Aula 03 - RandomForest - Exercício

Exercício 1

Utilizando o dataset breast_cancer_train.csv, desenvolva um modelo RandomForest (utilizando o sklearn) com o objetivo de prever se o resultado de uma biópsia indica a presença de câncer malígno. Procure fazer com que o seu o modelo não apresente **overfitting** e maximize a acurácia.

Ao final, reporte:

1 - A acurácia, precisão e recall do seu modelo na base utilizada para treino e validação
2 - A acurácia, precisão e recall do seu modelo na base breast_cancer_test.csv
3 - Os parâmetros do modelo
4 - Imagine que o hospital queira diminuir custos no ano seguinte e pretende deixar de colher algumas das variáveis. Quais variáveis você recomendaria a exclusão? Justifique.

In [1]:
```
import numpy as np
import pandas as pd
import random
import seaborn as sns

from matplotlib import pyplot as plt
from mlxtend.plotting import plot_confusion_matrix
from scipy import stats
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

In [2]:
```
df_cancer_train = pd.read_csv('breast_cancer_train.csv')
df_cancer_test = pd.read_csv('breast_cancer_test.csv')
```

In [3]:
```
# Uma olhada no dataset
df_cancer_train.head()
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	pe
0	857156	B	13.49	22.30	86.91	561.0	0.08752	0.07698	0.047510	0.033840	...	15.15	31.82	
1	844981	M	13.00	21.82	87.50	519.8	0.12730	0.19320	0.185900	0.093530	...	15.49	30.73	
2	88330202	M	17.46	39.28	113.40	920.6	0.09812	0.12980	0.141700	0.088110	...	22.51	44.87	
3	88203002	B	11.22	33.81	70.79	386.8	0.07780	0.03574	0.004967	0.006434	...	12.36	41.78	
4	892189	M	11.76	18.14	75.00	431.1	0.09968	0.05914	0.026850	0.035150	...	13.36	23.39	

5 rows × 32 columns

In [4]:

```
# Distintos valores para a variável-alvo
df_cancer_train['diagnosis'].value_counts()
```

Out[4]:

```
B    292
M    163
Name: diagnosis, dtype: int64
```

In [5]:

```
# Mapeando a variável alvo para valores numéricos
df_cancer_train['diagnosis'] = df_cancer_train['diagnosis'].apply(lambda x : 1 if x == 'M' else 0)
df_cancer_test['diagnosis'] = df_cancer_test['diagnosis'].apply(lambda x : 1 if x == 'M' else 0)
display(df_cancer_train.head(5))
display(df_cancer_test.head(5))
```

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	pe
0	857156	0	13.49	22.30	86.91	561.0	0.08752	0.07698	0.047510	0.033840	...	15.15	31.82	
1	844981	1	13.00	21.82	87.50	519.8	0.12730	0.19320	0.185900	0.093530	...	15.49	30.73	
2	88330202	1	17.46	39.28	113.40	920.6	0.09812	0.12980	0.141700	0.088110	...	22.51	44.87	
3	88203002	0	11.22	33.81	70.79	386.8	0.07780	0.03574	0.004967	0.006434	...	12.36	41.78	
4	892189	1	11.76	18.14	75.00	431.1	0.09968	0.05914	0.026850	0.035150	...	13.36	23.39	

5 rows × 32 columns

	id	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_mean	compactness_mean	concavity_mean	concave points_mean	...	radius_worst	texture_worst	pe
0	84348301	1	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	...	14.91	26.50	
1	84358402	1	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	...	22.54	16.67	
2	843786	1	12.45	15.70	82.57	477.1	0.12780	0.17000	0.15780	0.08089	...	15.47	23.75	
3	846226	1	19.17	24.80	132.40	1123.0	0.09740	0.24580	0.20650	0.11180	...	20.96	29.94	
4	855133	1	14.99	25.20	95.54	698.8	0.09387	0.05131	0.02398	0.02899	...	14.99	25.20	

5 rows × 32 columns

In [6]:
```
# Verificando se há dados faltantes
df_cancer_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 455 entries, 0 to 454
Data columns (total 32 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   id                       455 non-null    int64
 1   diagnosis                455 non-null    int64
 2   radius_mean              455 non-null    float64
 3   texture_mean             455 non-null    float64
 4   perimeter_mean           455 non-null    float64
 5   area_mean                455 non-null    float64
 6   smoothness_mean          455 non-null    float64
 7   compactness_mean         455 non-null    float64
 8   concavity_mean           455 non-null    float64
 9   concave points_mean      455 non-null    float64
 10  symmetry_mean            455 non-null    float64
 11  fractal_dimension_mean   455 non-null    float64
 12  radius_se                455 non-null    float64
 13  texture_se               455 non-null    float64
 14  perimeter_se             455 non-null    float64
 15  area_se                  455 non-null    float64
 16  smoothness_se            455 non-null    float64
 17  compactness_se           455 non-null    float64
 18  concavity_se             455 non-null    float64
 19  concave points_se        455 non-null    float64
 20  symmetry_se              455 non-null    float64
 21  fractal_dimension_se     455 non-null    float64
 22  radius_worst             455 non-null    float64
 23  texture_worst            455 non-null    float64
 24  perimeter_worst          455 non-null    float64
 25  area_worst               455 non-null    float64
 26  smoothness_worst         455 non-null    float64
 27  compactness_worst        455 non-null    float64
 28  concavity_worst          455 non-null    float64
 29  concave points_worst     455 non-null    float64
 30  symmetry_worst           455 non-null    float64
 31  fractal_dimension_worst  455 non-null    float64
dtypes: float64(30), int64(2)
memory usage: 113.9 KB
```

