2. Classificação

A classificação é um método de aprendizado de máquina supervisionado em que o modelo tenta prever o rótulo correto de um determinado dado de entrada.

Encoding

Alguns algoritmos mesmo que de classificação não conseguem compreender rótulos, apenas números, então é necessário transformar os dados categóricos utilizando técnicas codificação.

Label Encoding

Cada categoria recebe um número, normalmente em ordem alfabética. Normalmente utilizado para dados categóricos ordinários.

Color	Size	Price
Red	Small	10
Green	Medium	20
Blue	Large	30
Red	Large	25
Green	Small	15

Category	Encoded Value
Color: Red	0
Color: Green	1
Color: Blue	2

Category	Encoded Value
Size: Small	0
Size: Medium	1
Size: Large	2

Color_Encoded	Size_Encoded	Price
0	0	10
1	1	20
2	2	30
0	2	25
1	0	15

Dessa forma já é possível trabalhar com diversos algoritmos, no entanto alguns podem correlacionar os dados como uma ordem de grandeza. Isso acaba conferindo uma "importância" maior para categorias com um números maior. Nesses casos utilizamos outro encolder.

One-hot Encoding

Cada categoria é transformada em outro atributo: dummy variable, um valor binário que informa a ocorrência.

Color	Size	Price
Red	Small	10
Green	Medium	20
Blue	Large	30
Red	Large	25
Green	Small	15

Color_Red	Color_Green	Color_Blue	Size_Small	Size_Medium	Size_Large	Price
1	0	0	1	0	0	10
0	1	0	0	1	0	20
0	0	1	0	0	1	30
1	0	0	0	0	1	25
0	1	0	1	0	0	15

Temos que tomar cuidado com alguns fatores:

- Muitas colunas podem gerar um espaço de características de alta dimensão, que pode causar super ajuste e ter um custo computacional muito alto.
- Maldição da Dimensionaldiade: Dados esparços, muitas colunas com valor zero, tornando difícil encontrar valores nos dados
- Dummy Variable Trap: valores de colunas binárias podem ser previstos a partir dos valores de outras colunas.

Solução: Excluir um dos atributos ou combinar colunas binárias

Color_Red	Color_Green	Color_Blue	Size_Small	Size_Medium	Size_Large	Price
1	0	0	1	0	0	10
0	1	0	0	1	0	20
0	0	1	0	0	1	30
1	0	0	0	0	1	25
0	1	0	1	0	0	15

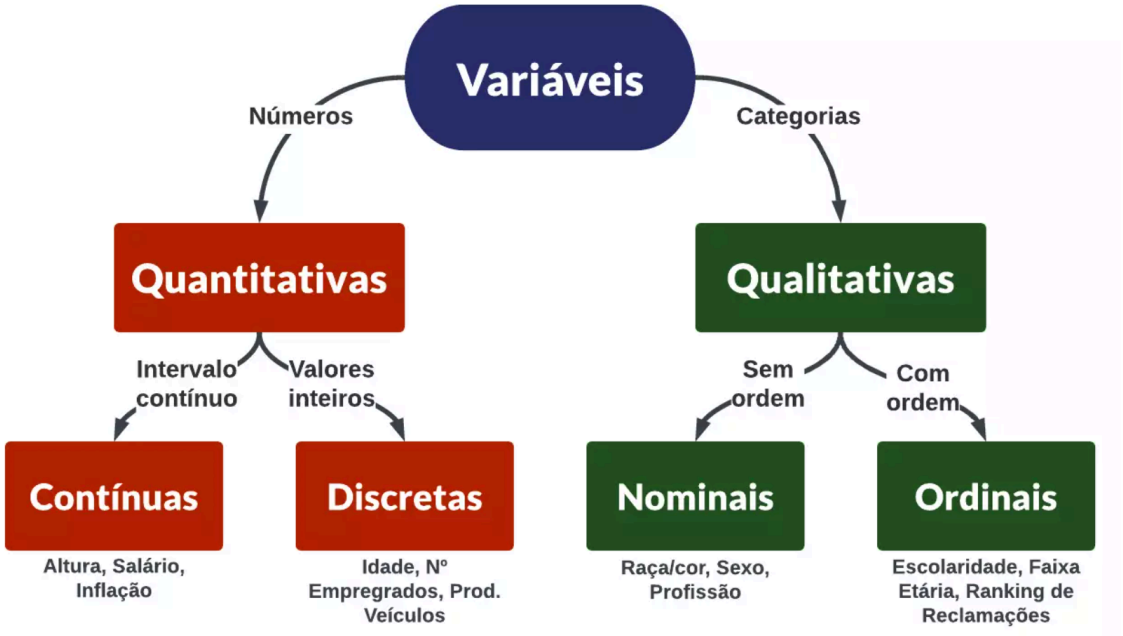
Qual utilizar?

Label encoding	One-hot encoding
Há ordem (progr. Junior, Pleno, Sênior)	Não há ordem
Grande Número de categorias, não da pra usar One-hot encoding	Número de categorias é pequeno

Vamos utilizar uma base de dados do censo para testarmos.

