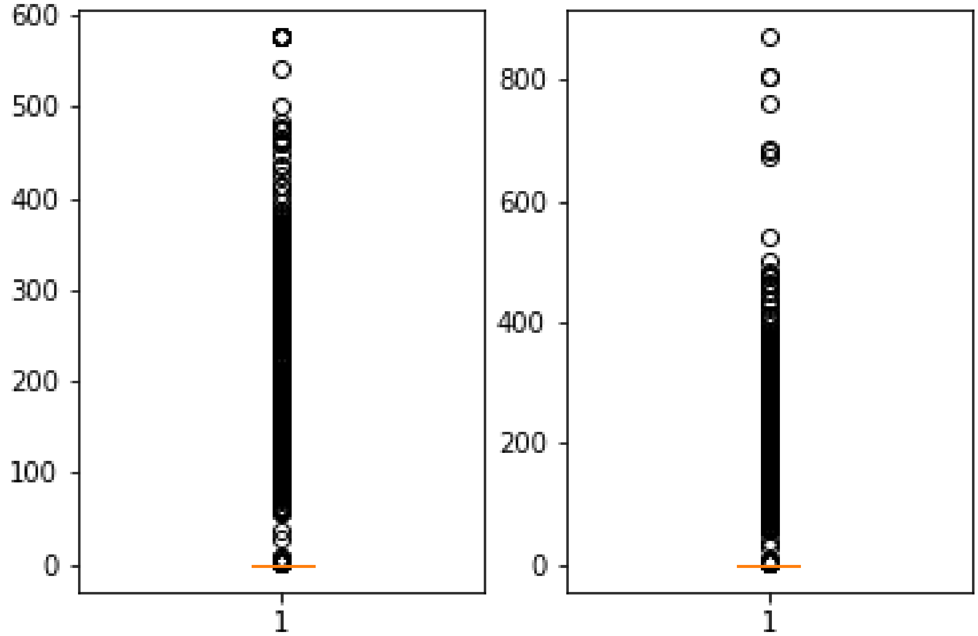
[9]:
```
dataframe_orgin = pd.read_csv('bank_marketing_weka_dataset.csv')
upper_lim_balance = dataframe_orgin['balance'].mean() +␣
↪dataframe_orgin['balance'].std() * 3
lower_lim_balance = dataframe_orgin['balance'].mean() -␣
↪dataframe_orgin['balance'].std() * 3

dataframe.loc[(dataframe['balance'] > upper_lim_balance),'balance'] =␣
↪upper_lim_balance
dataframe.loc[(dataframe['balance'] < lower_lim_balance),'balance'] =␣
↪lower_lim_balance

upper_lim_pdays = dataframe_orgin.loc[dataframe_orgin.pdays > -1, 'pdays'].
↪mean() + dataframe_orgin.loc[dataframe_orgin.pdays > -1, 'pdays'].std() * 3

dataframe.loc[(dataframe['pdays'] > upper_lim_pdays),'pdays'] = upper_lim_pdays

upper_lim_duration = dataframe_orgin['duration'].mean() +␣
↪dataframe_orgin['duration'].std() * 3

dataframe.loc[(dataframe['duration'] > upper_lim_pdays),'duration'] =␣
↪upper_lim_duration
```

[10]:
```
ig, axs = plt.subplots(1,2)

axs[0].boxplot(dataframe.balance)
axs[1].boxplot(dataframe_orgin.balance)
```

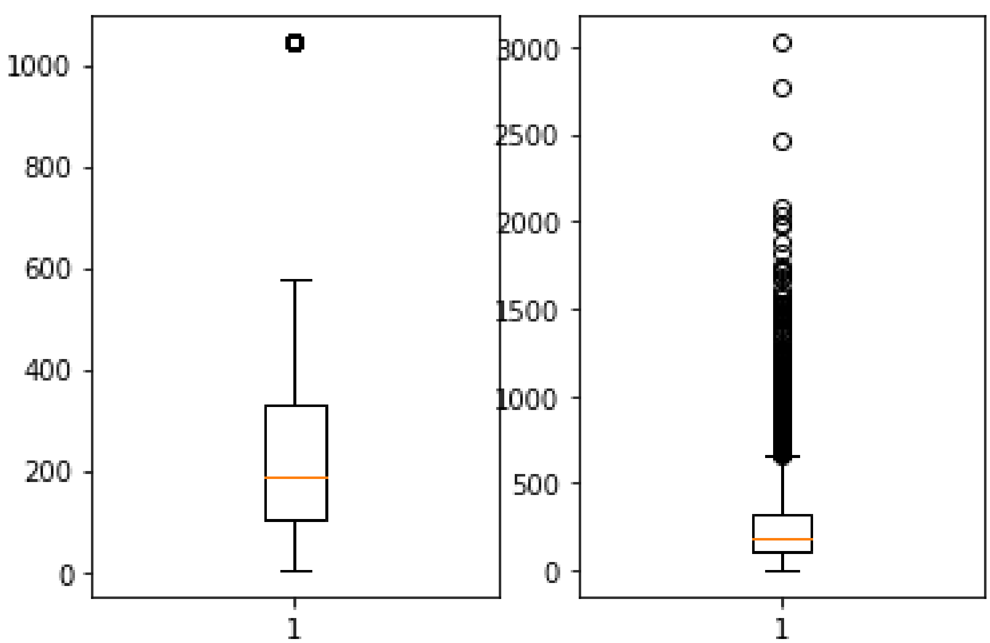
```
plt.show()
```



[11]:
```
fig, axs = plt.subplots(1,2)

axs[0].boxplot(dataframe.pdays)
axs[1].boxplot(dataframe_orgin.pdays)

plt.show()
```



[12]:
```
fig, axs = plt.subplots(1,2)

axs[0].boxplot(dataframe.duration)
axs[1].boxplot(dataframe_orgin.duration)

plt.show()
```



4 Model/models performance

4.0.1 Podział zbiorów

[275]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
                                         X, y, test_size=0.33, random_state=42,␣
 ↪stratify = y)
scores = pd.DataFrame({'score':[],'f1':[],'roc':[], 'recall':[]})
```

[276]:
```
from sklearn.metrics import f1_score
from sklearn.feature_selection import RFE
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn import datasets, metrics, model_selection, svm
from sklearn.metrics import recall_score

def importance(estimator, n):
    estimator = rfc
    selector = RFE(estimator, n_features_to_select=n, step=1)
    selector = selector.fit(X_train, y_train)
    selector.score(X_test, y_test)
    istot = [x for _,x in sorted(zip(selector.ranking_,X.columns))]
    return istot
```

4.0.2 DTC

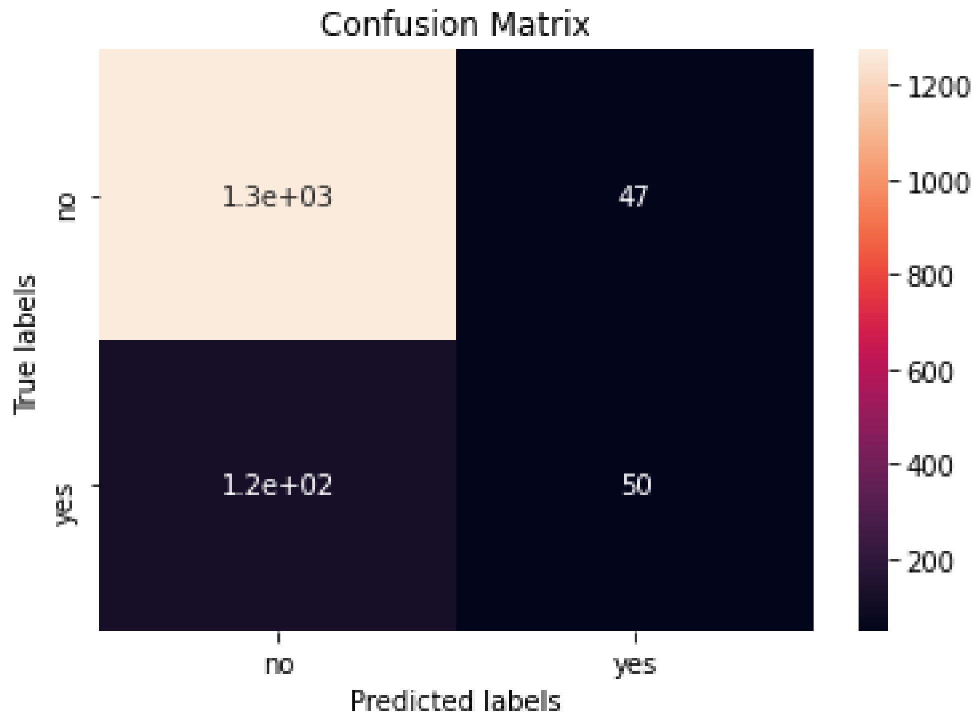
[277]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import RFE

tree_model = DecisionTreeClassifier()
tree_model
```