Sem dados faltantes

In [7]:

```
# Verificando se temos registros duplicados...
df_cancer_train.drop_duplicates()['id'].count()
```

Out[7]:

```
455
```

Sem registros duplicados

In [8]:

```
# Segregando a variável alvo dos dados
```

```python
X = df_cancer_train.drop(['diagnosis'], axis=1)
y = df_cancer_train['diagnosis']
```

In [9]:
```python
# Obtendo os datasets de treino e de validação
X_train, X_valid, y_train, y_valid = train_test_split(X,
                                                      y,
                                                      test_size=0.2,
                                                      random_state=10)
```

In [10]:
```python
print(X_train.shape, y_train.shape)
print(X_valid.shape, y_valid.shape)
```

```
(364, 31) (364,)
(91, 31) (91,)
```

In [11]:
```python
modelo = RandomForestClassifier(n_estimators=100, max_depth=3, random_state=10)
modelo
```

Out[11]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=3, random_state=10)
```

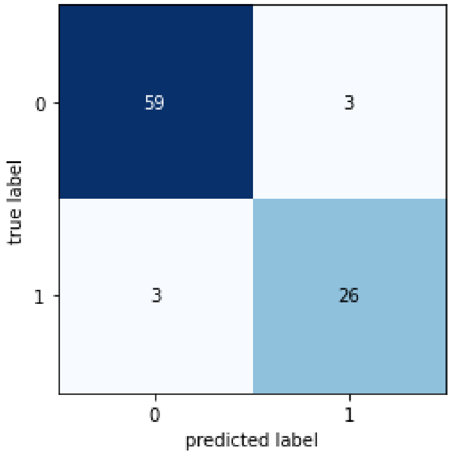
In [12]:
```python
modelo.fit(X_train, y_train)
```

Out[12]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=3, random_state=10)
```

In [13]:
```python
y_train_pred = modelo.predict(X_train)
y_valid_pred = modelo.predict(X_valid)
print(y_train_pred.shape, y_valid_pred.shape)
```

```
(364,) (91,)
```

In [14]:
```python
cm = confusion_matrix(y_valid, y_valid_pred)
plot_confusion_matrix(conf_mat=cm)
plt.show()
```



In [15]:
```
acc_train  = accuracy_score(y_train, y_train_pred)
prec_train = precision_score(y_train, y_train_pred)
rec_train  = recall_score(y_train, y_train_pred)

acc_valid  = accuracy_score(y_valid, y_valid_pred)
prec_valid = precision_score(y_valid, y_valid_pred)
rec_valid  = recall_score(y_valid, y_valid_pred)
```

Respostas:

1 - A acurácia, precisão e recall do seu modelo na base utilizada para treino e validação

In [16]:
```
print(f'Treino:\nAcc: {acc_train:.2f}, Precision: {prec_train:.2f}, Recall: {rec_train:.2f}')
print(f'Validação:\nAcc: {acc_valid:.2f}, Precision: {prec_valid:.2f}, Recall: {rec_valid:.2f}')
```

```
Treino:
Acc: 0.98, Precision: 0.99, Recall: 0.96
Validação:
Acc: 0.93, Precision: 0.90, Recall: 0.90
```

2 - A acurácia, precisão e recall do seu modelo na base breast_cancer_test.csv

In [17]:
```
X_test = df_cancer_test.drop(['diagnosis'], axis=1)
y_test = df_cancer_test['diagnosis']

y_test_pred = modelo.predict(X_test)

acc_test  = accuracy_score(y_test, y_test_pred)
prec_test = precision_score(y_test, y_test_pred)
rec_test  = recall_score(y_test, y_test_pred)

print(f'Teste:\nAcc: {acc_test:.2f}, Precision: {prec_test:.2f}, Recall: {rec_test:.2f}')
```

```
Teste:
Acc: 0.93, Precision: 0.94, Recall: 0.90
```

3 - Os parâmetros do modelo

estimators=100

max_depth=3

random_state=10

4 - Imagine que o hospital queira diminuir custos no ano seguinte e pretende deixar de colher algumas das variáveis. Quais variáveis você recomendaria a exclusão? Justifique.

In [18]:
```
df = pd.DataFrame(np.reshape([X_test.columns.values, modelo.feature_importances_], (2,31)).T)
df = df.sort_values(by=1, ascending=True)
df.head(10)
```

Out[18]:

	0	1
15	smoothness_se	0.000669
19	symmetry_se	0.000854
10	fractal_dimension_mean	0.001091
0	id	0.00133
16	compactness_se	0.001659
9	symmetry_mean	0.001684
12	texture_se	0.002569
20	fractal_dimension_se	0.002817
2	texture_mean	0.003045
18	concave points_se	0.004132

Recomendaria as variáveis acima, na ordem, pois são as que representam menor importância no modelo de predição.

Exercício 2

A partir do dataset abaixo, utilize um modelo **RandomForest** para prever, a partir das variáveis de entrada, a probabilidade do individuo receber mais de $50k por ano. Defina um ponto de corte na probabilidade em que, sempre que a probabilidade for maior que o corte, a classe será predita como 1.

Uma descrição do dataset pode ser encontrada aqui.