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.express as px
```

Agora diferente dos casos de regressão, na classificação temos diferentes tipos de variáveis e é importante distingui-las no dataset antes de começarmos a trabalhar.



In [3]:
```
base_census = pd.read_csv('census.csv')
base_census
```

Out[3]:

	age	workclass	final-weight	education	education-num	marital-status	occupation	relationship
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife
...	...	...	...	...	...	...	...	...
32556	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife
32557	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband
32558	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried
32559	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child
32560	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife

32561 rows × 15 columns

In [4]:
```
base_census.describe()
```

Out[4]:

	age	final-weight	education-num	capital-gain	capital-loos	hour-per-week
count	32561.000000	3.256100e+04	32561.000000	32561.000000	32561.000000	32561.000000
mean	38.581647	1.897784e+05	10.080679	1077.648844	87.303830	40.437456
std	13.640433	1.055500e+05	2.572720	7385.292085	402.960219	12.347429
min	17.000000	1.228500e+04	1.000000	0.000000	0.000000	1.000000
25%	28.000000	1.178270e+05	9.000000	0.000000	0.000000	40.000000
50%	37.000000	1.783560e+05	10.000000	0.000000	0.000000	40.000000
75%	48.000000	2.370510e+05	12.000000	0.000000	0.000000	45.000000
max	90.000000	1.484705e+06	16.000000	99999.000000	4356.000000	99.000000

In [ ]:
```
base_census.isnull().sum()
```

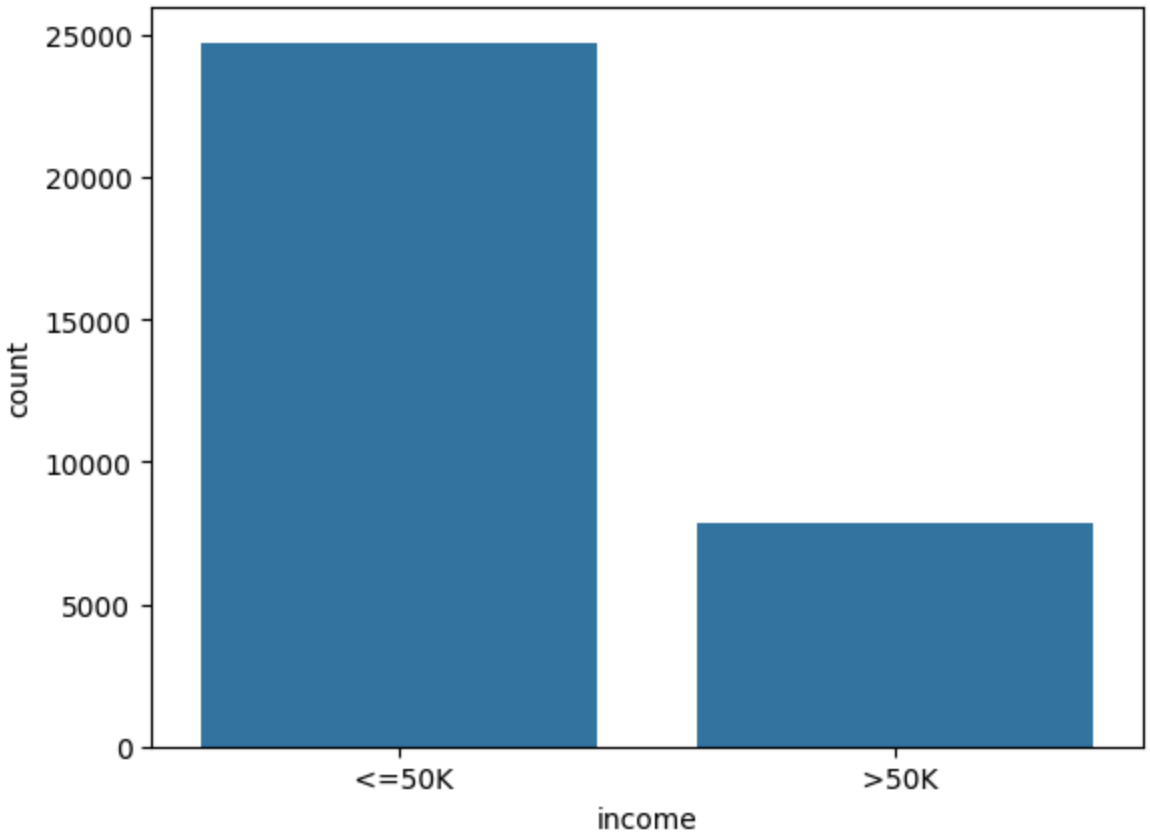
Income é o nosso target, se refere a renda anual dos induvíduos do dataset, que se dividem em duas classes, as que ganham mais de 50 mil no ano e as que ganham menos.

In [5]:
```
np.unique(base_census['income'], return_counts=True)
```

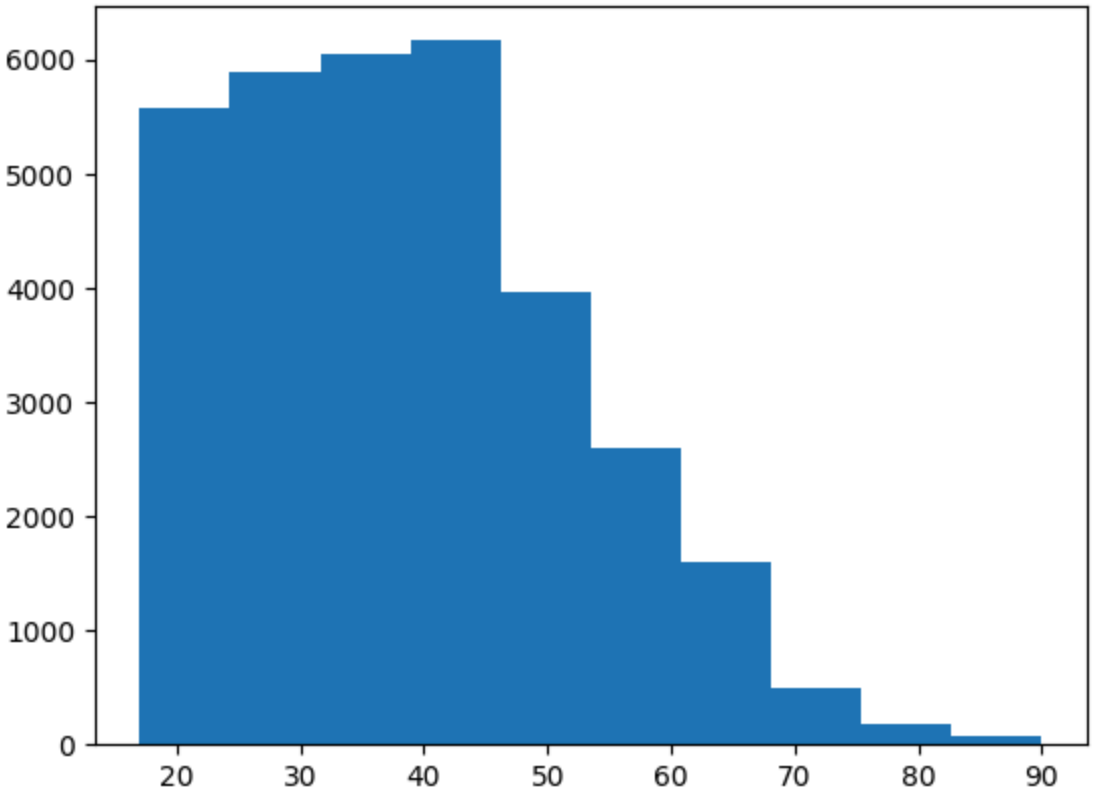
Out[5]:
```
(array([' <=50K', ' >50K'], dtype=object), array([24720,  7841]))
```

Podemos ver que majoritariamente os dados se referem a pessoas que ganham menos de 50 mil no ano.

In [6]:
```
sns.countplot(x = base_census['income'])
```

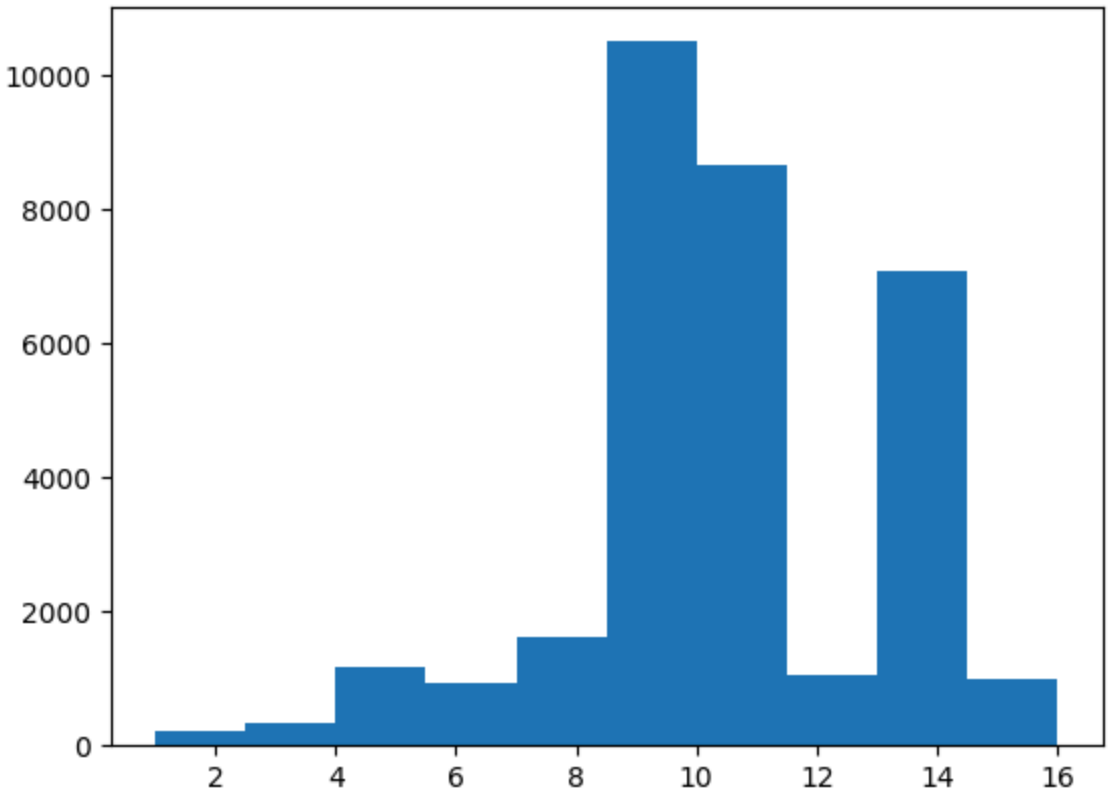


In [7]:
```
plt.hist(x = base_census['age']);
```



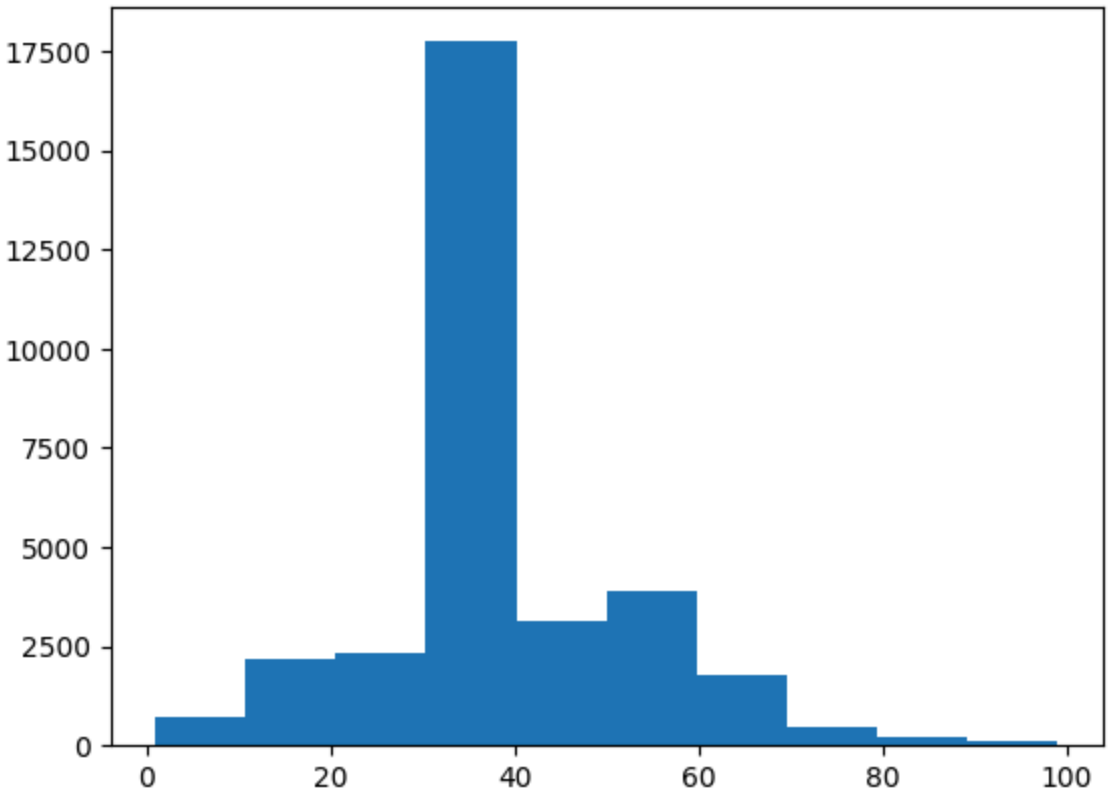
Podemos ver que a maioria tem entre 8 a 12 anos de dedicação aos estudos.

In [8]:
```
plt.hist(x = base_census['education-num']);
```



E trabalham de 30 a 40 horas por semana.

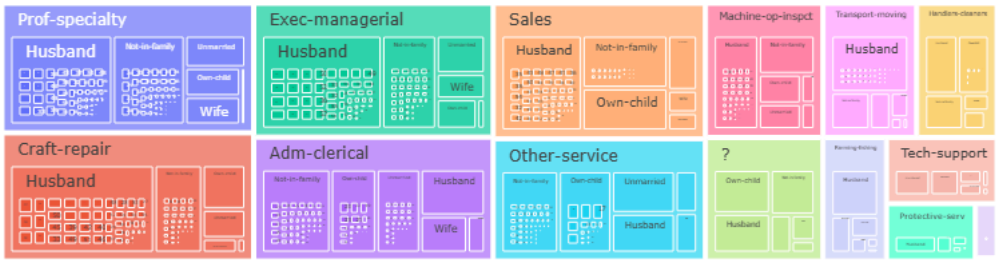
In [9]:
```
plt.hist(x = base_census['hour-per-week']);
```



Podemos observar a distribuição por categoria utilizando o gráfico treemap da biblioteca plotly.

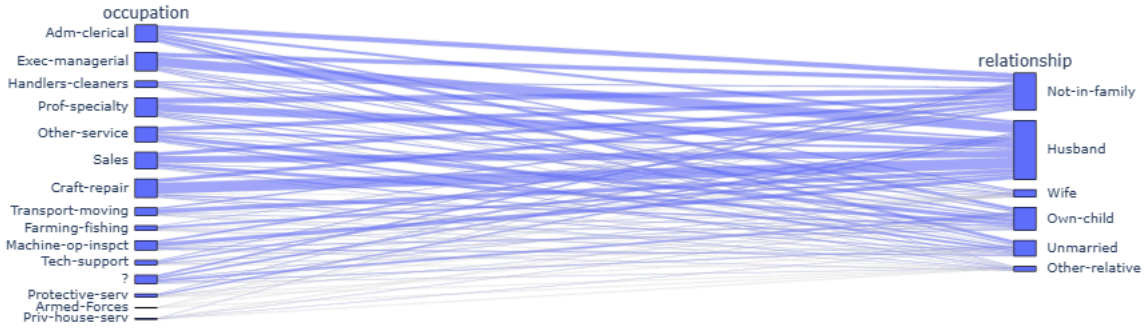