[277]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

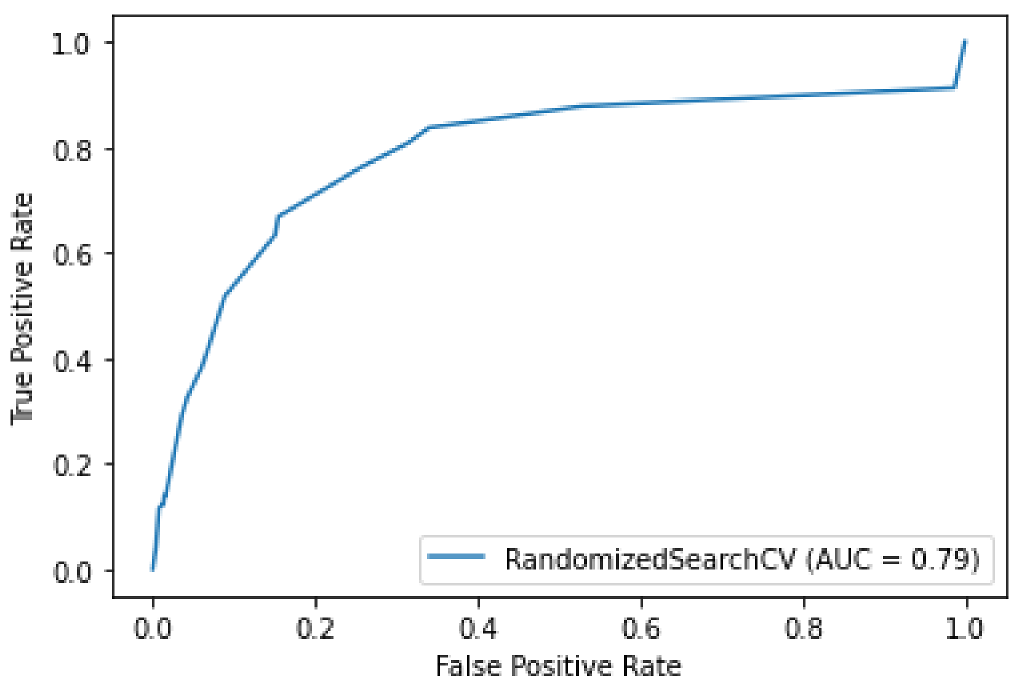
[368]:
```
max_depth=[3, 5, 6]
criterion=["gini","entropy"]
min_samples_split=[5,10,15]
param_grid = dict(max_depth=max_depth,criterion=criterion,␣
↪min_samples_split=min_samples_split)

random = RandomizedSearchCV(estimator=tree_model,␣
↪param_distributions=param_grid, cv = 3, n_jobs=-1, scoring='f1')

random_result = random.fit(X_train, y_train)
# Summarize results
print("Best: %f using %s" % (random_result.best_score_, random_result.
↪best_params_))

random.fit(X_train, y_train)

score = accuracy_score(y_test, random.predict(X_test))
f1 = f1_score(y_test, random.predict(X_test))
roc = roc_auc_score(y_test,random.predict(X_test))
recall = recall_score(y_test,random.predict(X_test))

print('Score:', score)
print('F1:', f1)
print('ROC:', roc)
print('Recall:', recall)

row_df = pd.DataFrame({'score':score, 'f1':f1, "roc":roc, "recall":recall},␣
↪index = ["DTC"])
scores = pd.concat([row_df, scores])
```

```
Best: 0.439195 using {'min_samples_split': 5, 'max_depth': 6, 'criterion':
'gini'}
Score: 0.8867292225201072
F1: 0.37174721189591076
ROC: 0.627545806906272
Recall: 0.29069767441860467
```

[369]:
```
cm = confusion_matrix(y_test, random.predict(X_test))

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['no', 'yes']); ax.yaxis.set_ticklabels(['no', 'yes']);
```

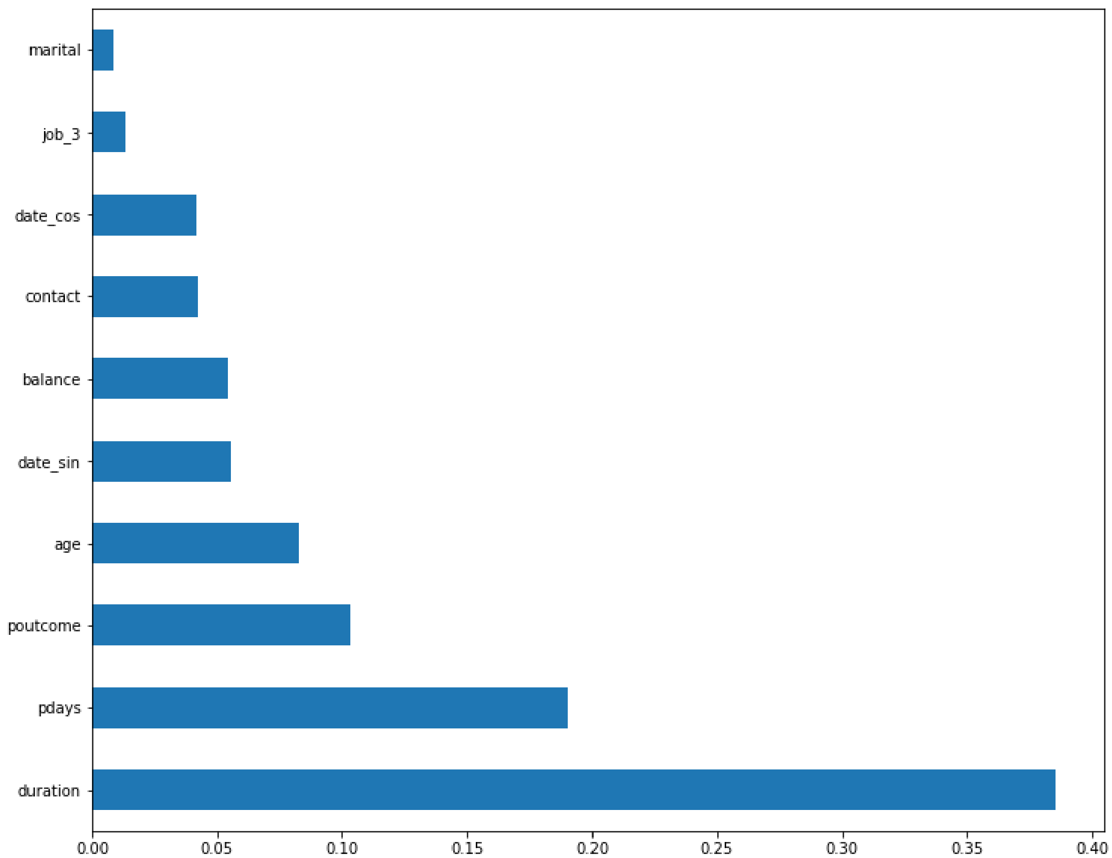


[370]:
```
metrics.plot_roc_curve(random, X_test, y_test)
plt.show()
```



[372]:
```
plt.figure(figsize=(12,10))
model = random.best_estimator_
print(model.feature_importances_) #use inbuilt class feature_importances of tree␣
 ↪based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[0.08258421 0.00830744 0.00232608 0.00620289 0.05436098 0.
 0.0019881  0.04244205 0.38527368 0.00541674 0.19039464 0.0065777
 0.10306954 0.05558017 0.04194222 0.         0.         0.
 0.01353357 0.        ]
```