Ao final do exercício, reporte:

1 - O tamanho da base utilizada para treino/validação e o tamanho da base utilizada para teste;

2 - Um gráfico de barras com a importância de cada variável no precesso de predição;

3 - A acurácia, precisão e recall de treino, validação e teste;

4 - Uma comparação com o desempenho de uma árvore de decisão simples.

Importante: lembre-se de fazer a divisão do conjunto de teste antes de iniciar o exercício!

In [19]:
```
import pandas as pd
```

In [20]:
```
adult_dataset = pd.read_csv('adult.csv')

print(adult_dataset.shape)

adult_dataset.head()
```

```
(48842, 15)
```

Out[20]:

	age	workclass	fnlwgt	education	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	11th	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	HS-grad	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	Assoc-acdm	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K
3	44	Private	160323	Some-college	10	Married-civ-spouse	Machine-op-inspct	Husband	Black	Male	7688	0	40	United-States	>50K
4	18	?	103497	Some-college	10	Never-married	?	Own-child	White	Female	0	0	30	United-States	<=50K

In [21]:
```
for col in adult_dataset.columns.values:
    print(f'Coluna: {col}:\n')
    display(adult_dataset[col].value_counts())
```

```
Coluna: age:
```

```
36    1348
35    1337
33    1335
23    1329
31    1325
      ...
88       6
85       5
87       3
89       2
86       1
Name: age, Length: 74, dtype: int64
Coluna: workclass:

Private             33906
Self-emp-not-inc     3862
Local-gov            3136
?                    2799
State-gov            1981
Self-emp-inc         1695
Federal-gov          1432
Without-pay            21
Never-worked           10
Name: workclass, dtype: int64
Coluna: fnlwgt:

203488    21
190290    19
120277    19
125892    18
126569    18
          ..
188488     1
285290     1
293579     1
114874     1
257302     1
Name: fnlwgt, Length: 28523, dtype: int64
Coluna: education:
```

```
HS-grad         15784
Some-college    10878
Bachelors        8025
Masters          2657
Assoc-voc        2061
11th             1812
Assoc-acdm       1601
10th             1389
7th-8th           955
Prof-school       834
9th               756
12th              657
Doctorate         594
5th-6th           509
1st-4th           247
Preschool          83
Name: education, dtype: int64
Coluna: educational-num:

9     15784
10    10878
13     8025
14     2657
11     2061
7      1812
12     1601
6      1389
4       955
15      834
5       756
8       657
16      594
3       509
2       247
1        83
Name: educational-num, dtype: int64
Coluna: marital-status:

Married-civ-spouse       22379
Never-married            16117
Divorced                  6633
Separated                 1530
Widowed                   1518
Married-spouse-absent      628
Married-AF-spouse           37
Name: marital-status, dtype: int64
Coluna: occupation:
```

```
Prof-specialty       6172
Craft-repair         6112
Exec-managerial      6086
Adm-clerical         5611
Sales                5504
Other-service        4923
Machine-op-inspct    3022
?                    2809
Transport-moving     2355
Handlers-cleaners    2072
Farming-fishing      1490
Tech-support         1446
Protective-serv       983
Priv-house-serv       242
Armed-Forces           15
Name: occupation, dtype: int64
Coluna: relationship:

Husband           19716
Not-in-family     12583
Own-child          7581
Unmarried          5125
Wife               2331
Other-relative     1506
Name: relationship, dtype: int64
Coluna: race:

White                 41762
Black                  4685
Asian-Pac-Islander     1519
Amer-Indian-Eskimo      470
Other                   406
Name: race, dtype: int64
Coluna: gender:

Male      32650
Female    16192
Name: gender, dtype: int64
Coluna: capital-gain:

0        44807
15024      513
7688       410
7298       364
99999      244
         ...
1111         1
7262         1
22040        1
1639         1
2387         1
Name: capital-gain, Length: 123, dtype: int64
```

```
Coluna: capital-loss:

0       46560
1902      304
1977      253
1887      233
2415       72
        ...
2465        1
2080        1
155         1
1911        1
2201        1
Name: capital-loss, Length: 99, dtype: int64
Coluna: hours-per-week:

40    22803
50     4246
45     2717
60     2177
35     1937
      ...
69        1
87        1
94        1
82        1
79        1
Name: hours-per-week, Length: 96, dtype: int64
Coluna: native-country:
```

```
United-States                 43832
Mexico                          951
?                               857
Philippines                     295
Germany                         206
Puerto-Rico                     184
Canada                          182
El-Salvador                     155
India                           151
Cuba                            138
England                         127
China                           122
South                           115
Jamaica                         106
Italy                           105
Dominican-Republic              103
Japan                            92
Guatemala                        88
Poland                           87
Vietnam                          86
Columbia                         85
Haiti                            75
Portugal                         67
Taiwan                           65
Iran                             59
Greece                           49
Nicaragua                        49
Peru                             46
Ecuador                          45
France                           38
Ireland                          37
Hong                             30
Thailand                         30
Cambodia                         28
Trinadad&Tobago                  27
Laos                             23
Yugoslavia                       23
Outlying-US(Guam-USVI-etc)       23
Scotland                         21
Honduras                         20
Hungary                          19
Holand-Netherlands                1
Name: native-country, dtype: int64
Coluna: income:

<=50K    37155
>50K     11687
Name: income, dtype: int64
```

Análise das Colunas:

- age: OK
- workclass: 2799 valores não informados a tratar
- fnlwgt: OK
- education: pode ser desconsiderada para predições, pois temos uma correspondente numérica: educational-num
- educational-num: OK
- marital-status: necessário mapeamento.
- occupation: 2809 valores não informados a tratar. Necessário mapeamento
- relationship: necessário mapeamento
- race: necessário mapeamento
- gender: necessário mapeamento
- capital-gain: OK
- capital-loss: OK
- hours-per-week: OK
- native-country: 857 valores não informados a tratar. Necessário mapeamento
- income: necessário mapeamento.