In [10]:
```
grafico = px.treemap(base_census, path=['workclass', 'age'])
grafico.show()
```

In [11]:
```
grafico = px.treemap(base_census, path=['occupation', 'relationship','age'])
grafico.show()
```

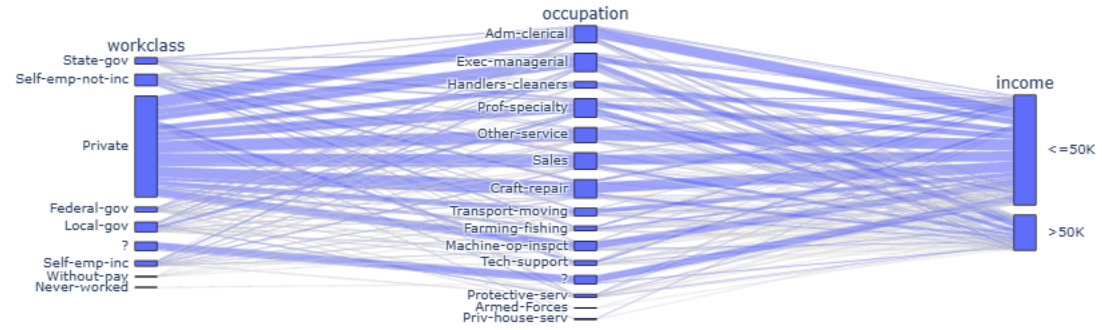


Assim como podemos ver a relação entre as categorias.

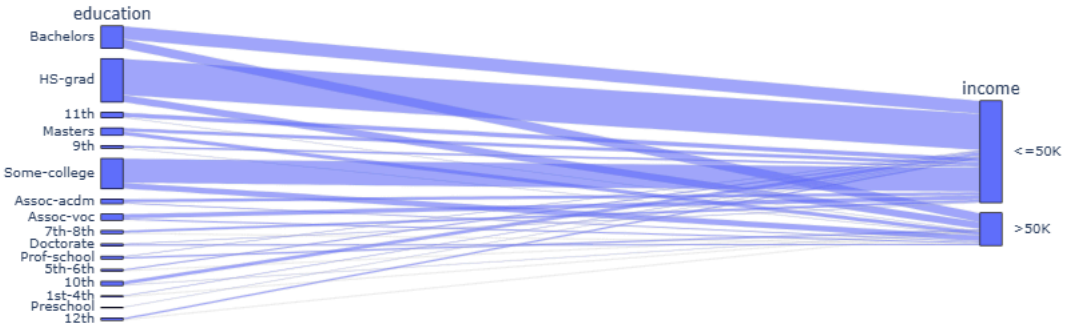
In [12]:
```
grafico = px.parallel_categories(base_census, dimensions=['occupation', 'relationsh
grafico.show()
```



In [13]:
```
grafico = px.parallel_categories(base_census, dimensions=['workclass', 'occupation'
grafico.show()
```



In [14]:
```
grafico = px.parallel_categories(base_census, dimensions=['education', 'income'])
grafico.show()
```



In [15]:
```
base_census.columns
```

Out[15]:
```
Index(['age', 'workclass', 'final-weight', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'race', 'sex',
       'capital-gain', 'capital-loos', 'hour-per-week', 'native-country',
       'income'],
      dtype='object')
```

Vamos dividir os nossos dados entre target e as variáveis exploratórias.

In [16]:
```
X_census = base_census.iloc[:, 0:14].values
y_census = base_census.iloc[:, 14].values
```

Conforme estudamos anteriormente, quando se tratam de dados categóricos é necessário transformar os rótulos em numéricos para melhor compreensão do algoritmo.

In [17]:
```
from sklearn.preprocessing import LabelEncoder
label_encoder_teste = LabelEncoder()
X_census[:,1]
```

Out[17]:
```
array([' State-gov', ' Self-emp-not-inc', ' Private', ..., ' Private',
       ' Private', ' Self-emp-inc'], dtype=object)
```

In [18]:
```
teste = label_encoder_teste.fit_transform(X_census[:,1])
teste
```

Out[18]:
```
array([7, 6, 4, ..., 4, 4, 5])
```

In [19]:
```
X_census[0]
```

Out[19]:
```
array([39, ' State-gov', 77516, ' Bachelors', 13, ' Never-married',
       ' Adm-clerical', ' Not-in-family', ' White', ' Male', 2174, 0, 40,
       ' United-States'], dtype=object)
```

In [20]:
```
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()
```

In [21]:
```
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])
```

In [22]:
```
X_census[0]
```

Out[22]:
```
array([39, 7, 77516, 9, 13, 4, 1, 1, 4, 1, 2174, 0, 40, 39], dtype=object)
```

In [23]:
```
X_census
```

Out[23]:
```
array([[39, 7, 77516, ..., 0, 40, 39],
       [50, 6, 83311, ..., 0, 13, 39],
       [38, 4, 215646, ..., 0, 40, 39],
       ...,
       [58, 4, 151910, ..., 0, 40, 39],
       [22, 4, 201490, ..., 0, 20, 39],
       [52, 5, 287927, ..., 0, 40, 39]], dtype=object)
```

In [24]:
```
len(np.unique(base_census['occupation']))
```

Out[24]:
```
15
```

In [25]:
```
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