4.0.3 Random Forest

[281]:
```
from sklearn.ensemble import RandomForestClassifier

rfc = RandomForestClassifier()
rfc
```

[281]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

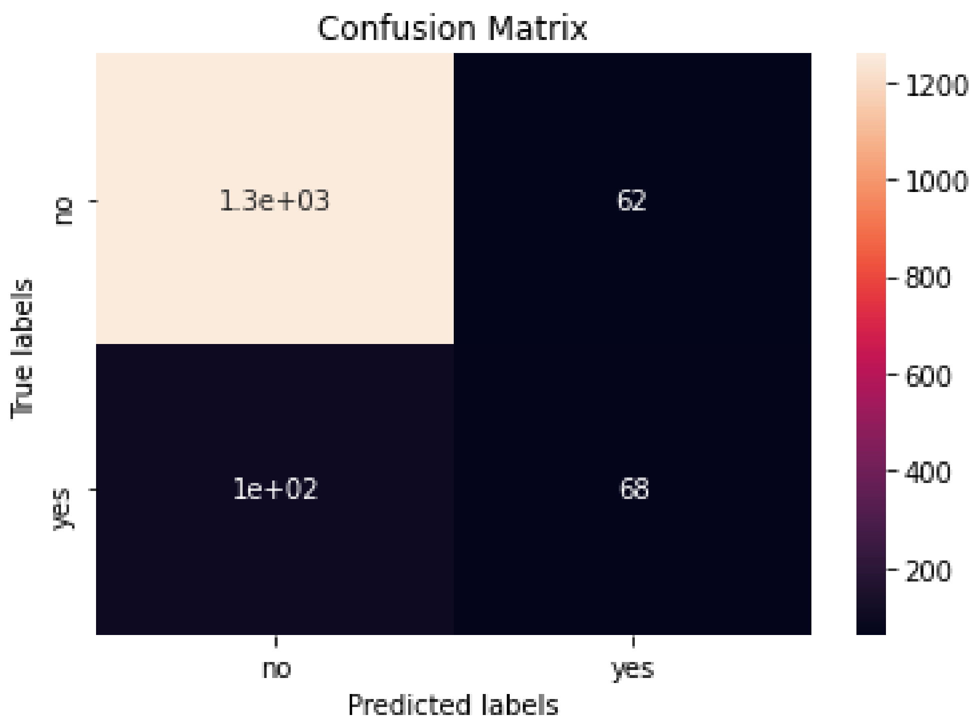
[334]:
```
max_depth=[3, 5, 6]
criterion=["gini","entropy"]
min_samples_split=[5,10,15]
param_grid = dict(max_depth=max_depth,criterion=criterion,␣
 ↪min_samples_split=min_samples_split)

random = RandomizedSearchCV(estimator=tree_model,␣
 ↪param_distributions=param_grid, cv = 3, n_jobs=-1, scoring='f1')

random_result = random.fit(X_train, y_train)
# Summarize results
print("Best: %f using %s" % (random_result.best_score_, random_result.
 ↪best_params_))

random.fit(X_train, y_train)

score = accuracy_score(y_test, random.predict(X_test))
f1 = f1_score(y_test, random.predict(X_test))
roc = roc_auc_score(y_test,random.predict(X_test))
recall = recall_score(y_test,random.predict(X_test))

print('Score:', score)
print('F1:', f1)
print('ROC:', roc)
print('Recall:', recall)

row_df = pd.DataFrame({'score':score, 'f1':f1, "roc":roc, "recall":recall},␣
 ↪index = ["RFC"])
scores = pd.concat([row_df, scores])
```

```
Best: 0.441759 using {'min_samples_split': 5, 'max_depth': 6, 'criterion':
'gini'}
```

```
Score: 0.8887399463806971
F1: 0.4503311258278146
ROC: 0.6741895701198026
Recall: 0.3953488372093023
```

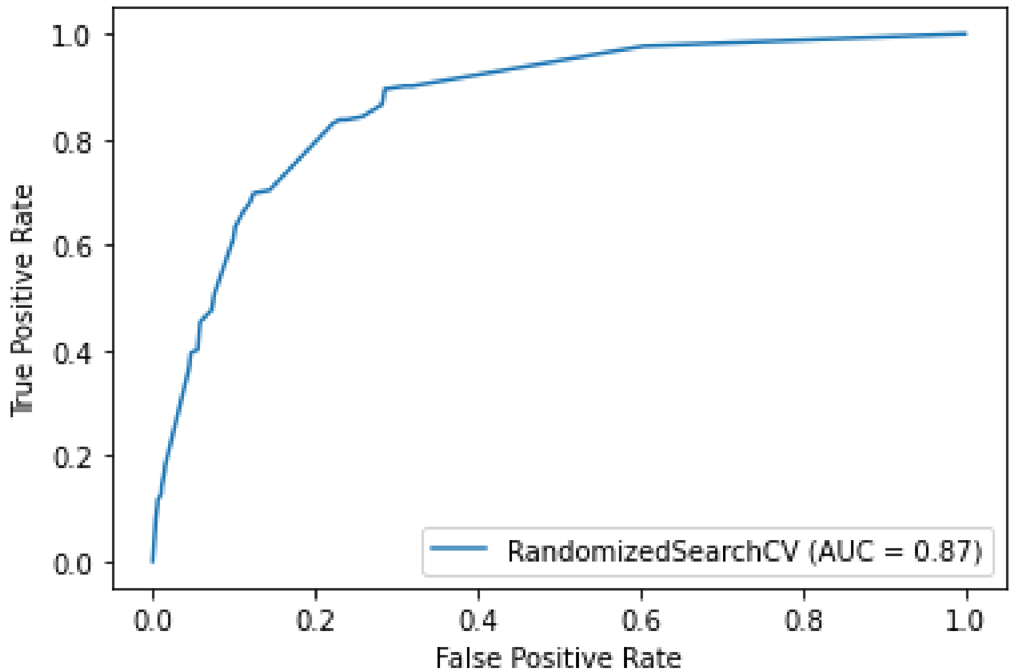
[335]:

```
cm = confusion_matrix(y_test, random.predict(X_test))

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['no', 'yes']); ax.yaxis.set_ticklabels(['no', 'yes']);
```



[336]:

