Machine Learning e Data Science com Python de A à Z (Classificacão) - IA Expert Academy

Importação das bibliotecas básicas

```
!pip -q install plotly --upgrade
```

```
!pip -q install yellowbrick
```

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

Base de dados de crédito

- Fonte (adaptado): https://www.kaggle.com/laotse/credit-risk-dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.r
```

Exploração dos dados

```
base_credit = pd.read_csv('/content/credit_data.csv')
```

```
base_credit # defaulted
```

	clientid	income	age	loan	default
0	1	66155.925095	59.017015	8106.532131	0
1	2	34415.153966	48.117153	6564.745018	0
2	3	57317.170063	63.108049	8020.953296	0
3	4	42709.534201	45.751972	6103.642260	0
4	5	66952.688845	18.584336	8770.099235	1
...	...	...	...	...	...
1995	1996	59221.044874	48.518179	1926.729397	0

```
base_credit.head(10)
```

	clientid	income	age	loan	default
0	1	66155.925095	59.017015	8106.532131	0
1	2	34415.153966	48.117153	6564.745018	0
2	3	57317.170063	63.108049	8020.953296	0
3	4	42709.534201	45.751972	6103.642260	0
4	5	66952.688845	18.584336	8770.099235	1
5	6	24904.064140	57.471607	15.498598	0
6	7	48430.359613	26.809132	5722.581981	0
7	8	24500.141984	32.897548	2971.003310	1
8	9	40654.892537	55.496853	4755.825280	0
9	10	25075.872771	39.776378	1409.230371	0

```
base_credit.tail(8)
```

	clientid	income	age	loan	default
1992	1993	30803.806165	23.250084	623.024153	0
1993	1994	54421.410155	26.821928	3273.631823	0
1994	1995	24254.700791	37.751622	2225.284643	0
1995	1996	59221.044874	48.518179	1926.729397	0
1996	1997	69516.127573	23.162104	3503.176156	0
1997	1998	44311.449262	28.017167	5522.786693	1
1998	1999	43756.056605	63.971796	1622.722598	0
1999	2000	69436.579552	56.152617	7378.833599	0

```
base_credit.describe()
```

	clientid	income	age	loan	default
count	2000.000000	2000.000000	1997.000000	2000.000000	2000.000000
mean	1000.500000	45331.600018	40.807559	4444.369695	0.141500
std	577.494589	14326.327119	13.624469	3045.410024	0.348624
min	1.000000	20014.489470	-52.423280	1.377630	0.000000
25%	500.750000	32796.459717	28.990415	1939.708847	0.000000
50%	1000.500000	45789.117313	41.317159	3974.719419	0.000000
75%	1500.250000	57791.281668	52.587040	6432.410625	0.000000
max	2000.000000	69995.685578	63.971796	13766.051239	1.000000

```
base_credit[base_credit['income'] >= 69995.685578]
```

	clientid	income	age	loan	default
422	423	69995.685578	52.719673	2084.370861	0

```
base_credit[base_credit['loan'] <= 1.377630]
```

	clientid	income	age	loan	default
865	866	28072.604355	54.142548	1.37763	0

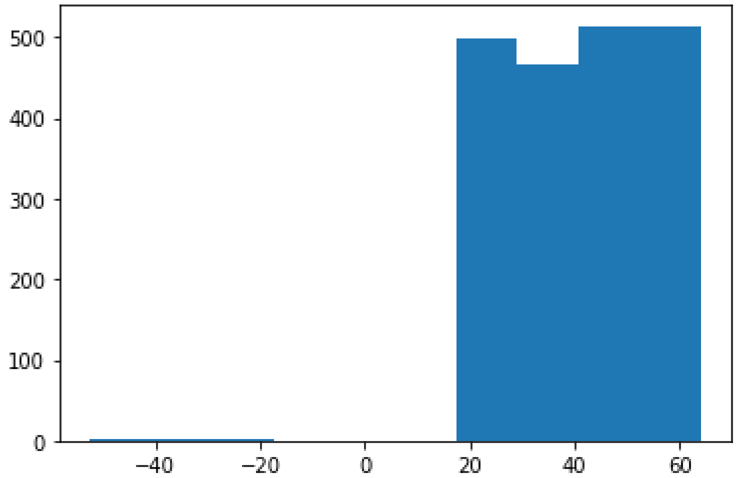
Visualização dos dados

```
np.unique(base_credit['default'], return_counts=True)
```

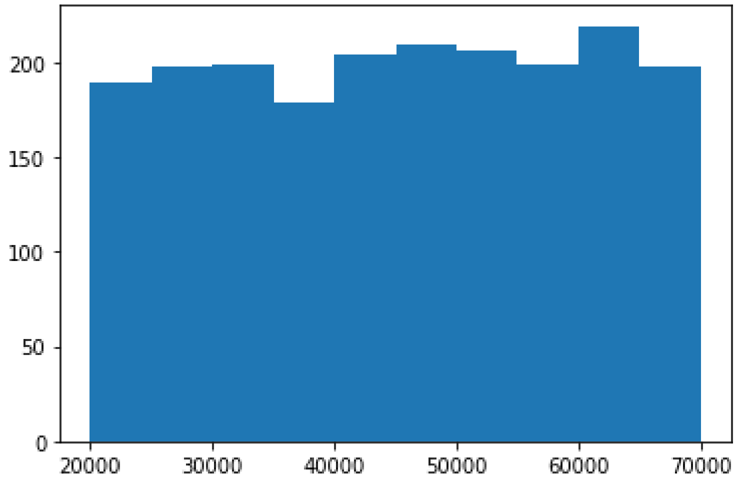
```
(array([0, 1]), array([1717,  283]))
```

```
sns.countplot(x = base_credit['default']);
```

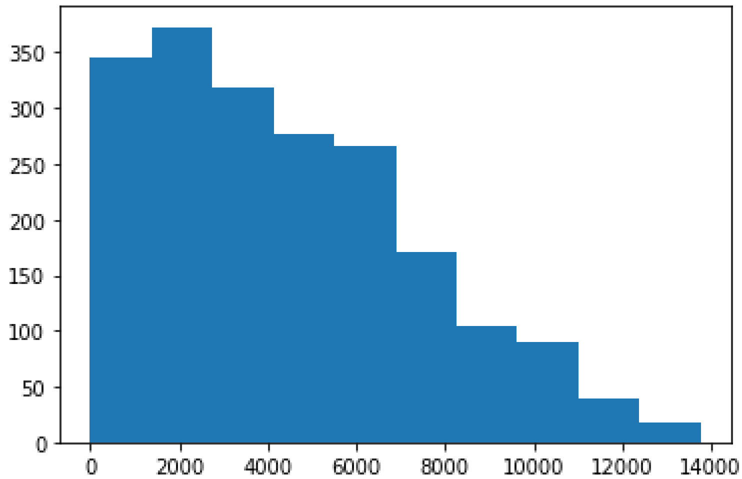
```
plt.hist(x = base_credit['age']);
```



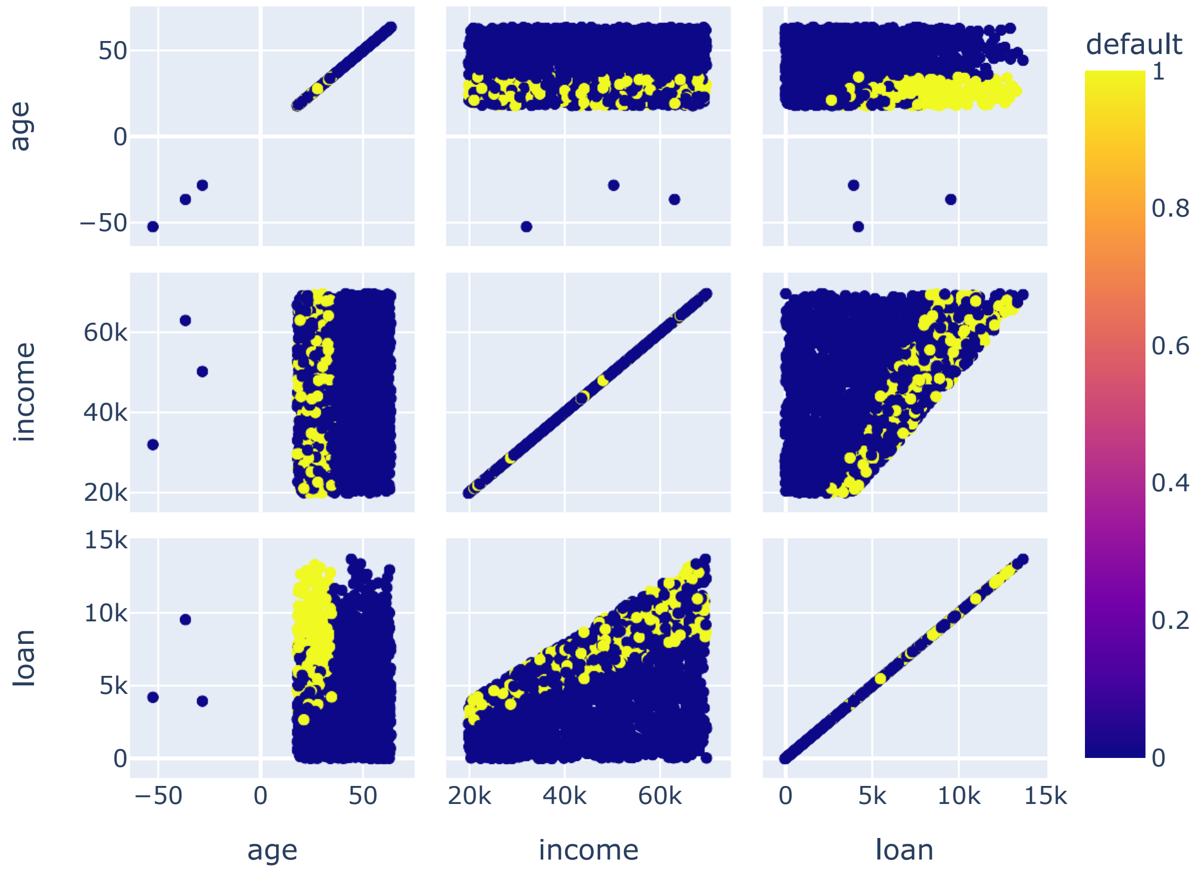