In [22]:

```
df_na = adult_dataset.copy()

q_na_workclass  = len(adult_dataset.loc[adult_dataset['workclass'] == '?'])
q_na_occupation = len(adult_dataset.loc[adult_dataset['occupation'] == '?'])
q_na_native_c   = len(adult_dataset.loc[adult_dataset['native-country'] == '?'])

df_na['workclass'] = df_na['workclass'].apply(lambda x : np.nan if x == '?' else x)
df_na['occupation'] = df_na['occupation'].apply(lambda x : np.nan if x == '?' else x)
df_na['native-country'] = df_na['native-country'].apply(lambda x : np.nan if x == '?' else x)
df_na = df_na.dropna()
```

In [23]:

```
q_tot = len(adult_dataset)
q_na  = q_tot - len(df_na)
print(f'Total de registros: {q_tot}, Registros sem dados: {q_na} = {float(q_na) / q_tot * 100:.2f}%')
print(f'workclass.....: {q_na_workclass:5.0f} = {float(q_na_workclass) / q_tot * 100:4.2f}%')
print(f'occupation....: {q_na_occupation:5.0f} = {float(q_na_occupation) / q_tot * 100:4.2f}%')
print(f'native-country: {q_na_native_c:5.0f} = {float(q_na_native_c) / q_tot * 100:4.2f}%')
```

```
Total de registros: 48842, Registros sem dados: 3620 = 7.41%
workclass.....:  2799 = 5.73%
occupation....:  2809 = 5.75%
native-country:   857 = 1.75%
```

Decisão:

Vamos definir uma categoria 'X' para workclass e occupation ausentes, e vamos excluir os registros com país nativo ausente (native-country) pois representam menos de 2% da base.

In [24]:

```python
# Liberando memoria
df_na = None
# Uma cópia para preservar o Dataset original
df_ad = adult_dataset.copy()
df_ad['workclass'     ] = df_ad['workclass'     ].apply(lambda x : x if x != '?' else 'X')
df_ad['occupation'    ] = df_ad['occupation'    ].apply(lambda x : x if x != '?' else 'X')
df_ad['native-country'] = df_ad['native-country'].apply(lambda x : x if x != '?' else np.nan)
```

In [25]:

```python
df_ad = df_ad.dropna()
q_tot = len(df_ad)
print(f'Total de registros apos limpeza: {q_tot}')
```

```
Total de registros apos limpeza: 47985
```

In [26]:

```python
# Excluindo a coluna 'education' para a qual já existe uma categorica numérica correspondente
df_ad = df_ad.drop('education', axis=1)
df_ad.head(3)
```

Out[26]:

	age	workclass	fnlwgt	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	Private	226802	7	Never-married	Machine-op-inspct	Own-child	Black	Male	0	0	40	United-States	<=50K
1	38	Private	89814	9	Married-civ-spouse	Farming-fishing	Husband	White	Male	0	0	50	United-States	<=50K
2	28	Local-gov	336951	12	Married-civ-spouse	Protective-serv	Husband	White	Male	0	0	40	United-States	>50K

In [27]:

```python
# Aplicando os mapeamentos

''' Função que gera dict de mapeamento para uma lista fornecida '''
def apply_map(df, col):
    lista = df[col].drop_duplicates().values
    mapper = {}
    for i in range(len(lista)):
        mapper[lista[i]]=i
    df[col] = df[col].map(mapper)

# Aplicando para as colunas em que isso é necessário
for col in ['workclass','marital-status','occupation','relationship','race','gender','native-country','income']:
    apply_map(df_ad, col)
```

In [28]:

```
# Conferindo
df_ad.head(30)
```

Out[28]:

	age	workclass	fnlwgt	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
0	25	0	226802	7	0	0	0	0	0	0	0	40	0	0
1	38	0	89814	9	1	1	1	1	0	0	0	50	0	0
2	28	1	336951	12	1	2	1	1	0	0	0	40	0	1
3	44	0	160323	10	1	0	1	0	0	7688	0	40	0	1
4	18	2	103497	10	0	3	0	1	1	0	0	30	0	0
5	34	0	198693	6	0	4	2	1	0	0	0	30	0	0
6	29	2	227026	9	0	3	3	0	0	0	0	40	0	0
7	63	3	104626	15	1	5	1	1	0	3103	0	32	0	1
8	24	0	369667	10	0	4	3	1	1	0	0	40	0	0
9	55	0	104996	4	1	6	1	1	0	0	0	10	0	0
10	65	0	184454	9	1	0	1	1	0	6418	0	40	0	1
11	36	4	212465	13	1	7	1	1	0	0	0	40	0	0
12	26	0	82091	9	0	7	2	1	1	0	0	39	0	0
13	58	2	299831	9	1	3	1	1	0	0	0	35	0	0
14	48	0	279724	9	1	0	1	1	0	3103	0	48	0	1
15	43	0	346189	14	1	8	1	1	0	0	0	50	0	1
16	20	5	444554	10	0	4	0	1	0	0	0	25	0	0
17	43	0	128354	9	1	7	4	1	1	0	0	30	0	0
18	37	0	60548	9	2	0	3	1	1	0	0	20	0	0
20	34	0	107914	13	1	9	1	1	0	0	0	47	0	1
21	34	0	238588	10	0	4	0	0	1	0	0	35	0	0
22	72	2	132015	4	3	3	2	1	1	0	0	6	0	0
23	25	0	220931	13	0	5	2	1	0	0	0	43	1	0
24	25	0	205947	13	1	5	1	1	0	0	0	40	0	0
25	45	3	432824	9	1	6	1	1	0	7298	0	90	0	1
26	22	0	236427	9	0	7	0	1	0	0	0	20	0	0
27	23	0	134446	9	4	0	3	0	0	0	0	54	0	0
28	54	0	99516	9	1	6	1	1	0	0	0	35	0	0
29	32	3	109282	10	0	5	2	1	0	0	0	60	0	0