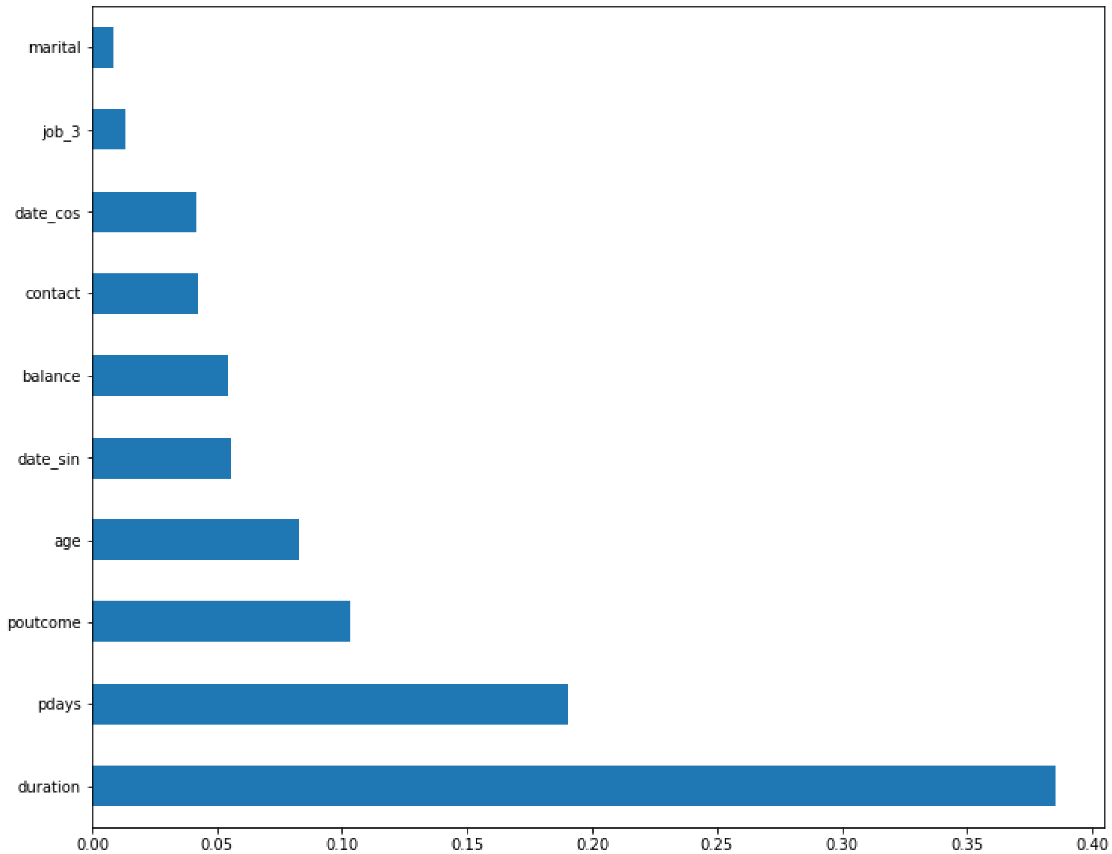
```
metrics.plot_roc_curve(random, X_test, y_test)
plt.show()
```



[373]:

```
plt.figure(figsize=(12,10))
model = random.best_estimator_
print(model.feature_importances_) #use inbuilt class feature_importances of tree␣
 ↪based classifiers
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```

```
[0.08258421 0.00830744 0.00232608 0.00620289 0.05436098 0.
 0.0019881  0.04244205 0.38527368 0.00541674 0.19039464 0.0065777
 0.10306954 0.05558017 0.04194222 0.         0.         0.
 0.01353357 0.        ]
```



4.0.4 Regresja logistyczna

[310]:
```
from sklearn.linear_model import LogisticRegression

lr = LogisticRegression()
lr
```

[310]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='lbfgs', tol=0.0001, verbose=0,
                   warm_start=False)
```

[375]:
```
C = [0.001,0.01,0.1,1,10,100,1000]
penalty = ['l1', 'l2', 'none']
dual = [True, False]
tol = [1e-4, 1e-5, 1e-6, 1e-3, 1e-2]
solver = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']
```

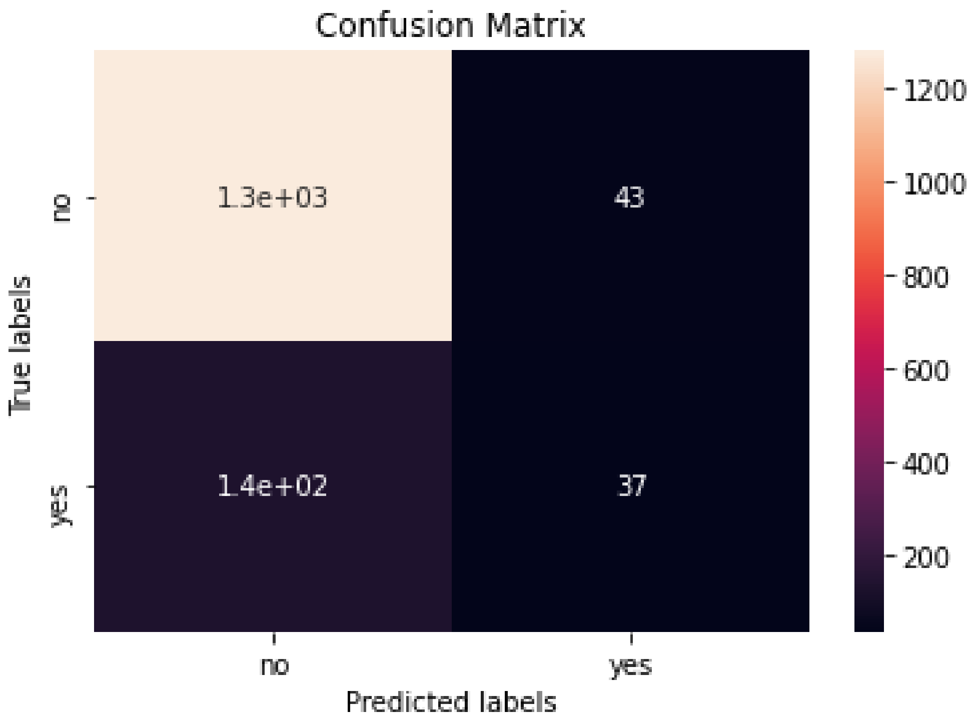
```
random_grid = {'C': C,
               'penalty': penalty,
               'dual': dual,
               'tol': tol,
               'solver': solver}




random = RandomizedSearchCV(estimator=lr, param_distributions=random_grid, cv =␣
 ↪3, n_jobs=-1, scoring='f1')

random_result = random.fit(X_train, y_train)
# Summarize results
print("Best: %f using %s" % (random_result.best_score_, random_result.
 ↪best_params_))

random.fit(X_train, y_train)

score = accuracy_score(y_test, random.predict(X_test))
f1 = f1_score(y_test, random.predict(X_test))
roc = roc_auc_score(y_test,random.predict(X_test))
recall = recall_score(y_test,random.predict(X_test))

print('Score:', score)
print('F1:', f1)
print('ROC:', roc)
print('Recall:', recall)

row_df = pd.DataFrame({'score':score, 'f1':f1, "roc":roc, "recall":recall},␣
 ↪index = ["LR"])
scores = pd.concat([row_df, scores])
```

```
Best: 0.320725 using {'tol': 0.001, 'solver': 'newton-cg', 'penalty': 'none',
'dual': False, 'C': 0.001}
Score: 0.8806970509383378
F1: 0.2936507936507936
ROC: 0.591270260747005
Recall: 0.21511627906976744
```