onehotencoder_census = ColumnTransformer(transformers=[('OneHot', OneHotEncoder(),
```

In [27]:
```
X_census = onehotencoder_census.fit_transform(X_census)
X_census[0]
```

Out[27]:
```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 13 stored elements and shape (1, 108)>
```

In [28]:
```
X_census.shape
```

Out[28]:
```
(32561, 108)
```

In [31]:
```
from sklearn.preprocessing import StandardScaler
scaler_census = StandardScaler(with_mean=False)
X_census = scaler_census.fit_transform(X_census)
X_census[0]
```

Out[31]:
```
<Compressed Sparse Row sparse matrix of dtype 'float64'
        with 13 stored elements and shape (1, 108)>
```

In [117…
```
from sklearn.model_selection import train_test_split

X_census_treinamento, X_census_teste, y_census_treinamento, y_census_teste = train_

X_census_treinamento.shape, X_census_teste.shape
```

Out[117…
```
((26048, 108), (6513, 108))
```

In [118…
```
import pickle
with open('census.pkl', mode = 'wb') as f:
  pickle.dump([X_census_treinamento, y_census_treinamento, X_census_teste, y_census
```

Naïve Bayes

O algoritmo assume a premissa "ingênua" de que todas as características de entrada são independentes da presença umas das outras, dado a classe.Ele aplica o teorema de bayes para calcular a probabilidade de um dado pertencer a uma classe específica. A fórmula essencial é:

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

$$P(A|B) = \frac{P(B|A) \times P(A)}{P(B)}$$

Onde A é cada classe e B os a quantidade de Dados.

Na prática temos uma base de dados original:

História do crédito	Dívida	Garantias	Renda anual	Risco
Ruim	Alta	Nenhuma	< 15.000	Alto
Desconhecida	Alta	Nenhuma	>= 15.000 a <= 35.000	Alto
Desconhecida	Baixa	Nenhuma	>= 15.000 a <= 35.000	Moderado
Desconhecida	Baixa	Nenhuma	> 35.000	Alto
Desconhecida	Baixa	Nenhuma	> 35.000	Baixo
Desconhecida	Baixa	Adequada	> 35.000	Baixo
Ruim	Baixa	Nenhuma	< 15.000	Alto
Ruim	Baixa	Adequada	> 35.000	Moderado
Boa	Baixa	Nenhuma	> 35.000	Baixo
Boa	Alta	Adequada	> 35.000	Baixo
Boa	Alta	Nenhuma	< 15.000	Alto
Boa	Alta	Nenhuma	>= 15.000 a <= 35.000	Moderado
Boa	Alta	Nenhuma	> 35.0000	Baixo
Ruim	Alta	Nenhuma	>= 15.000 a <= 35.000	Alto

E para cada feature faz a divisão relacionada com a classe que queremos prever:

Risco de crédito	História do crédito			Dívida		Garantias		Renda anual		
	Boa 5	Desconhecida 5	Ruim 4							
Alto 6/14	1/6	2/6	3/6							
Moderado 3/14	1/3	1/3	1/3							
Baixo 5/14	3/5	2/5	0							

História do crédito	Risco
Ruim	Alto
Desconhecida	Alto
Desconhecida	Moderado
Desconhecida	Alto
Desconhecida	Baixo
Desconhecida	Baixo
Ruim	Alto
Ruim	Moderado
Boa	Baixo
Boa	Baixo
Boa	Alto
Boa	Moderado
Boa	Baixo
Ruim	Alto

Risco de crédito	História do crédito			Dívida		Garantias		Renda anual		
	Boa 5	Desconhecida 5	Ruim 4	Alta 7	Baixa 7					
Alto 6/14	1/6	2/6	3/6	4/6	2/6					
Moderado 3/14	1/3	1/3	1/3	1/3	2/3					
Baixo 5/14	3/5	2/5	0	2/5	3/5					

Dívida	Risco
Alta	Alto
Alta	Alto
Baixa	Moderado
Baixa	Alto
Baixa	Baixo
Baixa	Baixo
Baixa	Alto
Baixa	Moderado
Baixa	Baixo
Alta	Baixo
Alta	Alto
Alta	Moderado
Alta	Baixo
Alta	Alto

Risco de crédito	História do crédito			Dívida		Garantias		Renda anual		
	Boa 5	Desconhecida 5	Ruim 4	Alta 7	Baixa 7	Nenhuma 11	Adequada 3			
Alto 6/14	1/6	2/6	3/6	4/6	2/6	6/6	0			
Moderado 3/14	1/3	1/3	1/3	1/3	2/3	2/3	1/3			
Baixo 5/14	3/5	2/5	0	2/5	3/5	3/5	2/5			

Garantias	Risco
Nenhuma	Alto
Nenhuma	Alto
Nenhuma	Moderado
Nenhuma	Alto
Nenhuma	Baixo
Adequada	Baixo
Nenhuma	Alto
Adequada	Moderado
Nenhuma	Baixo
Adequada	Baixo
Nenhuma	Alto
Nenhuma	Moderado
Nenhuma	Baixo
Nenhuma	Alto

Risco de crédito	História do crédito			Dívida		Garantias		Renda anual		
	Boa 5	Desconhecida 5	Ruim 4	Alta 7	Baixa 7	Nenhuma 11	Adequada 3	< 15 3	>= 15 <= 35 4	> 35 7
Alto 6/14	1/6	2/6	3/6	4/6	2/6	6/6	0	3/6	2/6	1/6
Moderado 3/14	1/3	1/3	1/3	1/3	2/3	2/3	1/3	0	2/3	1/3
Baixo 5/14	3/5	2/5	0	2/5	3/5	3/5	2/5	0	0	5/5

Renda anual	Risco
< 15.000	Alto
>= 15.000 a <= 35.000	Alto
>= 15.000 a <= 35.000	Moderado
> 35.000	Alto
> 35.000	Baixo
> 35.000	Baixo
< 15.000	Alto
> 35.000	Moderado
> 35.000	Baixo
> 35.000	Baixo
< 15.000	Alto
>= 15.000 a <= 35.000	Moderado
> 35.0000	Baixo
>= 15.000 a <= 35.000	Alto

Essa tabela pronta é o que vai gerar o aprendizado do algoritmo, a partir disso podemos prever novos valores com os cálculos de probabilidade de cada classe.

Risco de crédito	História do crédito			Dívida		Garantias		Renda anual		
	Boa 5	Desconhecida 5	Ruim 4	Alta 7	Baixa 7	Nenhuma 11	Adequada 3	< 15 3	>= 15 <= 35 4	> 35 7
Alto 6/14	1/6	2/6	3/6	4/6	2/6	6/6	0	3/6	2/6	1/6
Moderado 3/14	1/3	1/3	1/3	1/3	2/3	2/3	1/3	0	2/3	1/3
Baixo 5/14	3/5	2/5	0	2/5	3/5	3/5	2/5	0	0	5/5

História = Boa
Dívida = Alta
Garantias = Nenhuma
Renda = > 35

Soma: 0,0079 + 0,0052 + 0,0514 = **0,0645**

P(Alto) = 6/14 * 1/6 * 4/6 * 6/6 * 1/6
P(Alto) = 0,0079
P(Alto) = 0,0079 / 0,0645 * 100 = **12,24%**

P(Moderado) = 3/14 * 1/3 * 1/3 * 2/3 * 1/3
P(Moderado) = 0,0052
P(Moderado) = 0,0052 / 0,0645 * 100 = **8,06%**

P(Baixo) = 5/14 * 3/5 * 2/5 * 3/5 * 5/5
P(Baixo) = 0,0514
P(Baixo) = 0,0514 / 0,0645 * 100 = **79,68%**

Métricas de desempenho para classificação.

Para avaliar as métricas abaixo considere que:

tp = True Positive; Quantidade de dados que eram de uma classe e foram preditas corretamente.

tn = True Negative. Quantidade de dados que não eram de um classe e foram preditas corretamente.

fp = False Positiive. Quantidade de dados que eram de uma classe e foram preditas incorretamente.

fn = False Negative. Quantidade de dados que não eram de uma classe e foram preditas incorretamente

Acurácia

Mede a porcentagem/proporção de valores que foram preditos de forma correta. ((tp + tn) / (tp + tn + fp + fn))

Precisão

Retorna a proporção de todos os dados que foram predições corretas apenas para quais seu modelo apontou como verdadeiras. (tp / (tp + fp)

Recall

Retorna a proporção de todos os dados que eram de fato verdadeiras e quantas foram corretamente preditas como positiva. (tp / (tp + fn).

F1_SCORE

Ele é a média harmônica entre a precisão e o recall. 2(precisao*recall/precisao+recall)

Matriz de confusão

Mostra a matriz de confusão (tp, tn, fp, fn). Funciona para quando a mais de duas classes, mostrando nas linhas o que realmente existia e nas colunas o que foi previsto.

Vamos testar, como os dados foram tratados anteriormente para não termos que rodar toda vez esses tratamentos podemos salvar os dados tratados num arquivo pkl, PKL é um arquivo binário gerado pelo módulo pickle do Python para serializar (salvar) um objeto Python. Isso permite que objetos complexos, como modelos de aprendizado de máquina ou estruturas de dados.

In [119…
```
from sklearn.naive_bayes import GaussianNB
import pickle
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pick