	age	workclass	fnlwgt	educational-num	marital-status	occupation	relationship	race	gender	capital-gain	capital-loss	hours-per-week	native-country	income
30	46	5	106444	10	1	8	1	0	0	7688	0	38	0	1

In [29]:
```
def simulacoes(df):
    RANDOM_STATE    = 7
    rec_test_prev   = 0
    best_perc_test  = 0
    best_perc_valid = 0
    best_estims     = 0
    best_max_feats  = 0
    best_profund    = 0
    best_min_smp_s  = 0
    best_min_smp_l  = 0
    test = 0

    MIN_SAMPLES_SPLIT = 2
    MIN_SAMPLES_LEAF  = 1
    for ESTIMADORES in [50,100]:
        for MAX_FEATURES in [7,9]:
            for PROFUND_ARVORE in [3,5,7]:
                print(f'#{test}: Estims: {ESTIMADORES}, Max.Feats: {MAX_FEATURES}, '+
                      f'Profund.: {PROFUND_ARVORE}, MSSPT: {MIN_SAMPLES_SPLIT}, MSLF: {MIN_SAMPLES_LEAF}')
                mod_ad = RandomForestClassifier(n_jobs=8,
                                                n_estimators=ESTIMADORES,
                                                max_depth=PROFUND_ARVORE,
                                                max_features=MAX_FEATURES,
                                                random_state=RANDOM_STATE,
                                                min_samples_split=MIN_SAMPLES_SPLIT,
                                                min_samples_leaf=MIN_SAMPLES_LEAF)
                for perc_test in [25,30]:
                    PERC_TEST = float(perc_test) / 100
                    for perc_valid in [10,15]:
                        PERC_VALID = float(perc_valid) / 100
                        test += 1
                        df_ad_test  = df.sample(frac=PERC_TEST, replace=False)
                        df_ad_train = df.drop(df_ad_test.index)

                        X = df_ad_train.drop(['income'], axis=1)
                        y = df_ad_train['income']

                        X_test = df_ad_test.drop(['income'], axis=1)
                        y_test = df_ad_test['income']

                        X_train, X_valid, y_train, y_valid = \
                        train_test_split(X, y, test_size=PERC_VALID, random_state=RANDOM_STATE,)

                        mod_ad.fit(X_train, y_train)

                        y_train_pred = mod_ad.predict(X_train)
                        y_valid_pred = mod_ad.predict(X_valid)
```

```python
                        acc_train  = accuracy_score(y_train, y_train_pred)
                        prec_train = precision_score(y_train, y_train_pred)
                        rec_train  = recall_score(y_train, y_train_pred)

                        if acc_train < 1 and prec_train < 1:

                            acc_valid  = accuracy_score(y_valid, y_valid_pred)
                            prec_valid = precision_score(y_valid, y_valid_pred)
                            rec_valid  = recall_score(y_valid, y_valid_pred)

                            if acc_valid < 1 and prec_valid < 1:

                                y_test_pred = mod_ad.predict(X_test)

                                acc_test  = accuracy_score(y_test, y_test_pred)
                                prec_test = precision_score(y_test, y_test_pred)
                                rec_test  = recall_score(y_test, y_test_pred)

                                if rec_test > rec_test_prev:
                                    rec_test_prev    = rec_test
                                    best_perc_test   = PERC_TEST
                                    best_perc_valid  = PERC_VALID
                                    best_estims      = ESTIMADORES
                                    best_max_feats   = MAX_FEATURES
                                    best_profund     = PROFUND_ARVORE
                                    best_min_smp_s   = MIN_SAMPLES_SPLIT
                                    best_min_smp_l   = MIN_SAMPLES_LEAF
                                    print('>>> #{}: Acc: {:6.2f} Prec: {:6.2f} Rec: {:6.2f} ' \
                                         .format(test,acc_test,prec_test,rec_test))
                                    print(f'RDST: {RANDOM_STATE}', end=' ')
                                    print(f'TTST: {PERC_TEST:.2f}', end=' ')
                                    print(f'TVLD: {PERC_VALID:.2f}', end=' ')
                                    print(f'ESTS: {ESTIMADORES}', end=' ')
                                    print(f'MXFT: {MAX_FEATURES}', end=' ')
                                    print(f'PROF: {PROFUND_ARVORE}', end=' ')
                                    print(f'MSPT: {MIN_SAMPLES_SPLIT}', end=' ')
                                    print(f'MSLF: {MIN_SAMPLES_LEAF}')
    print(f'\n{test} simulacoes executadas.')
    return RANDOM_STATE,best_perc_test,best_perc_valid,best_estims, \
           best_max_feats,best_profund,best_min_smp_s,best_min_smp_l
```