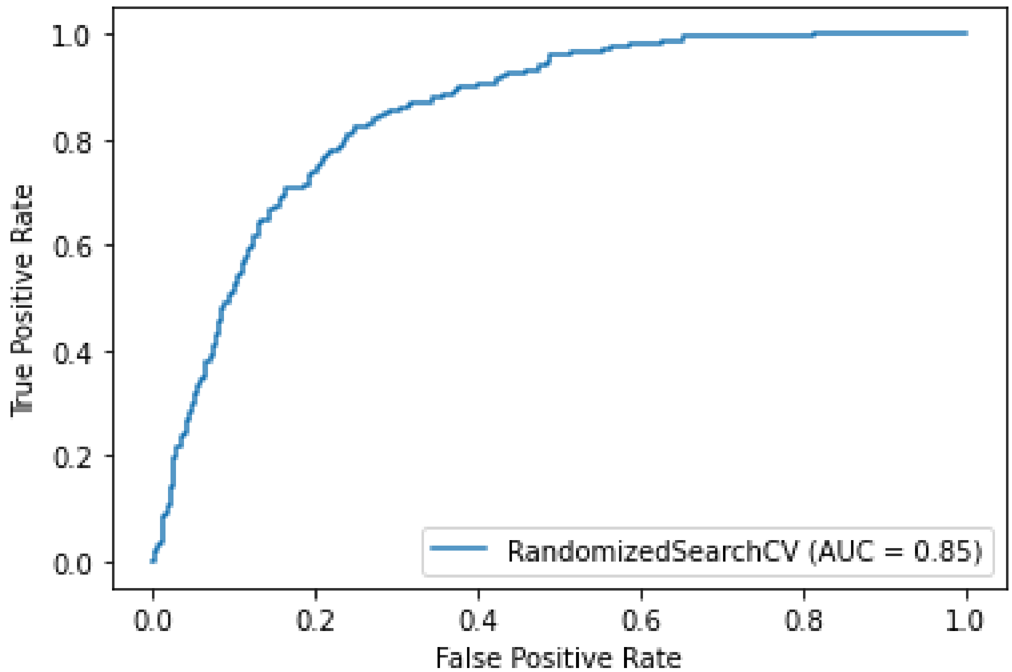
[376]:

```
cm = confusion_matrix(y_test, random.predict(X_test))

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
```

```
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['no', 'yes']); ax.yaxis.set_ticklabels(['no', 'yes']);
```



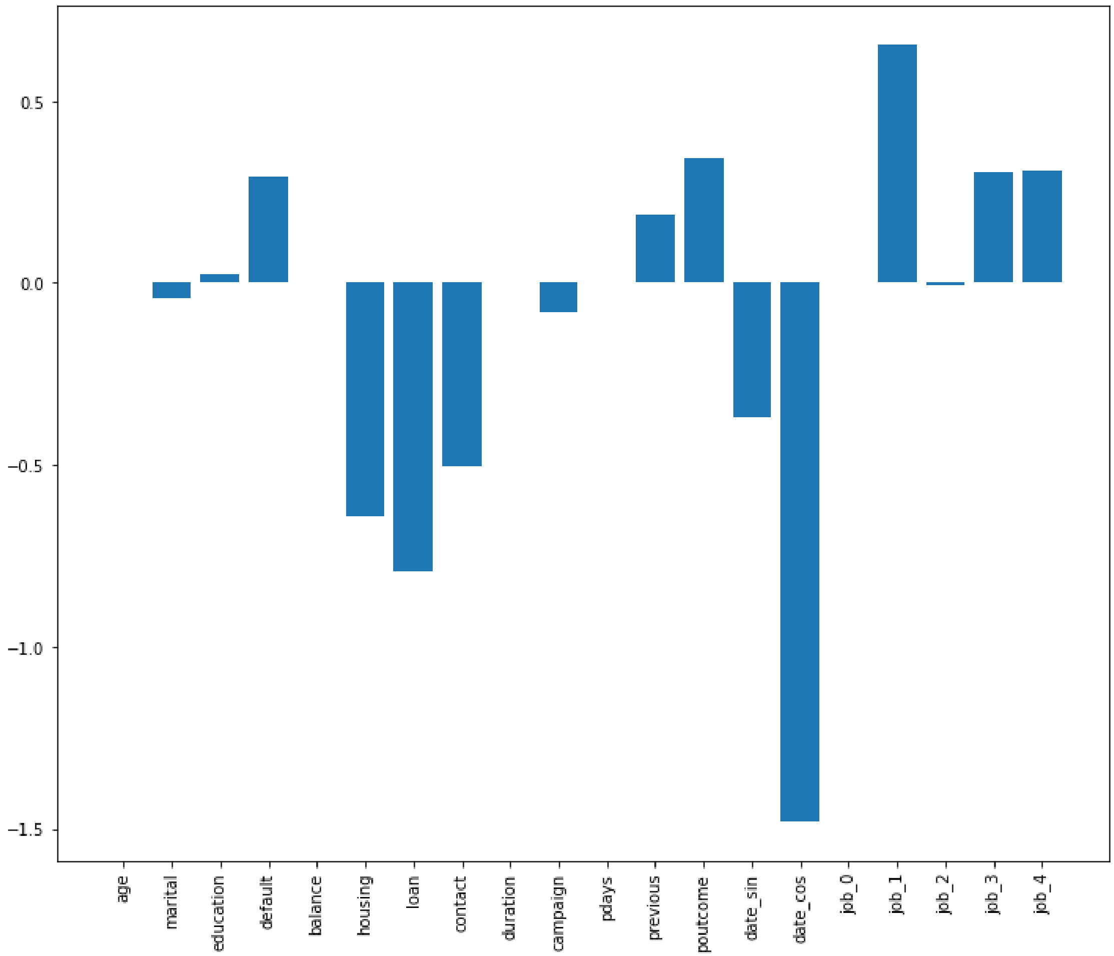
[377]:
```
metrics.plot_roc_curve(random, X_test, y_test)
plt.show()
```



[378]:
```
plt.figure(figsize=(12,10))

# logistic regression for feature importance
from sklearn.datasets import make_classification

model = random.best_estimator_
# get importance
importance = model.coef_[0]
# summarize feature importance
for i,v in enumerate(importance):
        print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
plt.bar([x for x in range(len(importance))], importance, tick_label = X.columns)
plt.xticks(rotation='vertical')
plt.show()
```

```
Feature: 0, Score: 0.00057
Feature: 1, Score: -0.04030
Feature: 2, Score: 0.02481
Feature: 3, Score: 0.29312
Feature: 4, Score: 0.00003
Feature: 5, Score: -0.64061
Feature: 6, Score: -0.79511
Feature: 7, Score: -0.50312
Feature: 8, Score: 0.00337
```

```
Feature: 9, Score: -0.08197
Feature: 10, Score: 0.00368
Feature: 11, Score: 0.18833
Feature: 12, Score: 0.34371
Feature: 13, Score: -0.37084
Feature: 14, Score: -1.48199
Feature: 15, Score: 0.00000
Feature: 16, Score: 0.65418
Feature: 17, Score: -0.00629
Feature: 18, Score: 0.30404
Feature: 19, Score: 0.30999
```