```
plt.hist(x = base_credit['income']);
```




```
plt.hist(x = base_credit['loan']);
```




```
grafico = px.scatter_matrix(base_credit, dimensions=['age', 'income', 'loan'], color = 'de
grafico.show()
```



Tratamento de valores inconsistentes

[ ] ↳ *15 células ocultas*

Tratamento de valores faltantes

[ ] ↳ *7 células ocultas*

Divisão entre previsores e classe

[ ] ↳ *7 células ocultas*

Escalonamento dos valores

[ ] ↳ *7 células ocultas*

Base de dados do censo

- Fonte: https://archive.ics.uci.edu/ml/datasets/adult

Exploração dos dados

[ ] ↳ *4 células ocultas*

Visualização dos dados

[ ] ↳ *10 células ocultas*

Divisão entre previsores e classe

[ ] ↳ *6 células ocultas*

Tratamento de atributos categóricos

[ ] ↳ *21 células ocultas*

Escalonamento dos valores

[ ] ↳ *2 células ocultas*

Divisão das bases em treinamento e teste

```
from sklearn.model_selection import train_test_split
```

Credit data

[ ] ↳ *4 células ocultas*

Census

[ ] ↳ *3 células ocultas*

Salvar as variáveis

[ ] ↳ *3 células ocultas*

Naïve Bayes

```
from sklearn.naive_bayes import GaussianNB
```

Base risco de crédito

[ ] ↳ *14 células ocultas*

Base credit data - 93.80%

[ ] ↳ *13 células ocultas*

Base census - 47.67%

[ ] ↳ *8 células ocultas*

Árvores de decisão

```
from sklearn.tree import DecisionTreeClassifier
```

Base risco de crédito

[ ] ↳ *8 células ocultas*

Base credit data - 98.20%

[ ] ↳ *12 células ocultas*

Base census - 81.04%

```
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pickle.load
```

```
X_census_treinamento.shape, y_census_treinamento.shape
```

```
    ((27676, 108), (27676,))
```

```
X_census_teste.shape, y_census_teste.shape
```

```
((4885, 108), (4885,))
```

```
arvore_census = DecisionTreeClassifier(criterion='entropy', random_state=0)
arvore_census.fit(X_census_treinamento, y_census_treinamento)
```

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

```
previsoes = arvore_census.predict(X_census_teste)
previsoes
```

```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' >50K'],
      dtype=object)
```

```
y_census_teste
```

```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' <=50K'],
      dtype=object)
```

```
accuracy_score(y_census_teste, previsoes)
```

```
0.8104401228249745
```

```
from yellowbrick.classifier import ConfusionMatrix
```

```
from yellowbrick.classifier import ConfusionMatrix
#cm = ConfusionMatrix(arvore_credit) corrigido 10/04/2021
cm = ConfusionMatrix(arvore_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

```
0.8104401228249745
```

```
print(classification_report(y_census_teste, previsoes))
```

```
              precision    recall  f1-score   support

        <=50K       0.88      0.87      0.87      3693
         >50K       0.61      0.61      0.61      1192

     accuracy                           0.81      4885
    macro avg       0.74      0.74      0.74      4885
 weighted avg       0.81      0.81      0.81      4885
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
```

Base credit data - 98.40%

[ ] ↳ *9 células ocultas*

Base census - 85.07%

[ ] ↳ *10 células ocultas*

Regras

[ ] ↳ *26 células ocultas*

Classificador base - Majority learner

[ ] ↳ *15 células ocultas*

Aprendizagem baseada em instâncias - knn

```
from sklearn.neighbors import KNeighborsClassifier
```

Base credit data - 98.60%

```
import pickle
with open('credit.pkl', 'rb') as f:
  X_credit_treinamento, y_credit_treinamento, X_credit_teste, y_credit_teste = pickle.load
```

```
X_credit_treinamento.shape, y_credit_treinamento.shape
```

```
((1500, 3), (1500,))
```

```
X_credit_teste.shape, y_credit_teste.shape
```

```
((500, 3), (500,))
```

```
knn_credit = KNeighborsClassifier(n_neighbors=5, metric='minkowski', p = 2)
knn_credit.fit(X_credit_treinamento, y_credit_treinamento)
```

```
KNeighborsClassifier()
```

```
previsoes = knn_credit.predict(X_credit_teste)
previsoes
```

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

```
y_credit_teste
```

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
```

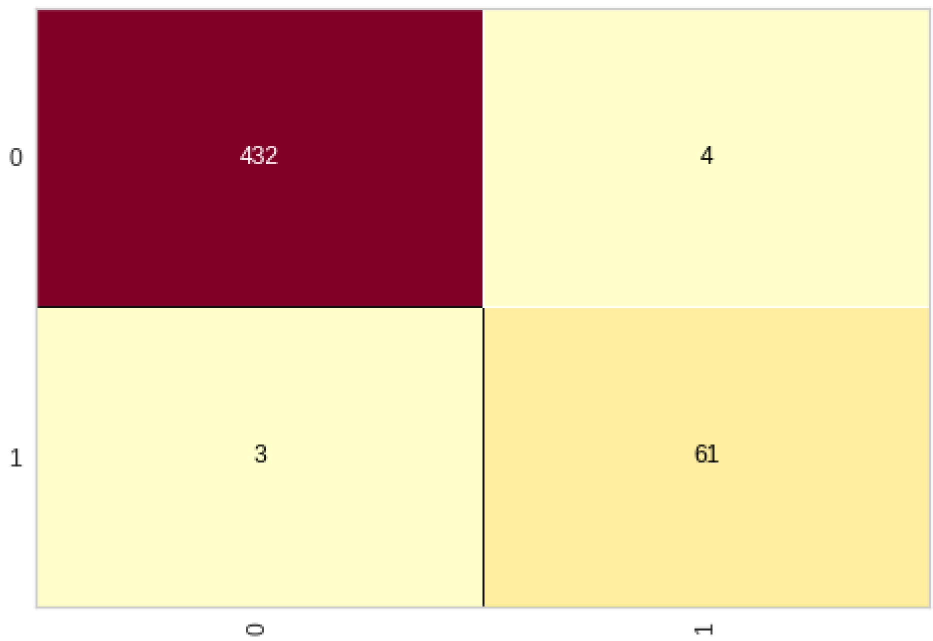
```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_credit_teste, previsoes) # padronização
```