In [30]:

```python
# Constantes que influenciam a modelagem
RANDOM_STATE,PERC_TEST,PERC_VALID,ESTIMADORES,MAX_FEATURES,PROFUND_ARVORE,MIN_SAMPLES_SPLIT,MIN_SAMPLES_LEAF = \
simulacoes(df_ad)
```

```
#0: Estims: 50, Max.Feats: 7, Profund.: 3, MSSPT: 2, MSLF: 1
>>> #1: Acc:   0.84 Prec:   0.78 Rec:   0.47
RDST: 7 TTST: 0.25 TVLD: 0.10 ESTS: 50 MXFT: 7 PROF: 3 MSPT: 2 MSLF: 1
>>> #2: Acc:   0.84 Prec:   0.77 Rec:   0.48
RDST: 7 TTST: 0.25 TVLD: 0.15 ESTS: 50 MXFT: 7 PROF: 3 MSPT: 2 MSLF: 1
#4: Estims: 50, Max.Feats: 7, Profund.: 5, MSSPT: 2, MSLF: 1
>>> #5: Acc:   0.85 Prec:   0.76 Rec:   0.54
RDST: 7 TTST: 0.25 TVLD: 0.10 ESTS: 50 MXFT: 7 PROF: 5 MSPT: 2 MSLF: 1
#8: Estims: 50, Max.Feats: 7, Profund.: 7, MSSPT: 2, MSLF: 1
>>> #9: Acc:   0.86 Prec:   0.79 Rec:   0.54
RDST: 7 TTST: 0.25 TVLD: 0.10 ESTS: 50 MXFT: 7 PROF: 7 MSPT: 2 MSLF: 1
#12: Estims: 50, Max.Feats: 9, Profund.: 3, MSSPT: 2, MSLF: 1
#16: Estims: 50, Max.Feats: 9, Profund.: 5, MSSPT: 2, MSLF: 1
#20: Estims: 50, Max.Feats: 9, Profund.: 7, MSSPT: 2, MSLF: 1
>>> #21: Acc:   0.86 Prec:   0.78 Rec:   0.54
RDST: 7 TTST: 0.25 TVLD: 0.10 ESTS: 50 MXFT: 9 PROF: 7 MSPT: 2 MSLF: 1
#24: Estims: 100, Max.Feats: 7, Profund.: 3, MSSPT: 2, MSLF: 1
#28: Estims: 100, Max.Feats: 7, Profund.: 5, MSSPT: 2, MSLF: 1
#32: Estims: 100, Max.Feats: 7, Profund.: 7, MSSPT: 2, MSLF: 1
#36: Estims: 100, Max.Feats: 9, Profund.: 3, MSSPT: 2, MSLF: 1
#40: Estims: 100, Max.Feats: 9, Profund.: 5, MSSPT: 2, MSLF: 1
#44: Estims: 100, Max.Feats: 9, Profund.: 7, MSSPT: 2, MSLF: 1
>>> #46: Acc:   0.86 Prec:   0.80 Rec:   0.55
RDST: 7 TTST: 0.25 TVLD: 0.15 ESTS: 100 MXFT: 9 PROF: 7 MSPT: 2 MSLF: 1

48 simulacoes executadas.
```

In [31]:

```python
# Baseado nas simulações anteriores, os melhores valores foram:
RANDOM_STATE=7
PERC_TEST=0.25
PERC_VALID=0.15
ESTIMADORES=100
MAX_FEATURES=7
PROFUND_ARVORE=5
MIN_SAMPLES_SPLIT=2
MIN_SAMPLES_LEAF=1
```

In [32]:

```python
# Segmentando o Dataset em TREINO (treino e validação) e TESTE, garantindo que não existam duplicidades entre as bases
df_ad_test  = df_ad.sample(frac=PERC_TEST, replace=False)
df_ad_train = df_ad.drop(df_ad_test.index)
print('Base de treino: {} registros\nBase de teste.: {} registros'.format(len(df_ad_train), len(df_ad_test)))
```

```
Base de treino: 35989 registros
Base de teste.: 11996 registros
```

In [33]:

```python
# Split-Train-Test
X = df_ad_train.drop(['income'], axis=1)
y = df_ad_train['income']

X_test = df_ad_test.drop(['income'], axis=1)
y_test = df_ad_test['income']
```

In [34]:
```
# Obtendo os datasets de treino e de validação
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size=PERC_VALID, random_state=RANDOM_STATE)
print(X_train.shape, y_train.shape)
print(X_valid.shape, y_valid.shape)
```

```
(30590, 13) (30590,)
(5399, 13) (5399,)
```

In [35]:
```
print('Tamanho da base de treino...: {}'.format(len(X_train)))
print('Tamanho da base de validação: {}'.format(len(X_valid)))
print('Tamanho da base de teste....: {}'.format(len(X_test )))
```

```
Tamanho da base de treino...: 30590
Tamanho da base de validação: 5399
Tamanho da base de teste....: 11996
```

In [36]:
```
# Parametrização do RandomForest
mod_ad = RandomForestClassifier(n_jobs=6,
                                n_estimators=ESTIMADORES,
                                max_depth=PROFUND_ARVORE,
                                max_features=MAX_FEATURES,
                                random_state=RANDOM_STATE,
                                min_samples_split=MIN_SAMPLES_SPLIT,
                                min_samples_leaf=MIN_SAMPLES_LEAF)
mod_ad
```