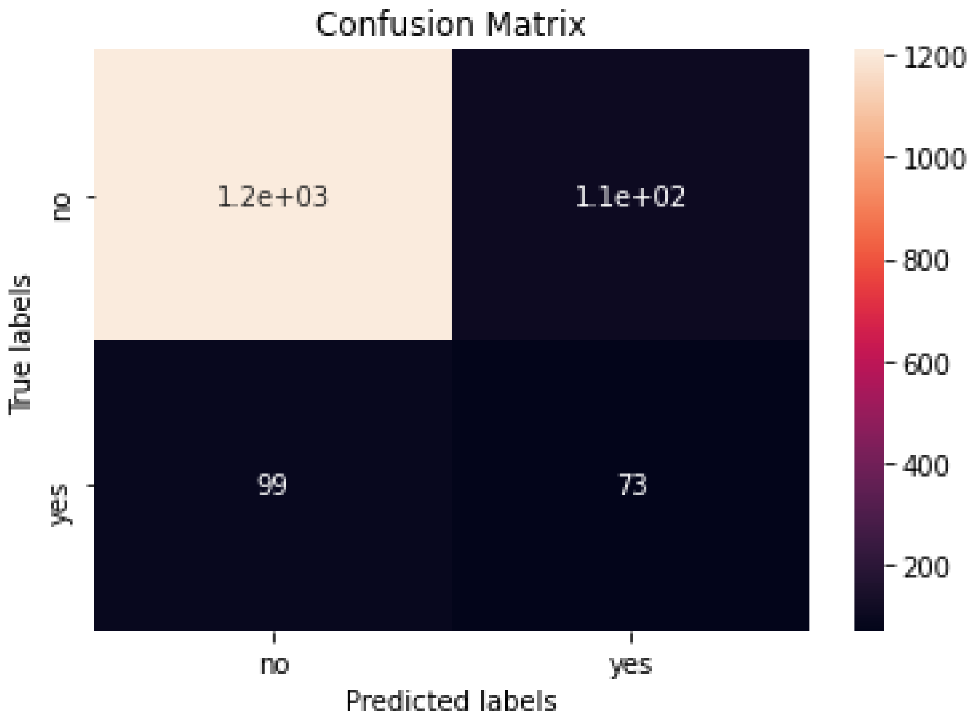
4.0.5 Naiwny klasyfikator Bayesowski

[309]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
```

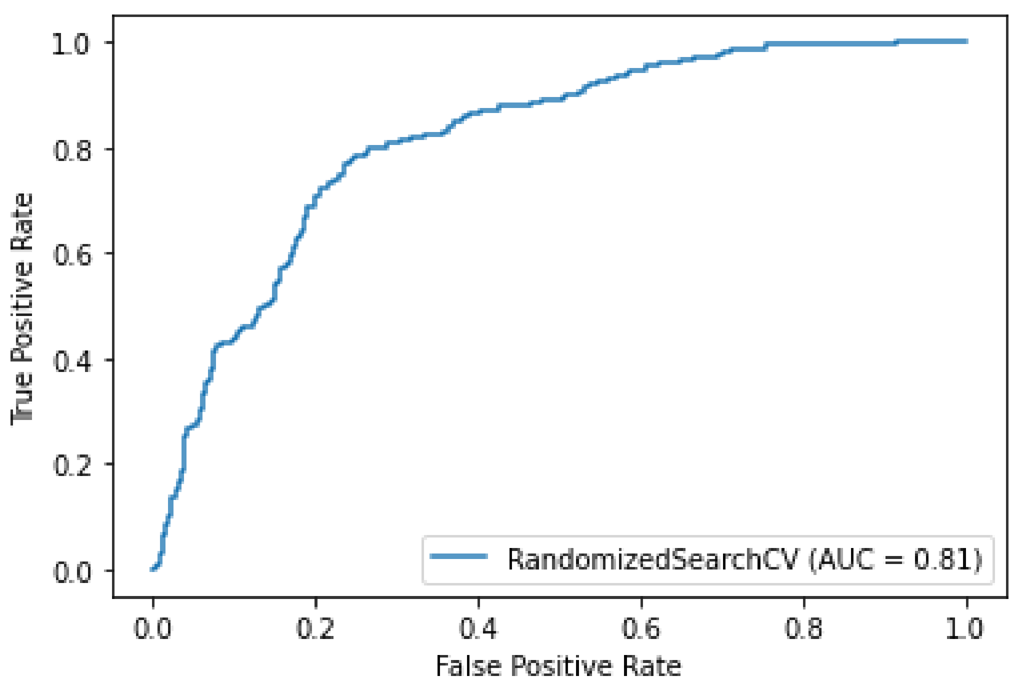
[290]:
```
var_smoothing = [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-10, 1e-11, 1e-12, 1e-13]

random_grid = {'var_smoothing': var_smoothing}


random = RandomizedSearchCV(estimator=gnb, param_distributions=random_grid, cv =␣
 ↪3, n_jobs=-1, scoring='f1')

random_result = random.fit(X_train, y_train)
# Summarize results
print("Best: %f using %s" % (random_result.best_score_, random_result.
 ↪best_params_))

random.fit(X_train, y_train)

score = accuracy_score(y_test, random.predict(X_test))
f1 = f1_score(y_test, random.predict(X_test))
roc = roc_auc_score(y_test,random.predict(X_test))
recall = recall_score(y_test,random.predict(X_test))

print('Score:', score)
print('F1:', f1)
print('ROC:', roc)
print('Recall:', recall)

row_df = pd.DataFrame({'score':score, 'f1':f1, "roc":roc, "recall":recall},␣
 ↪index = ["GNB"])
scores = pd.concat([row_df, scores])
```

```
Best: 0.431779 using {'var_smoothing': 1e-06}
Score: 0.8592493297587132
F1: 0.41011235955056174
ROC: 0.6701638477801268
Recall: 0.42441860465116277
```

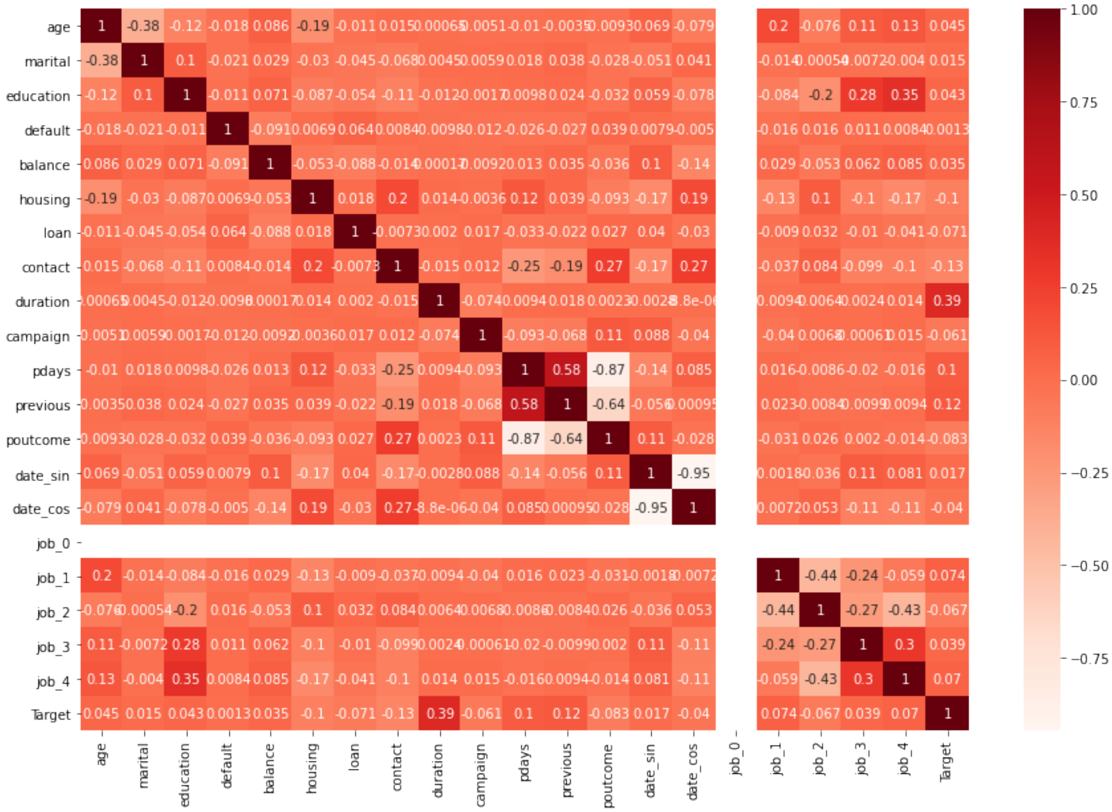
[291]:
```
cm = confusion_matrix(y_test, random.predict(X_test))