```
0.986
```

```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(knn_credit)
cm.fit(X_credit_treinamento, y_credit_treinamento)
cm.score(X_credit_teste, y_credit_teste)
```

```
0.986
```




```
print(classification_report(y_credit_teste, previsoes))
```

```
              precision    recall  f1-score   support

           0       0.99      0.99      0.99       436
           1       0.94      0.95      0.95        64

    accuracy                           0.99       500
```

```
   macro avg       0.97      0.97      0.97       500
weighted avg       0.99      0.99      0.99       500
```

Base census - 82.90%

```
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pickle.load
```

```
X_census_treinamento.shape, y_census_treinamento.shape
```

```
((27676, 108), (27676,))
```

```
X_census_teste.shape, y_census_teste.shape
```

```
((4885, 108), (4885,))
```

```
knn_census = KNeighborsClassifier(n_neighbors=10)
knn_census.fit(X_census_treinamento, y_census_treinamento)
```

```
KNeighborsClassifier(n_neighbors=10)
```

```
previsoes = knn_census.predict(X_census_teste)
previsoes
```

```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' >50K'],
      dtype=object)
```

```
y_census_teste
```

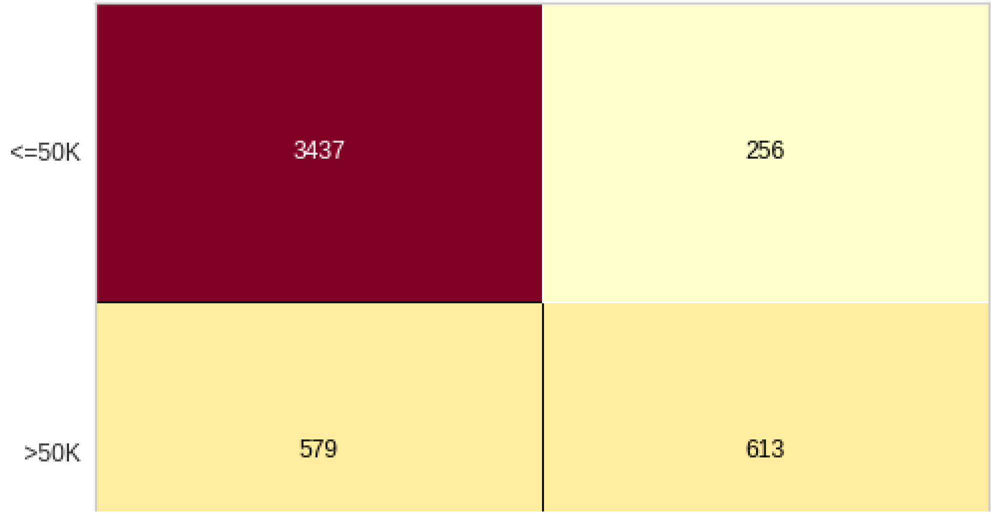
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' <=50K'],
      dtype=object)
```

```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_census_teste, previsoes)
```

```
0.8290685772773797
```

```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(knn_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

```
0.8290685772773797
```




```
print(classification_report(y_census_teste, previsoes))
```

```
              precision    recall  f1-score   support

        <=50K       0.86      0.93      0.89      3693
         >50K       0.71      0.51      0.59      1192

     accuracy                           0.83      4885
    macro avg       0.78      0.72      0.74      4885
 weighted avg       0.82      0.83      0.82      4885
```

Regressão logística

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

Base risco de crédito

```
import pickle
with open('risco_credito.pkl', 'rb') as f:
  X_risco_credito, y_risco_credito = pickle.load(f)
```

```
X_risco_credito
```

```
array([[2, 0, 1, 0],
       [1, 0, 1, 1],
       [1, 1, 1, 1],
       [1, 1, 1, 2],
       [1, 1, 1, 2],
       [1, 1, 0, 2],
       [2, 1, 1, 0],
       [2, 1, 0, 2],
       [0, 1, 1, 2],
       [0, 0, 0, 2],
```

```
       [0, 0, 1, 0],
       [0, 0, 1, 1],
       [0, 0, 1, 2],
       [2, 0, 1, 1]], dtype=object)
```

```
y_risco_credito # 2, 7, 11
```

```
array(['alto', 'alto', 'moderado', 'alto', 'baixo', 'baixo', 'alto',
       'moderado', 'baixo', 'baixo', 'alto', 'moderado', 'baixo', 'alto'],
      dtype=object)
```

```
X_risco_credito = np.delete(X_risco_credito, [2, 7, 11], axis = 0)
y_risco_credito = np.delete(y_risco_credito, [2, 7, 11], axis = 0)
```

```
X_risco_credito
```

```
array([[2, 0, 1, 0],
       [1, 0, 1, 1],
       [1, 1, 1, 2],
       [1, 1, 1, 2],
       [1, 1, 0, 2],
       [2, 1, 1, 0],
       [0, 1, 1, 2],
       [0, 0, 0, 2],
       [0, 0, 1, 0],
       [0, 0, 1, 2],
       [2, 0, 1, 1]], dtype=object)
```

```
y_risco_credito
```

```
array(['alto', 'alto', 'alto', 'baixo', 'baixo', 'alto', 'baixo', 'baixo',
       'alto', 'baixo', 'alto'], dtype=object)
```

```
logistic_risco_credito = LogisticRegression(random_state = 1)
logistic_risco_credito.fit(X_risco_credito, y_risco_credito)
```

```
LogisticRegression(random_state=1)
```

```
logistic_risco_credito.intercept_
```

```
array([-0.80828993])
```

```
logistic_risco_credito.coef_
```

```
array([[-0.76704533,  0.23906678, -0.47976059,  1.12186218]])
```

```
# história boa, dívida alta, garantias nenhuma, renda > 35
# história ruim, dívida alta, garantias adequada, renda < 15
previsoes1 = logistic_risco_credito.predict([[0,0,1,2], [2,0,0,0]])
previsoes1
```

```
array(['baixo', 'alto'], dtype=object)
```

Base credit data - 94.60%

```
import pickle
with open('credit.pkl', 'rb') as f:
  X_credit_treinamento, y_credit_treinamento, X_credit_teste, y_credit_teste = pickle.load
```

```
X_credit_treinamento.shape, y_credit_treinamento.shape
```

```
((1500, 3), (1500,))
```

```
X_credit_teste.shape, y_credit_teste.shape
```

```
((500, 3), (500,))
```

```
logistic_credit = LogisticRegression(random_state=1)
logistic_credit.fit(X_credit_treinamento, y_credit_treinamento)
```

```
LogisticRegression(random_state=1)
```

```
logistic_credit.intercept_
```

```
array([-6.02976095])
```

```
logistic_credit.coef_
```

```
array([[-2.54927091, -3.72279861,  3.93940349]])
```

```
previsoes = logistic_credit.predict(X_credit_teste)
previsoes
```

```
array([1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1])
```

```
y_credit_teste
```

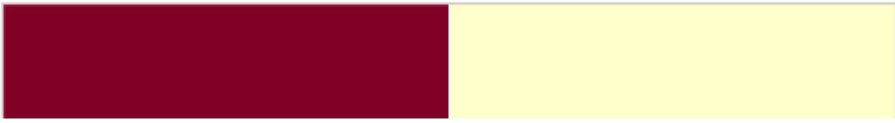
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_credit_teste, previsoes)
```

```
0.946
```

```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(logistic_credit)
cm.fit(X_credit_treinamento, y_credit_treinamento)
cm.score(X_credit_teste, y_credit_teste)
```

```
0.946
```



```
print(classification_report(y_credit_teste, previsoes))
```

```
              precision    recall  f1-score   support

           0       0.97      0.97      0.97       436
           1       0.79      0.78      0.79        64

    accuracy                           0.95       500
   macro avg       0.88      0.88      0.88       500
weighted avg       0.95      0.95      0.95       500
```


Base census - 84.95%

```
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pickle.load
```

```
X_census_treinamento.shape, y_census_treinamento.shape
```

```
((27676, 108), (27676,))
```

```
X_census_teste.shape, y_census_teste.shape
```

```
((4885, 108), (4885,))
```

```
logistic_census = LogisticRegression(random_state = 1)
logistic_census.fit(X_census_treinamento, y_census_treinamento)
```

```
LogisticRegression(random_state=1)
```

```
previsoes = logistic_census.predict(X_census_teste)
previsoes
```

```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' >50K'],
      dtype=object)
```

```
y_census_teste
```