naive_census = GaussianNB()
naive_census.fit(X_census_treinamento.toarray(), y_census_treinamento)
previsoes = naive_census.predict(X_census_teste.toarray())
previsoes
```

Out[119…
```
array([' >50K', ' >50K', ' >50K', ..., ' >50K', ' >50K', ' >50K'],
      dtype='<U6')
```

In [120…
```
y_census_teste
```

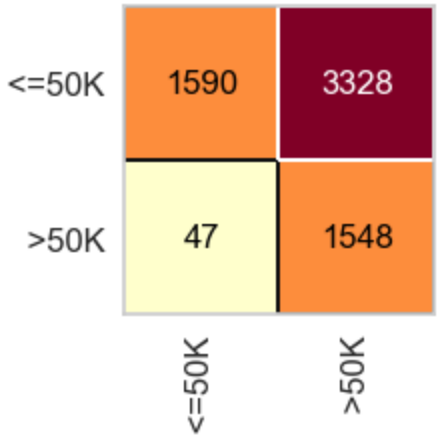
Out[120…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' >50K'],
      dtype=object)
```

In [121…
```
from sklearn.metrics import accuracy_score, classification_report
accuracy_score(y_census_teste, previsoes)
```

Out[121…
```
0.4818056195301704
```

In [122…
```
from yellowbrick.classifier import ConfusionMatrix
plt.figure(figsize=(2,2))
cm = ConfusionMatrix(naive_census)
cm.fit(X_census_treinamento.toarray(), y_census_treinamento)
cm.score(X_census_teste.toarray(), y_census_teste)
```

Out[122… 0.4818056195301704



In [123…
```
print(classification_report(y_census_teste, previsoes))
```

```
              precision    recall  f1-score   support

       <=50K       0.97      0.32      0.49      4918
        >50K       0.32      0.97      0.48      1595

    accuracy                           0.48      6513
   macro avg       0.64      0.65      0.48      6513
weighted avg       0.81      0.48      0.48      6513
```

Árvore de Decisão

In [165…
```
from sklearn.tree import DecisionTreeClassifier,plot_tree
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pick
```

In [168…
```
from mlxtend.plotting import plot_decision_regions

clf = DecisionTreeClassifier(max_depth=3)
clf.fit(X_census_treinamento, y_census_treinamento)
```

Out[168… ▾ DecisionTreeClassifier

▶ Parameters

A árvore de decisão tem como objetivo prever se uma pessoa ganha mais ou menos de 50 mil dólares por ano (classes <=50K e >50K).

Cada nó representa uma decisão baseada em uma variável (feature), e o modelo segue essas divisões até chegar em uma folha, que indica o resultado final da previsão.Em cada nó, o gráfico apresenta:

- Feature usada na divisão: variável que define o ponto de separação entre os grupos (ex.: relationship <= -0.589).
- Gini: mede o nível de impureza do nó; quanto mais próximo de 0, mais homogêneo é o grupo.
- Samples: quantidade de exemplos (linhas) que chegaram a esse nó.
- Value = [x, y]: número de amostras de cada classe (<=50K e >50K).
- Class: classe predominante entre as amostras daquele nó, ou seja, a previsão que o modelo faz para esse grupo.

In [126…
```
X_census_treinamento.shape, y_census_treinamento.shape
```

Out[126…
```
((26048, 108), (26048,))
```

In [127…
```
X_census_teste.shape, y_census_teste.shape
```

Out[127…
```
((6513, 108), (6513,))
```

In [128…
```
arvore_census = DecisionTreeClassifier(criterion='entropy', random_state=0)
arvore_census.fit(X_census_treinamento, y_census_treinamento)
```

Out[128…

▾ DecisionTreeClassifier

▶ Parameters

In [129…
```
previsoes = arvore_census.predict(X_census_teste)
previsoes
```

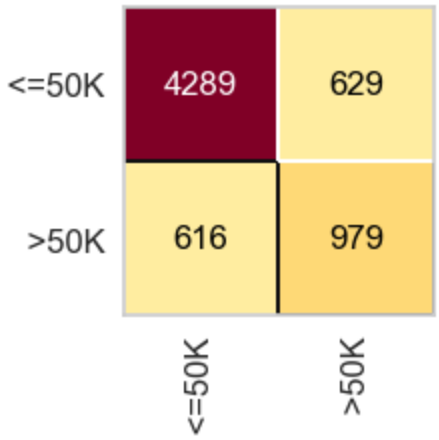
Out[129…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' >50K'],
      dtype=object)
```

In [130…
```
accuracy_score(y_census_teste, previsoes)
```

Out[130…
```
0.8088438507600184
```

In [131…
```
from yellowbrick.classifier import ConfusionMatrix
plt.figure(figsize=(2,2))
cm = ConfusionMatrix(arvore_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

Out[131…
```
0.8088438507600184
```



In [132…
```
print(classification_report(y_census_teste, previsoes))
```

```
              precision    recall  f1-score   support

       <=50K       0.87      0.87      0.87      4918
        >50K       0.61      0.61      0.61      1595

    accuracy                           0.81      6513
   macro avg       0.74      0.74      0.74      6513
weighted avg       0.81      0.81      0.81      6513
```

Random Forest

In [133…
```
from sklearn.ensemble import RandomForestClassifier
```

In [134…
```
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pick
```

In [135…
```
X_census_treinamento.shape, y_census_treinamento.shape
```

Out[135…
```
((26048, 108), (26048,))
```

In [136…
```
X_census_teste.shape, y_census_teste.shape
```

Out[136…
```
((6513, 108), (6513,))
```

In [137…
```
y_census_treinamento
```

Out[137…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' <=50K', ' >50K', ' <=50K'],
      dtype=object)
```

In [138…
```
random_forest_census = RandomForestClassifier(n_estimators=100, criterion='entropy'
random_forest_census.fit(X_census_treinamento, y_census_treinamento)
```

Out[138…

RandomForestClassifier

▶ Parameters

In [139…
```
previsoes = random_forest_census.predict(X_census_teste)
previsoes
```

Out[139…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' >50K'],
      dtype=object)
```