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
ax.xaxis.set_ticklabels(['no', 'yes']); ax.yaxis.set_ticklabels(['no', 'yes']);
```



[292]:
```
metrics.plot_roc_curve(random, X_test, y_test)
plt.show()
```



[387]:
```
X_y = X.copy()
X_y['Target'] = y

plt.figure(figsize=(15,10))
cor = X_y.corr()
sns.heatmap(cor, annot=True, cmap=plt.cm.Reds)
plt.show()
```



4.0.6 XGB Classifier

[379]:
```
from xgboost import XGBClassifier

model=XGBClassifier()

random_state = [1, 10, 100]
learning_rate = [0.01, 0.05, 0.1, 0.25, 0.001, 0.005]
booster = ['gbtree', 'gblinear']
nround = [100,200, 50, 10]
max_depth = [1,4,5,10]
```

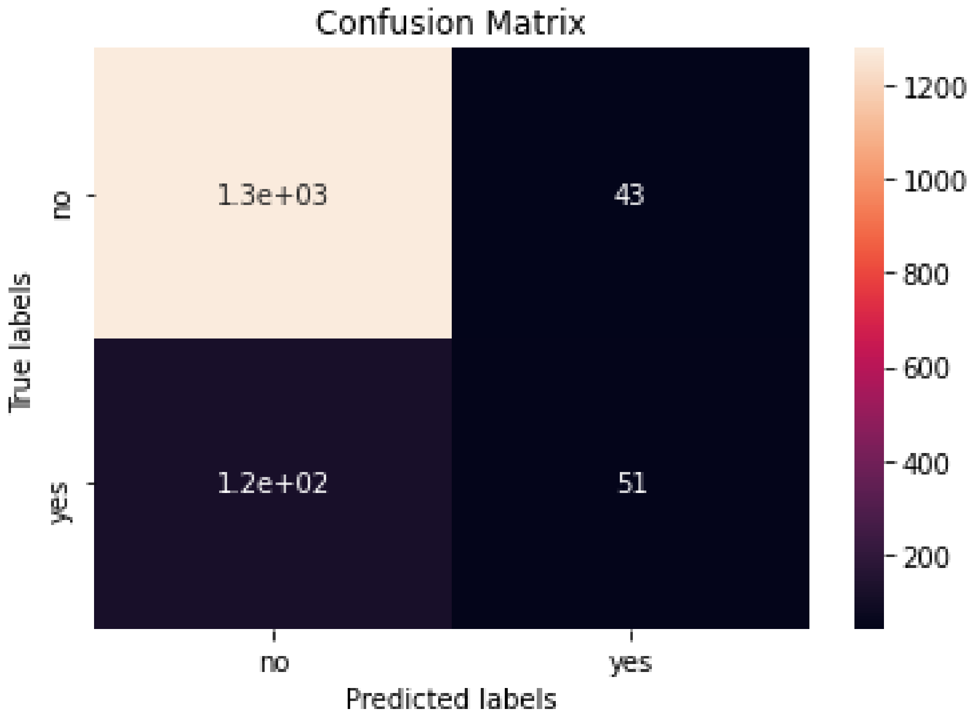
```
random_grid = {'random_state': random_state,
               'learning_rate': learning_rate,
               'booster': booster,
               'nround': nround,
               'max_depth': max_depth}



random = RandomizedSearchCV(estimator=model, param_distributions=random_grid, cv␣
 ↪= 3, n_jobs=-1, scoring='f1')

random_result = random.fit(X_train, y_train)
# Summarize results
print("Best: %f using %s" % (random_result.best_score_, random_result.
 ↪best_params_))

random.fit(X_train, y_train)

score = accuracy_score(y_test, random.predict(X_test))
f1 = f1_score(y_test, random.predict(X_test))
roc = roc_auc_score(y_test,random.predict(X_test))
recall = recall_score(y_test,random.predict(X_test))

print('Score:', score)
print('F1:', f1)
print('ROC:', roc)
print('Recall:', recall)

row_df = pd.DataFrame({'score':score, 'f1':f1, "roc":roc, "recall":recall},␣
 ↪index = ["XGBC"])
scores = pd.concat([row_df, scores])
```

```
Best: 0.429622 using {'random_state': 100, 'nround': 200, 'max_depth': 10,
'learning_rate': 0.25, 'booster': 'gbtree'}
Score: 0.8900804289544236
F1: 0.38345864661654144
ROC: 0.6319679351656097
Recall: 0.29651162790697677
```

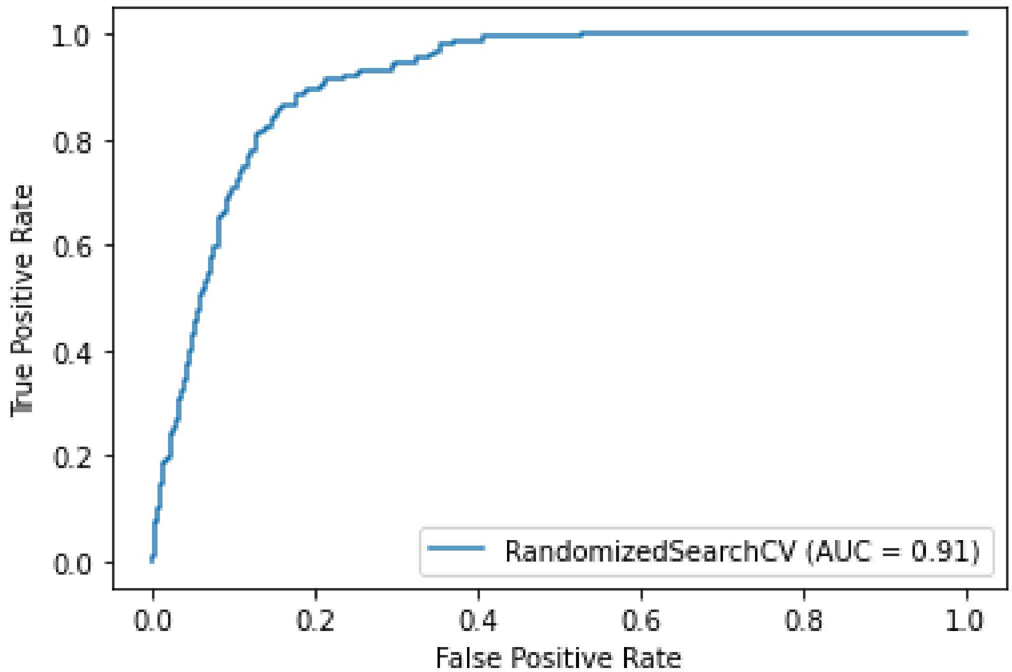
[380]:

```
cm = confusion_matrix(y_test, random.predict(X_test))

ax= plt.subplot()
sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells

# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix');
```

```
ax.xaxis.set_ticklabels(['no', 'yes']); ax.yaxis.set_ticklabels(['no', 'yes']);
```

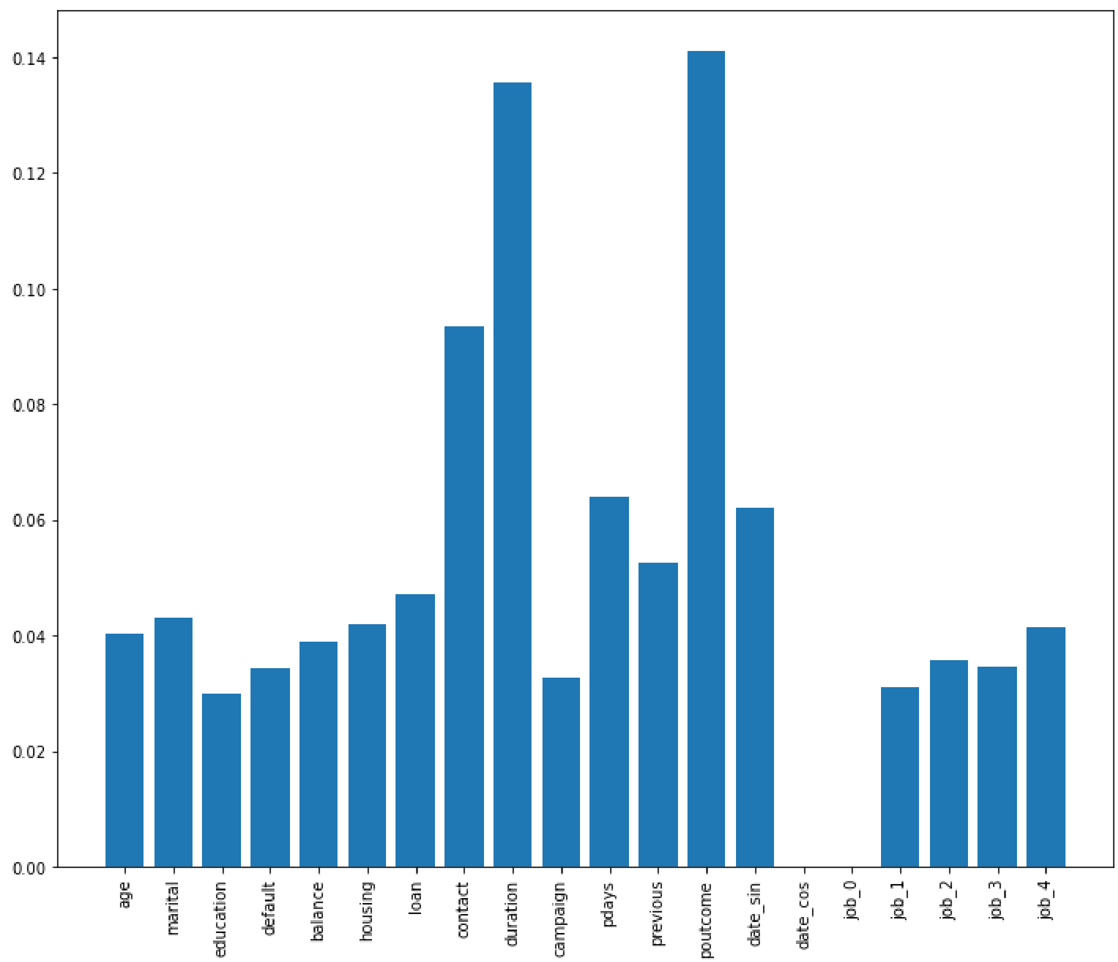


[381]:
```
metrics.plot_roc_curve(random, X_test, y_test)
plt.show()
```



[385]:
```
plt.figure(figsize=(12,10))

model = random.best_estimator_

print(model.feature_importances_)
# plot
plt.bar(range(len(model.feature_importances_)), model.feature_importances_,␣
 ↪tick_label = X.columns)
plt.xticks(rotation = '90')
plt.show()
```

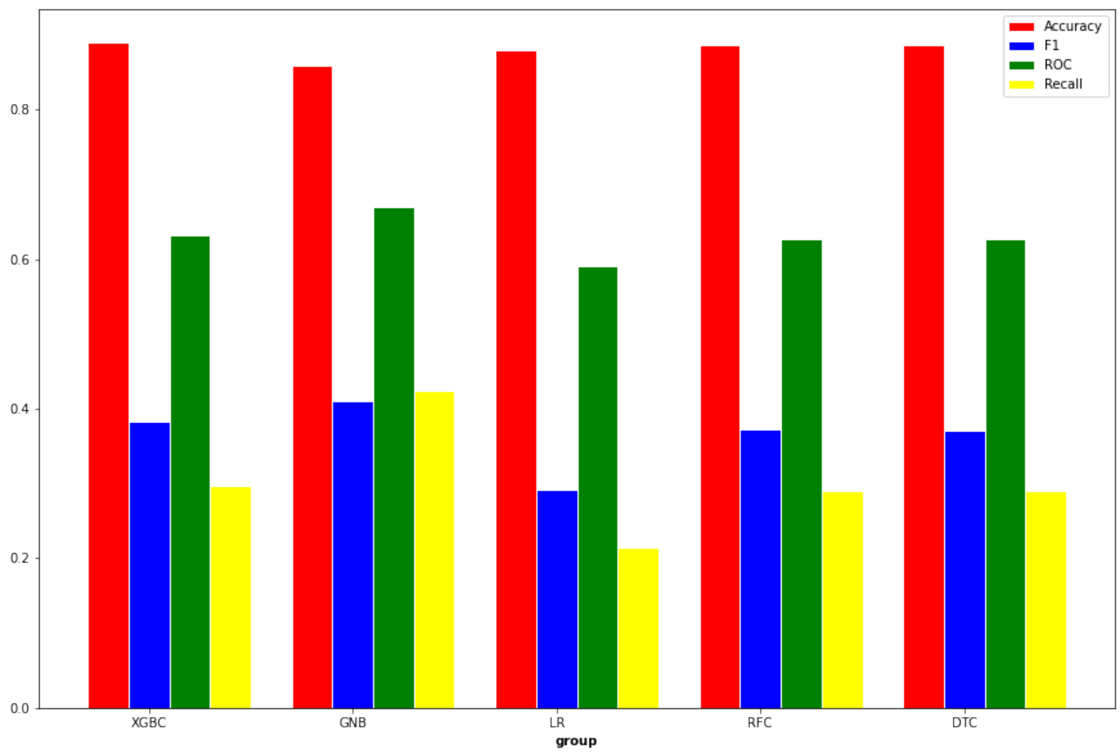
```
[0.04026253 0.04312887 0.02999257 0.0344202  0.03898999 0.04187821
 0.04722867 0.09337045 0.13558881 0.03268411 0.06398384 0.05254117
 0.14097984 0.06220829 0.         0.         0.03111406 0.03574214
 0.03456594 0.04132028]
```



4.0.7 Podsumowanie

[297]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(15,10))

# set width of bar
barWidth = 0.2

# set height of bar
bars1 = scores['score']
bars2 = scores['f1']
bars3 = scores['roc']
bars4 = scores['recall']

# Set position of bar on X axis
r1 = np.arange(len(bars1))
```

```
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]
r4 = [x + barWidth for x in r3]

# Make the plot
plt.bar(r1, bars1, color='red', width=barWidth, edgecolor='white',␣
 ↪label='Accuracy')
plt.bar(r2, bars2, color='blue', width=barWidth, edgecolor='white', label='F1')
plt.bar(r3, bars3, color='green', width=barWidth, edgecolor='white', label='ROC')
plt.bar(r4, bars4, color='yellow', width=barWidth, edgecolor='white',␣
 ↪label='Recall')


# Add xticks on the middle of the group bars
plt.xlabel('group', fontweight='bold')
plt.xticks([r + barWidth for r in range(len(bars1))], scores.index)

# Create legend & Show graphic
plt.legend()
plt.show()
```



[298]:
```
scores
```

[298]:

	score	f1	roc	recall
XGBC	0.890080	0.383459	0.631968	0.296512
GNB	0.859249	0.410112	0.670164	0.424419
LR	0.880027	0.292490	0.590891	0.215116
RFC	0.887399	0.373134	0.627925	0.290698
DTC	0.886729	0.371747	0.627546	0.290698

Zdecydowaliśmy się wybrać GNB, ponieważ ma jedne z najlepszych wyników, szczególnie warto zwrócić uwagę na Recall - lepiej zadzwonić do klienta, który odrzuci ofertę, niż nie zadzwonić do osoby, która ofertę by przyjęła.