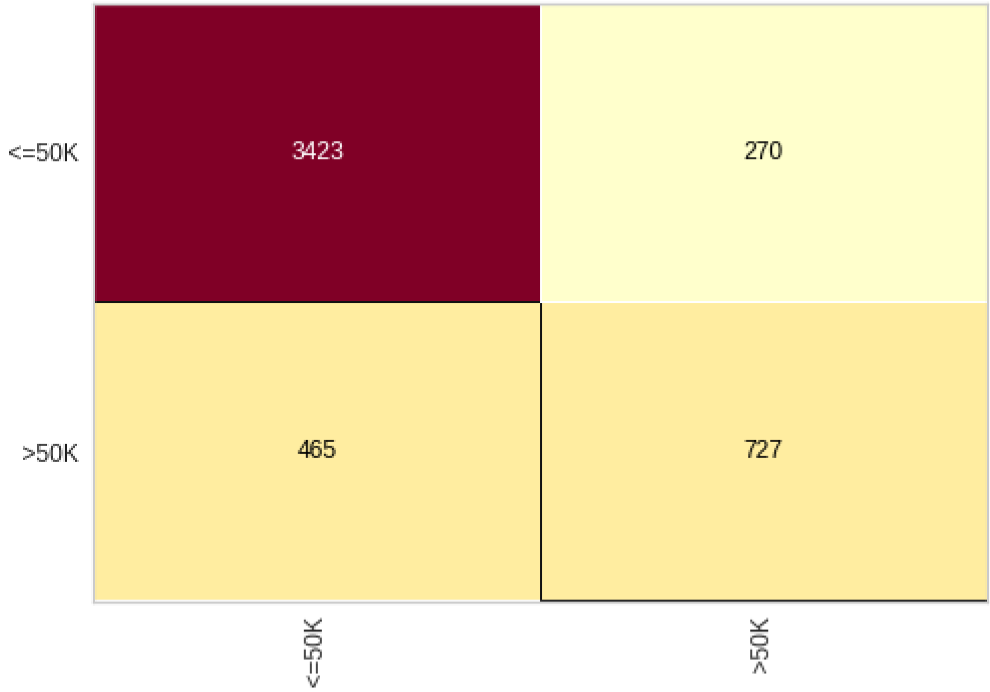
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' <=50K'],
      dtype=object)
```

```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_census_teste, previsoes)
```

```
0.849539406345957
```

```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(logistic_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

```
0.849539406345957
```




```
print(classification_report(y_census_teste, previsoes))
```

```
              precision    recall  f1-score   support

        <=50K       0.88      0.93      0.90      3693
         >50K       0.73      0.61      0.66      1192

     accuracy                           0.85      4885
    macro avg       0.80      0.77      0.78      4885
 weighted avg       0.84      0.85      0.84      4885
```

SVM

```
from sklearn.svm import SVC
```

Base credit data - 98.80%

```
import pickle
with open('credit.pkl', 'rb') as f:
  X_credit_treinamento, y_credit_treinamento, X_credit_teste, y_credit_teste = pickle.load
```

```
X_credit_treinamento.shape, y_credit_treinamento.shape
```

```
((1500, 3), (1500,))
```

```
X_credit_teste.shape, y_credit_teste.shape
```

```
((500, 3), (500,))
```

```
svm_credit = SVC(kernel='rbf', random_state=1, C = 2.0) # 2 -> 4
svm_credit.fit(X_credit_treinamento, y_credit_treinamento)
```

```
SVC(C=2.0, random_state=1)
```

```
previsoes = svm_credit.predict(X_credit_teste)
previsoes
```

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

```
y_credit_teste
```

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
```

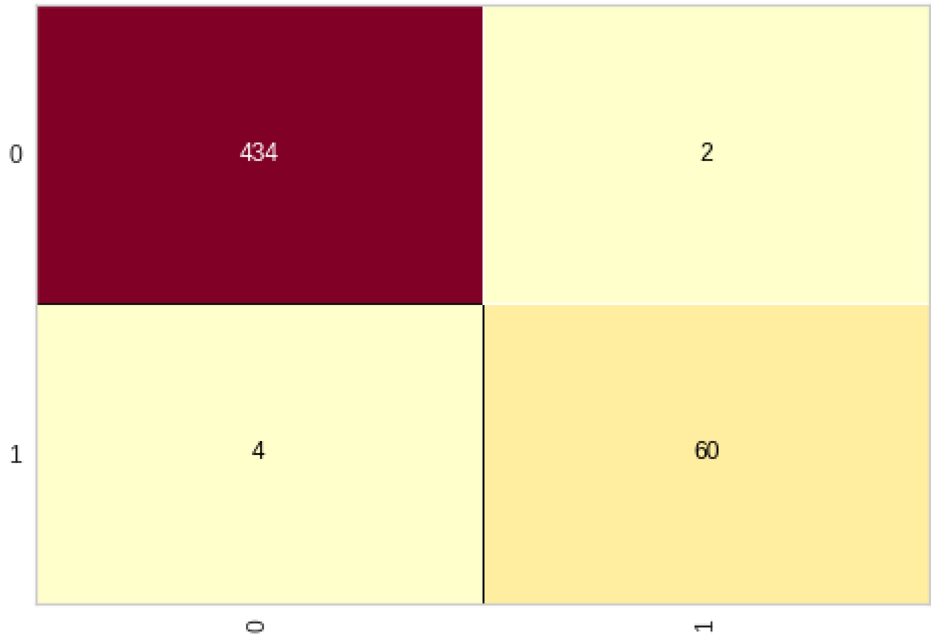
```
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_credit_teste, previsoes)
```

```
0.988
```

```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(svm_credit)
cm.fit(X_credit_treinamento, y_credit_treinamento)
cm.score(X_credit_teste, y_credit_teste)
```

```
0.988
```




```
print(classification_report(y_credit_teste, previsoes))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       436
           1       0.97      0.94      0.95        64

    accuracy                           0.99       500
   macro avg       0.98      0.97      0.97       500
weighted avg       0.99      0.99      0.99       500
```

Base census - 85.07%

```
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pickle.load
```

```
X_census_treinamento.shape, y_census_treinamento.shape
```

```
((27676, 108), (27676,))
```

```
X_census_teste.shape, y_census_teste.shape
```

```
((4885, 108), (4885,))
```

```
svm_census = SVC(kernel='linear', random_state=1)
svm_census.fit(X_census_treinamento, y_census_treinamento)
```

```
SVC(kernel='linear', random_state=1)
```

```
previsoes = svm_census.predict(X_census_teste)
previsoes
```

```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' >50K'],
      dtype=object)
```

```
y_census_teste
```

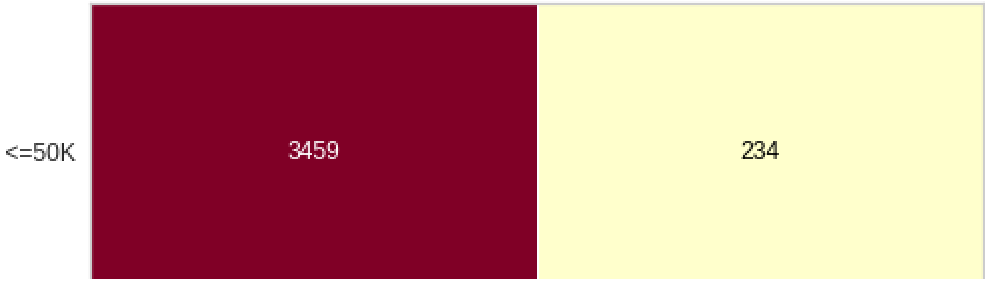
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' <=50K'],
      dtype=object)
```

```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_census_teste, previsoes)
```

```
0.8507676560900717
```

```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(svm_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

```
0.8507676560900717
```




```
print(classification_report(y_census_teste, previsoes))
```

```
              precision    recall  f1-score   support

        <=50K       0.87      0.94      0.90      3693
         >50K       0.75      0.58      0.66      1192

    accuracy                           0.85      4885
   macro avg       0.81      0.76      0.78      4885
weighted avg       0.84      0.85      0.84      4885
```

Redes neurais artificiais