Out[36]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=5, max_features=7, n_jobs=6, random_state=7)
```

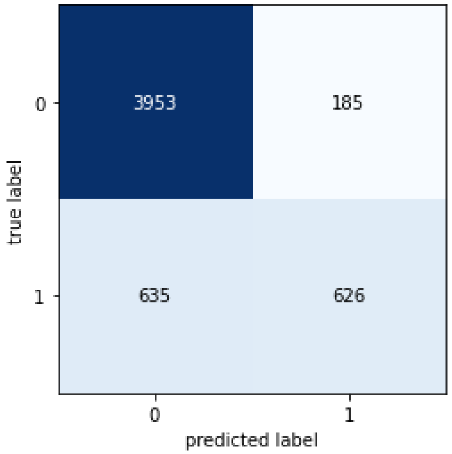
In [37]:
```
mod_ad.fit(X_train, y_train)
```

Out[37]:
```
▾ RandomForestClassifier
RandomForestClassifier(max_depth=5, max_features=7, n_jobs=6, random_state=7)
```

In [38]:
```
y_train_pred = mod_ad.predict(X_train)
y_valid_pred = mod_ad.predict(X_valid)
print(y_train_pred.shape, y_valid_pred.shape)
```

```
(30590,) (5399,)
```

In [39]:
```
cm = confusion_matrix(y_valid, y_valid_pred)
plot_confusion_matrix(conf_mat=cm)
plt.show()
```



In [40]:
```
acc_train  = accuracy_score(y_train, y_train_pred)
prec_train = precision_score(y_train, y_train_pred)
rec_train  = recall_score(y_train, y_train_pred)

acc_valid  = accuracy_score(y_valid, y_valid_pred)
prec_valid = precision_score(y_valid, y_valid_pred)
rec_valid  = recall_score(y_valid, y_valid_pred)
```

In [41]:
```
print(f'Treino:\nAcc: {acc_train:.2f}, Precision: {prec_train:.2f}, Recall: {rec_train:.2f}')
print(f'Validação:\nAcc: {acc_valid:.2f}, Precision: {prec_valid:.2f}, Recall: {rec_valid:.2f}')
```

```
Treino:
Acc: 0.85, Precision: 0.77, Recall: 0.52
Validação:
Acc: 0.85, Precision: 0.77, Recall: 0.50
```

In [42]:
```
y_test_pred = mod_ad.predict(X_test)

acc_test  = accuracy_score(y_test, y_test_pred)
prec_test = precision_score(y_test, y_test_pred)
rec_test  = recall_score(y_test, y_test_pred)

print(f'Teste:\nAcc: {acc_test:.2f}, Precision: {prec_test:.2f}, Recall: {rec_test:.2f}')
```

```
Teste:
Acc: 0.85, Precision: 0.78, Recall: 0.53
```

Árvore de Decisão Simples

In [43]:
```
arvore = DecisionTreeClassifier(max_depth=PROFUND_ARVORE,
                                min_samples_split=MIN_SAMPLES_SPLIT,
                                min_samples_leaf=MIN_SAMPLES_LEAF,
```

```python
                                random_state=RANDOM_STATE)
arvore
```

Out[43]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, random_state=7)
```

In [44]:

```python
arvore.fit(X_train, y_train)
```

Out[44]:

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=5, random_state=7)
```

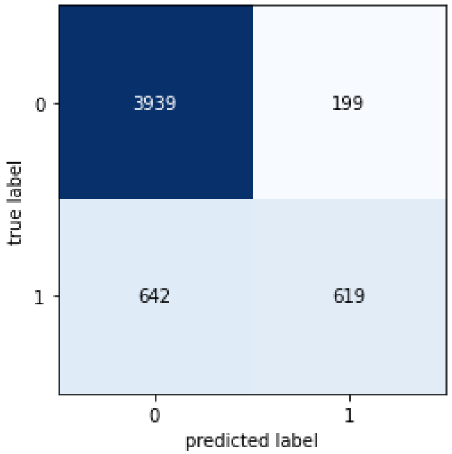
In [45]:

```python
y_train_pred = arvore.predict(X_train)
y_valid_pred = arvore.predict(X_valid)
print(y_train_pred.shape, y_valid_pred.shape)
```

```
(30590,) (5399,)
```

In [46]:

```python
cm = confusion_matrix(y_valid, y_valid_pred)
plot_confusion_matrix(conf_mat=cm)
plt.show()
```



In [47]:

```python
acc_train  = accuracy_score(y_train, y_train_pred)
prec_train = precision_score(y_train, y_train_pred)
rec_train  = recall_score(y_train, y_train_pred)
```

In [48]:

```python
acc_valid  = accuracy_score(y_valid, y_valid_pred)
prec_valid = precision_score(y_valid, y_valid_pred)
rec_valid  = recall_score(y_valid, y_valid_pred)
```

In [49]:

```python
print(f'Treino:\nAcc: {acc_train:.2f}, Precision: {prec_train:.2f}, Recall: {rec_train:.2f}')
```

```
print(f'Validação:\nAcc: {acc_valid:.2f}, Precision: {prec_valid:.2f}, Recall: {rec_valid:.2f}')
```