```
from sklearn.neural_network import MLPClassifier
```

Base credit data - 99.80%

```
import pickle
with open('credit.pkl', 'rb') as f:
  X_credit_treinamento, y_credit_treinamento, X_credit_teste, y_credit_teste = pickle.load
```

```
X_credit_treinamento.shape, y_credit_treinamento.shape
```

```
((1500, 3), (1500,))
```

```
X_credit_teste.shape, y_credit_teste.shape
```

```
((500, 3), (500,))
```

```
(3 + 1) / 2
```

```
2.0
```

```
# 3 -> 100 -> 100 -> 1
# 3 -> 2 -> 2 -> 1
rede_neural_credit = MLPClassifier(max_iter=1500, verbose=True, tol=0.0000100,
                                   solver = 'adam', activation = 'relu',
```

```
                                    hidden_layer_sizes = (20,20))
rede_neural_credit.fit(X_credit_treinamento, y_credit_treinamento)
```

```
Iteration 1, loss = 0.65204293
Iteration 2, loss = 0.59098521
Iteration 3, loss = 0.53870648
Iteration 4, loss = 0.49443433
Iteration 5, loss = 0.45720984
Iteration 6, loss = 0.42609342
Iteration 7, loss = 0.39912533
Iteration 8, loss = 0.37570542
Iteration 9, loss = 0.35456489
Iteration 10, loss = 0.33525575
Iteration 11, loss = 0.31712993
Iteration 12, loss = 0.30071697
Iteration 13, loss = 0.28524999
Iteration 14, loss = 0.27094964
Iteration 15, loss = 0.25762619
Iteration 16, loss = 0.24545590
Iteration 17, loss = 0.23375509
Iteration 18, loss = 0.22324022
Iteration 19, loss = 0.21325300
Iteration 20, loss = 0.20410311
Iteration 21, loss = 0.19571024
Iteration 22, loss = 0.18761927
Iteration 23, loss = 0.18024214
Iteration 24, loss = 0.17323714
Iteration 25, loss = 0.16640819
Iteration 26, loss = 0.16013996
Iteration 27, loss = 0.15385369
Iteration 28, loss = 0.14817402
Iteration 29, loss = 0.14250644
Iteration 30, loss = 0.13699471
Iteration 31, loss = 0.13176464
Iteration 32, loss = 0.12647669
Iteration 33, loss = 0.12179600
Iteration 34, loss = 0.11709443
Iteration 35, loss = 0.11251869
Iteration 36, loss = 0.10836942
Iteration 37, loss = 0.10438001
Iteration 38, loss = 0.10052474
Iteration 39, loss = 0.09698033
Iteration 40, loss = 0.09337176
Iteration 41, loss = 0.08990146
Iteration 42, loss = 0.08674172
Iteration 43, loss = 0.08322457
Iteration 44, loss = 0.08019233
Iteration 45, loss = 0.07755223
Iteration 46, loss = 0.07487784
Iteration 47, loss = 0.07225380
Iteration 48, loss = 0.06997568
Iteration 49, loss = 0.06791483
Iteration 50, loss = 0.06582988
Iteration 51, loss = 0.06404354
Iteration 52, loss = 0.06226907
Iteration 53, loss = 0.06052696
Iteration 54, loss = 0.05891410
Iteration 55, loss = 0.05731052
Iteration 56, loss = 0.05591217
Iteration 57, loss = 0.05439610
```

```
previsoes = rede_neural_credit.predict(X_credit_teste)
previsoes
```

```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

```
y_credit_teste
```

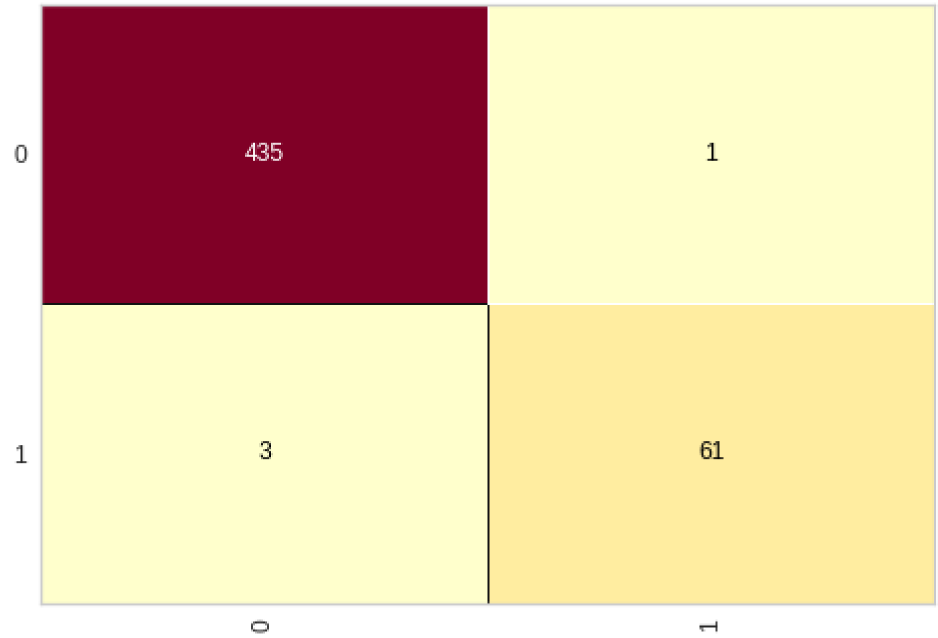
```
array([1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1])
```

```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_credit_teste, previsoes)
```

```
0.992
```

```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(rede_neural_credit)
cm.fit(X_credit_treinamento, y_credit_treinamento)
cm.score(X_credit_teste, y_credit_teste)
```

```
0.992
```




```
print(classification_report(y_credit_teste, previsoes))
```

```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       436
           1       0.98      0.95      0.97        64

    accuracy                           0.99       500
   macro avg       0.99      0.98      0.98       500
weighted avg       0.99      0.99      0.99       500
```

Base census - 81.53%

```
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pickle.load
```

```
X_census_treinamento.shape, y_census_treinamento.shape
```

```
((27676, 108), (27676,))
```

```
X_census_teste.shape, y_census_teste.shape
```

```
((4885, 108), (4885,))
```

```
(108 + 1) / 2
```

```
54.5
```

```
# 108 -> 55 -> 55 -> 1
rede_neural_census = MLPClassifier(verbose=True, max_iter = 1000, tol=0.000010,
                                  hidden_layer_sizes = (55,55))
rede_neural_census.fit(X_census_treinamento, y_census_treinamento)
```

```
Iteration 1, loss = 0.39432519
Iteration 2, loss = 0.32695311
Iteration 3, loss = 0.31597402
Iteration 4, loss = 0.30874944
Iteration 5, loss = 0.30404815
Iteration 6, loss = 0.30054436
Iteration 7, loss = 0.29701228
Iteration 8, loss = 0.29492061
Iteration 9, loss = 0.29194481
Iteration 10, loss = 0.28959595
Iteration 11, loss = 0.28713413
Iteration 12, loss = 0.28470848
Iteration 13, loss = 0.28278077
Iteration 14, loss = 0.28064719
Iteration 15, loss = 0.27910406
Iteration 16, loss = 0.27645999
Iteration 17, loss = 0.27514507
Iteration 18, loss = 0.27339473
Iteration 19, loss = 0.27104548
Iteration 20, loss = 0.26957841
Iteration 21, loss = 0.26849287
Iteration 22, loss = 0.26655556
Iteration 23, loss = 0.26473771
Iteration 24, loss = 0.26268347
Iteration 25, loss = 0.26073492
Iteration 26, loss = 0.25898137
Iteration 27, loss = 0.25764182
Iteration 28, loss = 0.25680764
Iteration 29, loss = 0.25495035
Iteration 30, loss = 0.25370149
Iteration 31, loss = 0.25286252
Iteration 32, loss = 0.25160981
Iteration 33, loss = 0.24957239
Iteration 34, loss = 0.24858877
Iteration 35, loss = 0.24803904
Iteration 36, loss = 0.24555263
Iteration 37, loss = 0.24383679
Iteration 38, loss = 0.24352651
Iteration 39, loss = 0.24128269
Iteration 40, loss = 0.24051108
Iteration 41, loss = 0.24015983
Iteration 42, loss = 0.23943530
Iteration 43, loss = 0.23773435
Iteration 44, loss = 0.23771926
Iteration 45, loss = 0.23522128
Iteration 46, loss = 0.23466492
Iteration 47, loss = 0.23323743
Iteration 48, loss = 0.23120017
```

```
Iteration 49, loss = 0.23049793
Iteration 50, loss = 0.23015591
Iteration 51, loss = 0.22891906
Iteration 52, loss = 0.22840673
Iteration 53, loss = 0.22737376
Iteration 54, loss = 0.22656310
Iteration 55, loss = 0.22630409
Iteration 56, loss = 0.22404703
Iteration 57, loss = 0.22424526
```

```
previsoes = rede_neural_census.predict(X_census_teste)
previsoes
```

```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' >50K'],
      dtype='<U6')
```

```
y_census_teste
```

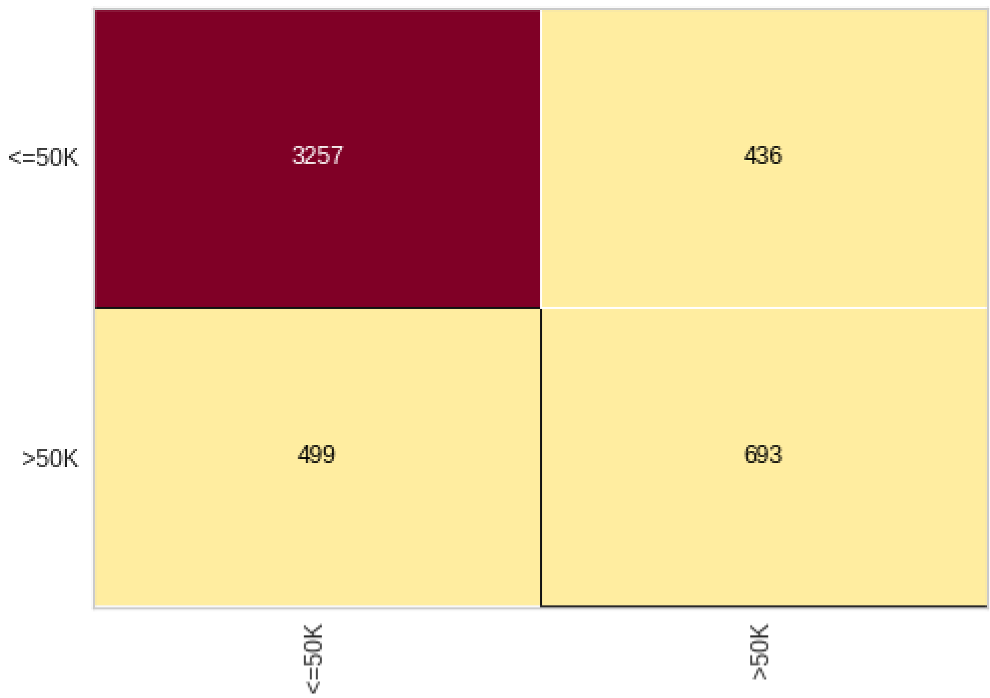
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' <=50K', ' <=50K'],
      dtype=object)
```

```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_census_teste, previsoes)
```

```
0.8085977482088025
```

```
from yellowbrick.classifier import ConfusionMatrix
cm = ConfusionMatrix(rede_neural_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

```
0.8085977482088025
```




```
print(classification_report(y_census_teste, previsoes))
```

```
              precision    recall  f1-score   support

       <=50K       0.87      0.88      0.87      3693
        >50K       0.61      0.58      0.60      1192

    accuracy                           0.81      4885
   macro avg       0.74      0.73      0.74      4885
weighted avg       0.81      0.81      0.81      4885
```

Avaliação dos algoritmos

- Naïve Bayes: 93.80
- Árvore de decisão: 98.20
- Random forest: 98.40
- Regras: 97.40
- Knn: 98.60
- Regressão logística: 94.60
- SVM: 98.80
- Redes neurais: 99.60

Tuning dos parâmetros com GridSearch

Preparação dos dados

```
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
```

```
import pickle
with open('credit.pkl', 'rb') as f:
  X_credit_treinamento, y_credit_treinamento, X_credit_teste, y_credit_teste = pickle.load
```

```
X_credit_treinamento.shape, y_credit_treinamento.shape
```

```
((1500, 3), (1500,))
```

```
X_credit_teste.shape, y_credit_teste.shape
```

```
((500, 3), (500,))
```

```
X_credit = np.concatenate((X_credit_treinamento, X_credit_teste), axis = 0)
X_credit.shape
```

```
(2000, 3)
```

```
X_credit
```

```
array([[-1.3754462 ,  0.50631087,  0.10980934],
       [ 1.45826409, -1.6489393 , -1.21501497],
       [-0.79356829,  0.22531191, -0.43370226],
       ...,
       [ 1.37445674, -1.05746281, -1.12564819],
       [-1.57087737, -0.63488173, -0.36981671],
       [-1.03572293, -0.93978122,  0.04244312]])
```

```
y_credit = np.concatenate((y_credit_treinamento, y_credit_teste), axis = 0)
y_credit.shape
```

```
(2000,)
```

```
y_credit
```

```
array([0, 0, 0, ..., 0, 1, 1])
```

Árvore de decisão

```
parametros = {'criterion': ['gini', 'entropy'],
              'splitter': ['best', 'random'],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 5, 10]}
```

```
grid_search = GridSearchCV(estimator=DecisionTreeClassifier(), param_grid=parametros)
grid_search.fit(X_credit, y_credit)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)
```

```
{'criterion': 'entropy', 'min_samples_leaf': 1, 'min_samples_split': 5, 'splitter':
0.983
```

Random forest

```
parametros = {'criterion': ['gini', 'entropy'],
              'n_estimators': [10, 40, 100, 150],
              'min_samples_split': [2, 5, 10],
              'min_samples_leaf': [1, 5, 10]}
```

```
grid_search = GridSearchCV(estimator=RandomForestClassifier(), param_grid=parametros)
grid_search.fit(X_credit, y_credit)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)
```

```
{'criterion': 'gini', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators':
0.986
```

Knn

```
parametros = {'n_neighbors': [3, 5, 10, 20],
              'p': [1, 2]}
```

```
grid_search = GridSearchCV(estimator=KNeighborsClassifier(), param_grid=parametros)
grid_search.fit(X_credit, y_credit)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)
```

```
{'n_neighbors': 20, 'p': 1}
0.9800000000000001
```

Regressão logística

```
parametros = {'tol': [0.0001, 0.00001, 0.000001],
              'C': [1.0, 1.5, 2.0],
              'solver': ['lbfgs', 'sag', 'saga']}
```

```
grid_search = GridSearchCV(estimator=LogisticRegression(), param_grid=parametros)
grid_search.fit(X_credit, y_credit)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)
```

```
{'C': 1.0, 'solver': 'lbfgs', 'tol': 0.0001}
0.9484999999999999
```

SVM

```
parametros = {'tol': [0.001, 0.0001, 0.00001],
              'C': [1.0, 1.5, 2.0],
```

```
             'kernel': ['rbf', 'linear', 'poly', 'sigmoid']}
```

```
grid_search = GridSearchCV(estimator=SVC(), param_grid=parametros)
grid_search.fit(X_credit, y_credit)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
print(melhores_parametros)
print(melhor_resultado)
```

```
{'C': 1.5, 'kernel': 'rbf', 'tol': 0.001}
0.9829999999999999
```

Redes neurais

```
parametros = {'activation': ['relu', 'logistic', 'tahn'],
              'solver': ['adam', 'sgd'],
              'batch_size': [10, 56]}
```

```
grid_search = GridSearchCV(estimator=MLPClassifier(), param_grid=parametros)
grid_search.fit(X_credit, y_credit)
melhores_parametros = grid_search.best_params_
melhor_resultado = grid_search.best_score_
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptro

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptro

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptro

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptro

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptro

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptro

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptro

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptro
```

```
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't
```

```
print(melhores_parametros)
print(melhor_resultado)
```

```
{'activation': 'relu', 'batch_size': 10, 'solver': 'adam'}
0.9970000000000001
```