```
Treino:
Acc: 0.84, Precision: 0.76, Recall: 0.51
Validação:
Acc: 0.84, Precision: 0.76, Recall: 0.49
```

In [50]:
```
y_test_pred = arvore.predict(X_test)
```

In [51]:
```
acc_test  = accuracy_score(y_test, y_test_pred)
prec_test = precision_score(y_test, y_test_pred)
rec_test  = recall_score(y_test, y_test_pred)
```

In [52]:
```
print(f'Teste:\nAcc: {acc_test:.2f}, Precision: {prec_test:.2f}, Recall: {rec_test:.2f}')
```

```
Teste:
Acc: 0.85, Precision: 0.77, Recall: 0.52
```

Respostas:

1 - O tamanho da base utilizada para treino/validação e o tamanho da base utilizada para teste;

Tamanho da base de treino......: 30590

Tamanho da base de validação: 5399

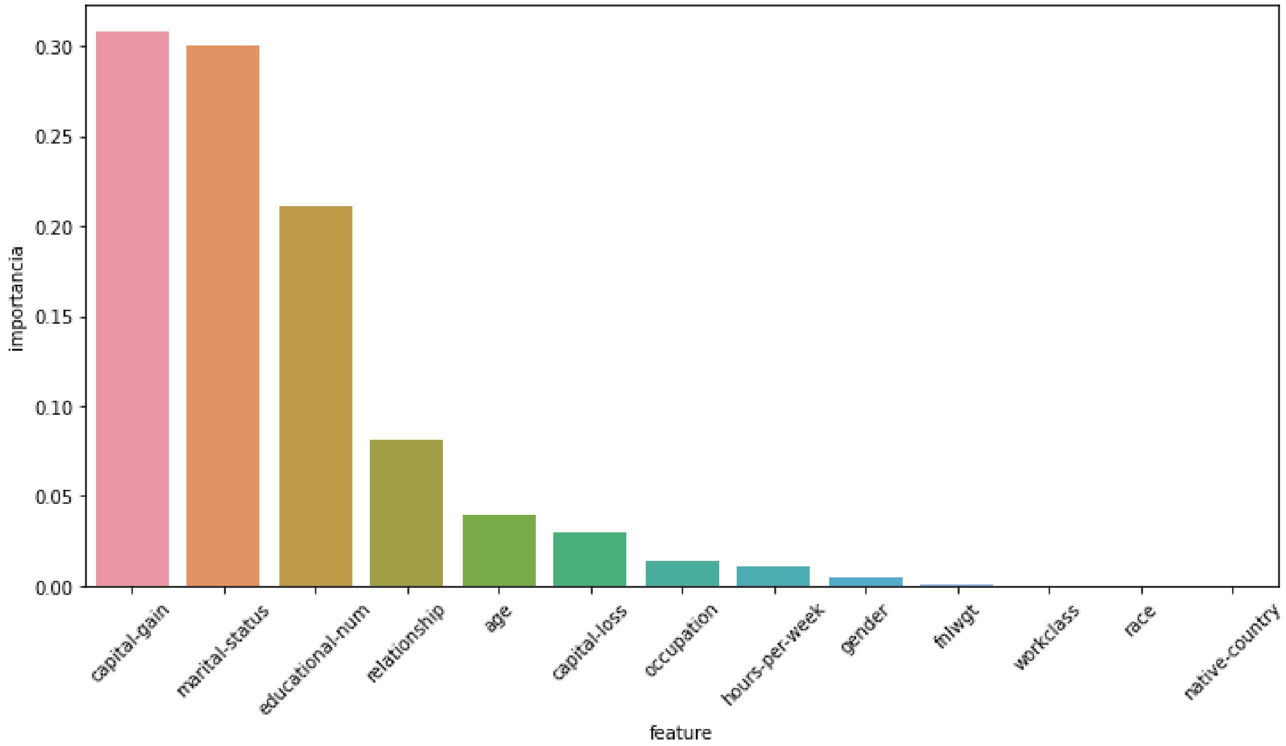
Tamanho da base de teste.......: 11996

2 - Um gráfico de barras com a importância de cada variável no precesso de predição;

In [53]:
```
df = pd.DataFrame(np.reshape([X_train.columns.values, mod_ad.feature_importances_], (2,13)).T)
df = df.sort_values(by=1, ascending=False)
df = df.rename(mapper={0:'feature',1:'importancia'}, axis=1)
```

In [54]:
```
fig = plt.figure(figsize=(12,6))
sns.barplot(x=df['feature'],y=df['importancia'])
plt.xticks(rotation=45)
fig.suptitle('Importância das Features', fontsize=20)
plt.show()
```

Importância das Features



3 - A acurácia, precisão e recall de treino, validação e teste;

<u>Treino:</u>

Acc: **0.99**, Precision: **0.98**, Recall: **0.98**

<u>Validação:</u>

Acc: **0.86**, Precision: **0.74**, Recall: **0.64**

<u>Teste:</u>

Acc: **0.85**, Precision: **0.73**, Recall: **0.62**

4 - Uma comparação com o desempenho de uma árvore de decisão simples.

<u>Random Forest:</u>

Acc: **0.85**, Precision: **0.73**, Recall: **0.62**

<u>Árvore de decisão simples:</u>

Acc: **0.82**, Precision: **0.63**, Recall: **0.62**

Conclusão: Random Forest permite alcançar níveis mais altos de acurácia, precisão e recall, desde que sejam exploradas diversas combinações dos hiperparâmetros pelo cientista de dados.

```
In [ ]:
```