Validação cruzada

```
from sklearn.model_selection import cross_val_score, KFold
```

```
10 * 30
```

```
300
```

```
resultados_arvore = []
resultados_random_forest = []
resultados_knn = []
resultados_logistica = []
resultados_svm = []
resultados_rede_neural = []

for i in range(30):
  print(i)
  kfold = KFold(n_splits=10, shuffle=True, random_state=i)
```

```
arvore = DecisionTreeClassifier(criterion='entropy', min_samples_leaf=1, min_samples_spl
scores = cross_val_score(arvore, X_credit, y_credit, cv = kfold)
#print(scores)
#print(scores.mean())
resultados_arvore.append(scores.mean())

random_forest = RandomForestClassifier(criterion = 'entropy', min_samples_leaf = 1, min_
scores = cross_val_score(random_forest, X_credit, y_credit, cv = kfold)
resultados_random_forest.append(scores.mean())

knn = KNeighborsClassifier()
scores = cross_val_score(knn, X_credit, y_credit, cv = kfold)
resultados_knn.append(scores.mean())

logistica = LogisticRegression(C = 1.0, solver = 'lbfgs', tol = 0.0001)
scores = cross_val_score(logistica, X_credit, y_credit, cv = kfold)
resultados_logistica.append(scores.mean())

svm = SVC(kernel = 'rbf', C = 2.0)
scores = cross_val_score(svm, X_credit, y_credit, cv = kfold)
resultados_svm.append(scores.mean())

rede_neural = MLPClassifier(activation = 'relu', batch_size = 56, solver = 'adam')
scores = cross_val_score(rede_neural, X_credit, y_credit, cv = kfold)
resultados_rede_neural.append(scores.mean())
```

```
0
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr
```

```
Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

1
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr

Stochastic Optimizer: Maximum iterations (200) reached and the optimization hasn't

/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptr
```

```
resultados = pd.DataFrame({'Arvore': resultados_arvore, 'Random forest': resultados_random
                           'KNN': resultados_knn, 'Logistica': resultados_logistica,
                           'SVM': resultados_svm, 'Rede neural': resultados_rede_neural})
resultados
```

```
resultados.describe()
```

```
resultados.var()
```

```
(resultados.std() / resultados.mean()) * 100
```

Teste de normalidade nos resultados

- Shapiro: https://en.wikipedia.org/wiki/Shapiro%E2%80%93Wilk_test

```
alpha = 0.05
```

```
from scipy.stats import shapiro
```

```
shapiro(resultados_arvore), shapiro(resultados_random_forest), shapiro(resultados_knn), sh
```

```
sns.displot(resultados_arvore, kind = 'kde');
```

```
sns.displot(resultados_random_forest, kind = 'kde');
```

```
sns.displot(resultados_knn, kind = 'kde');
```

```
sns.displot(resultados_logistica, kind = 'kde');
```

```
sns.displot(resultados_svm, kind = 'kde');
```

```
sns.displot(resultados_rede_neural, kind = 'kde');
```

Teste de hipótese com ANOVA e Tukey

```
from scipy.stats import f_oneway
```

```
_, p = f_oneway(resultados_arvore, resultados_random_forest, resultados_knn, resultados_lo
p
```

```
alpha = 0.05
if p <= alpha:
  print('Hipótese nula rejeitada. Dados são diferentes')
else:
  print('Hipótese alternativa rejeitada. Resultados são iguais')
```

```
resultados_algoritmos = {'accuracy': np.concatenate([resultados_arvore, resultados_random_
                         'algoritmo': ['arvore','arvore','arvore','arvore','arvore','arvor
                          'random_forest','random_forest','random_forest','random_forest',
                          'knn','knn','knn','knn','knn','knn','knn','knn','knn','knn','knn
                          'logistica','logistica','logistica','logistica','logistica','log
                          'svm','svm','svm','svm','svm','svm','svm','svm','svm','svm','svm
                          'rede_neural','rede_neural','rede_neural','rede_neural','rede_ne
```

```
resultados_df = pd.DataFrame(resultados_algoritmos)
resultados_df
```

```
from statsmodels.stats.multicomp import MultiComparison
```

```
compara_algoritmos = MultiComparison(resultados_df['accuracy'], resultados_df['algoritmo']
```

```
teste_estatistico = compara_algoritmos.tukeyhsd()
print(teste_estatistico)
```

```
resultados.mean()
```

```
teste_estatistico.plot_simultaneous();
```

Salvar um classificador já treinado

```
with open('credit.pkl', 'rb') as f:
  X_credit_treinamento, y_credit_treinamento, X_credit_teste, y_credit_teste = pickle.load
```

```
X_credit = np.concatenate((X_credit_treinamento, X_credit_teste), axis = 0)
y_credit = np.concatenate((y_credit_treinamento, y_credit_teste), axis = 0)
```

```
X_credit.shape, y_credit.shape
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
```

```
classificador_rede_neural = MLPClassifier(activation='relu', batch_size = 56, solver='adam
classificador_rede_neural.fit(X_credit, y_credit)
```

```
classificador_arvore = DecisionTreeClassifier(criterion='entropy', min_samples_leaf=1, min
classificador_arvore.fit(X_credit, y_credit)
```

```
classificador_svm = SVC(C = 2.0, kernel='rbf', probability=True)
classificador_svm.fit(X_credit, y_credit)
```

```
import pickle
pickle.dump(classificador_rede_neural, open('rede_neural_finalizado.sav', 'wb'))
pickle.dump(classificador_arvore, open('arvore_finalizado.sav', 'wb'))
pickle.dump(classificador_svm, open('svm_finalizado.sav', 'wb'))
```

Carregar um classificador já treinado

```
rede_neural = pickle.load(open('rede_neural_finalizado.sav', 'rb'))
arvore = pickle.load(open('arvore_finalizado.sav', 'rb'))
svm = pickle.load(open('svm_finalizado.sav', 'rb'))
```

```
novo_registro = X_credit[1999]
novo_registro
```

```
novo_registro.shape
```

```
novo_registro = novo_registro.reshape(1, -1)
novo_registro.shape
```

```
novo_registro
```

```
rede_neural.predict(novo_registro)
```

```
arvore.predict(novo_registro)
```

```
svm.predict(novo_registro)
```

Combinação de classificadores

```
novo_registro = X_credit[1999]
novo_registro = novo_registro.reshape(1, -1)
novo_registro, novo_registro.shape
```

```
resposta_rede_neural = rede_neural.predict(novo_registro)
resposta_arvore = arvore.predict(novo_registro)
resposta_svm = svm.predict(novo_registro)
```

```
resposta_rede_neural[0], resposta_arvore[0], resposta_svm[0]
```

```
paga = 0
nao_paga = 0

if resposta_rede_neural[0] == 1:
  nao_paga += 1
else:
  paga += 1

if resposta_arvore[0] == 1:
  nao_paga += 1
else:
  paga += 1

if resposta_svm[0] == 1:
  nao_paga += 1
else:
  paga += 1

if paga > nao_paga:
  print('Cliente pagará o empréstimo')
elif paga == nao_paga:
  print('Empate')
```

```
else:
  print('Cliente não pagará o empréstimo')
```

Rejeição de classificadores

```
novo_registro = X_credit[1999]
novo_registro = novo_registro.reshape(1, -1)
novo_registro, novo_registro.shape
```

```
resposta_rede_neural = rede_neural.predict(novo_registro)
resposta_arvore = arvore.predict(novo_registro)
resposta_svm = svm.predict(novo_registro)
```

```
resposta_rede_neural[0], resposta_arvore[0], resposta_svm[0]
```

```
probabilidade_rede_neural = rede_neural.predict_proba(novo_registro)
probabilidade_rede_neural
```

```
confianca_rede_neural = probabilidade_rede_neural.max()
confianca_rede_neural
```

```
probabilidade_arvore = arvore.predict_proba(novo_registro)
confianca_arvore = probabilidade_arvore.max()
confianca_arvore
```

```
probabilidade_svm = svm.predict_proba(novo_registro)
confianca_svm = probabilidade_svm.max()
confianca_svm
```

```
paga = 0
nao_paga = 0
confianca_minima = 0.999999
algoritmos = 0

if confianca_rede_neural >= confianca_minima:
  algoritmos += 1
  if resposta_rede_neural[0] == 1:
    nao_paga += 1
  else:
    paga += 1

if confianca_arvore >= confianca_minima:
  algoritmos += 1
  if resposta_arvore[0] == 1:
    nao_paga += 1
  else:
    paga += 1
```

```
  if confianca_svm >= confianca_minima:
    algoritmos += 1
    if resposta_svm[0] == 1:
      nao_paga += 1
    else:
      paga += 1

  if paga > nao_paga:
    print('Cliente pagará o empréstimo, baseado em {} algoritmos'.format(algoritmos))
  elif paga == nao_paga:
    print('Empate, baseado em {} algoritmos'.format(algoritmos))
  else:
    print('Cliente não pagará o empréstimo, baseado em {} algoritmos'.format(algoritmos))
```

Redução de dimensionalidade

Preparacão da base de dados

```
base_census = pd.read_csv('/content/census.csv')
base_census
```

```
X_census = base_census.iloc[:, 0:14].values
X_census
```

```
y_census = base_census.iloc[:, 14].values
y_census
```

```
from sklearn.preprocessing import LabelEncoder
label_encoder_workclass = LabelEncoder()
label_encoder_education = LabelEncoder()
label_encoder_marital = LabelEncoder()
label_encoder_occupation = LabelEncoder()
label_encoder_relationship = LabelEncoder()
label_encoder_race = LabelEncoder()
label_encoder_sex = LabelEncoder()
label_encoder_country = LabelEncoder()
```

```
X_census[:,1] = label_encoder_workclass.fit_transform(X_census[:,1])
X_census[:,3] = label_encoder_education.fit_transform(X_census[:,3])
X_census[:,5] = label_encoder_marital.fit_transform(X_census[:,5])
X_census[:,6] = label_encoder_occupation.fit_transform(X_census[:,6])
X_census[:,7] = label_encoder_relationship.fit_transform(X_census[:,7])
X_census[:,8] = label_encoder_race.fit_transform(X_census[:,8])
X_census[:,9] = label_encoder_sex.fit_transform(X_census[:,9])
X_census[:,13] = label_encoder_country.fit_transform(X_census[:,13])
```

```
X_census[0]
```

```
from sklearn.preprocessing import StandardScaler
scaler_census = StandardScaler()
X_census = scaler_census.fit_transform(X_census)
```

```
X_census
```

```
from sklearn.model_selection import train_test_split
X_census_treinamento, X_census_teste, y_census_treinamento, y_census_teste = train_test_sp
```

```
X_census_treinamento.shape, X_census_teste.shape
```

PCA (Principal component analysis)

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=8)
```

```
X_census_treinamento_pca = pca.fit_transform(X_census_treinamento)
X_census_testes_pca = pca.transform(X_census_teste)
```

```
X_census_treinamento_pca.shape, X_census_testes_pca.shape
```

```
X_census_treinamento
```

```
pca.explained_variance_ratio_
```

```
pca.explained_variance_ratio_.sum()
```

```
from sklearn.ensemble import RandomForestClassifier
```

```
random_forest_census_pca = RandomForestClassifier(n_estimators=40, random_state=0, criteri
random_forest_census_pca.fit(X_census_treinamento_pca, y_census_treinamento)
```

```
previsoes = random_forest_census_pca.predict(X_census_testes_pca)
previsoes
```

```
y_census_teste
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_census_teste, previsoes)
```

Kernel PCA

```
from sklearn.decomposition import KernelPCA
```

```
kpca = KernelPCA(n_components=8, kernel='rbf')
X_census_treinamento_kpca = kpca.fit_transform(X_census_treinamento)
X_census_teste_kpca = kpca.transform(X_census_teste)
```

```
X_census_treinamento_kpca.shape, X_census_teste_kpca.shape
```

```
X_census_treinamento_kpca
```

```
from sklearn.ensemble import RandomForestClassifier
random_forest_census_kpca = RandomForestClassifier(n_estimators = 40, criterion = 'entropy
random_forest_census_kpca.fit(X_census_treinamento_kpca, y_census_treinamento)
```

```
previsoes = random_forest_census_kpca.predict(X_census_teste_kpca)
previsoes
```

```
y_census_teste
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_census_teste, previsoes)
```

LDA (Linear discriminant analysis)

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components = 8)
```

```
X_census_treinamento_lda = lda.fit_transform(X_census_treinamento, y_census_treinamento)
X_census_teste_lda = lda.transform(X_census_teste)
```

```
X_census_treinamento_lda.shape, X_census_teste_lda.shape
```

```
X_census_treinamento_lda
```

```
from sklearn.ensemble import RandomForestClassifier
random_forest_census_lda = RandomForestClassifier(n_estimators = 40, criterion = 'entropy'
random_forest_census_lda.fit(X_census_treinamento_lda, y_census_treinamento)
```

```
previsoes = random_forest_census_lda.predict(X_census_teste_lda)
```

```
previsoes
```

```
y_census_teste
```

```
from sklearn.metrics import accuracy_score
accuracy_score(y_census_teste, previsoes)
```

Detecção de outliers

Boxplot

```
base_credit = pd.read_csv('credit_data.csv')
base_credit
```

```
base_credit.isnull().sum()
```

```
base_credit.dropna(inplace=True)
```

```
base_credit.isnull().sum()
```

```
1997 / 2
```

```
# Outliers idade
grafico = px.box(base_credit, y = 'age')
grafico.show()
```

```
outliers_age = base_credit[base_credit['age'] < 0]
outliers_age
```

```
# Outliers loan
grafico = px.box(base_credit, y='loan')
grafico.show()
```

```
outliers_loan = base_credit[base_credit['loan'] > 13300]
outliers_loan
```

Gráfico de dispersão

```
# Income x age
grafico = px.scatter(x = base_credit['income'], y = base_credit['age'])
grafico.show()
```

```
# Income x loan
grafico = px.scatter(x = base_credit['income'], y = base_credit['loan'])
grafico.show()
```

```
# Age x loan
grafico = px.scatter(x = base_credit['age'], y = base_credit['loan'])
grafico.show()
```

```
base_census = pd.read_csv('census.csv')
base_census
```

```
# Age x final weight
grafico = px.scatter(x = base_census['age'], y = base_census['final-weight'])
grafico.show()
```

Biblioteca PyOD

- Documentação: https://pyod.readthedocs.io/en/latest/#

```
!pip install pyod
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/p
Collecting pyod
  Downloading pyod-1.0.1.tar.gz (120 kB)
     |████████████████████████████████| 120 kB 14.3 MB/s
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages (fron
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: numpy>=1.19 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: numba>=0.51 in /usr/local/lib/python3.7/dist-packages
Collecting scipy>=1.5.1
  Downloading scipy-1.7.3-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (
     |████████████████████████████████| 38.1 MB 372 kB/s
Requirement already satisfied: scikit_learn>=0.20.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from py
Requirement already satisfied: statsmodels in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: llvmlite<0.35,>=0.34.0.dev0 in /usr/local/lib/python3
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-pac
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
Building wheels for collected packages: pyod
  Building wheel for pyod (setup.py) ... done
  Created wheel for pyod: filename=pyod-1.0.1-py3-none-any.whl size=147473 sha256=b6f
  Stored in directory: /root/.cache/pip/wheels/ea/c4/29/67ad87835b209f72e4706369c6837
Successfully built pyod
```

```
Installing collected packages: scipy, pyod
  Attempting uninstall: scipy
    Found existing installation: scipy 1.4.1
    Uninstalling scipy-1.4.1:
      Successfully uninstalled scipy-1.4.1
ERROR: pip's dependency resolver does not currently take into account all the package
albumentations 0.1.12 requires imgaug<0.2.7,>=0.2.5, but you have imgaug 0.2.9 which
Successfully installed pyod-1.0.1 scipy-1.7.3
```

```
from pyod.models.knn import KNN
```

```
base_credit.head(1)
```

```
detector = KNN()
detector.fit(base_credit.iloc[:,1:4])
```

```
previsoes = detector.labels_
previsoes
```

```
np.unique(previsoes, return_counts=True)
```

```
confianca_previsoes = detector.decision_scores_
confianca_previsoes
```

```
outliers = []
for i in range(len(previsoes)):
  #print(i)
  if previsoes[i] == 1:
    outliers.append(i)
```

```
print(outliers)
```

```
lista_outliers = base_credit.iloc[outliers,:]
lista_outliers
```