In [140…
```
y_census_teste
```

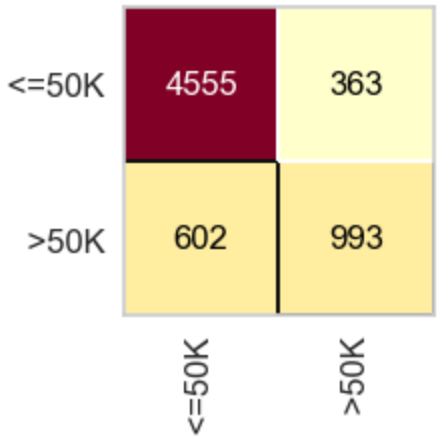
Out[140…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' >50K'],
      dtype=object)
```

In [141…
```
accuracy_score(y_census_teste, previsoes)
```

Out[141…
```
0.8518347919545525
```

In [142…
```
plt.figure(figsize=(2,2))
cm = ConfusionMatrix(random_forest_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

Out[142…
```
0.8518347919545525
```



In [143…
```
print(classification_report(y_census_teste, previsoes))
```

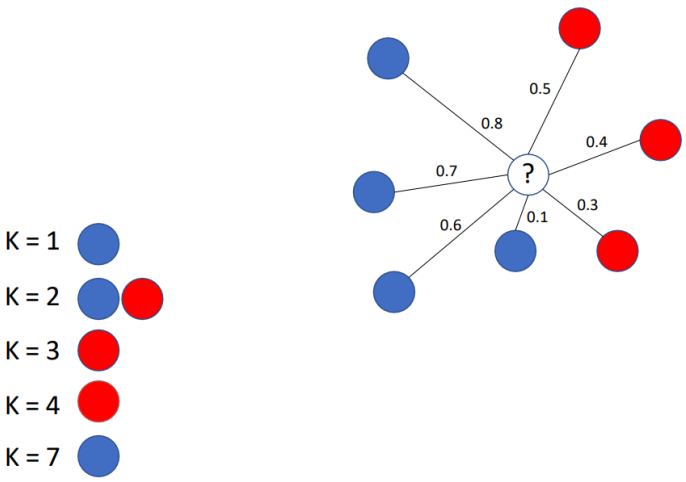
```
              precision    recall  f1-score   support

       <=50K       0.88      0.93      0.90      4918
        >50K       0.73      0.62      0.67      1595

    accuracy                           0.85      6513
   macro avg       0.81      0.77      0.79      6513
weighted avg       0.85      0.85      0.85      6513
```

KNN (K-Nearest Neighbors)

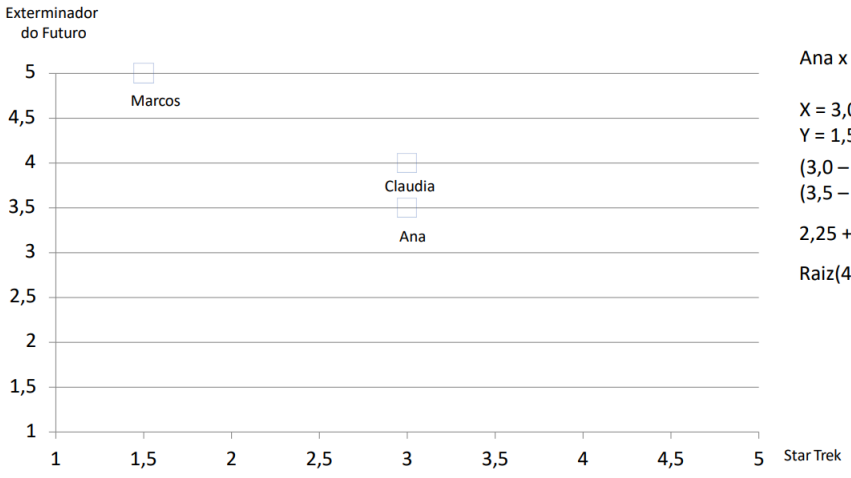
“K Vizinhos Mais Próximos” — é um algoritmo simples de classificação, que classifica um novo dado olhando quem são os vizinhos mais próximos dele. Onde o valor de K determina o número de vizinhos que o modelo vai considerar.



Calcula a distância entre o novo ponto e todos os pontos do conjunto de treinamento (geralmente usa-se a distância Euclidiana).

$$DE(x,y) = \sqrt{\sum_{i}^{p}(x_i - y_i)^2}$$

Pega os K vizinhos mais próximos (os que têm menor distância).



Ana x Marcos	Ana x Cláudia
X = 3,0 3,5	X = 3,0 3,5
Y = 1,5 5,0	Y = 3,0 4,0
$(3,0 - 1,5)^2 = 2,25$	$(3,0 - 3,5)^2 = 0,25$
$(3,5 - 5,0)^2 = 2,25$	$(3,0 - 4,0)^2 = 1,00$
2,25 + 2,25 = 4,5	0,25 + 1,00 = 1,25
Raiz(4,5) = **2,12**	Raiz(1,25) = **1,11**

O modelo escolhe a classe mais frequente entre os vizinhos.

Filme	Violência	Romance	Ação	Comédia	Classe
Invocação do Mal	0,6	0,0	0,3	0,0	Terror
Floresta Maldita	0,9	0,0	0,5	0,1	Terror
Meu Passado me Condena	0,1	0,2	0,1	0,9	Comédia
Tirando o atraso	0,0	0,2	0,2	0,8	Comédia

Violência = 0.8
Romance = 0.1
Ação = 0.5
Comédia = 0.0

A Hora do Pesadelo

Pesadelo x Invocação
0,8 0,1 0,5 0,0
0,6 0,0 0,3 0,0

$0,2^2 + 0,1^2 + 0,2^2 + 0$
0,04 + 0,01 + 0,04 = 0,09
Raiz(0,09) = **0,30**

Pesadelo x Floresta
0,8 0,1 0,5 0,0
0,9 0,0 0,5 0,1

$0,1^2 + 0,1^2 + 0 + 0,1^2$
0,01 + 0,01 + 0,01 = 0,03
Raiz(0,03) = **0,17**

Pesadelo x Passado
0,8 0,1 0,5 0,0
0,1 0,2 0,1 0,9

$0,7^2 + 0,1^2 + 0,4^2 + 0,9^2$
0,49 + 0,01 + 0,16 + 0,8 = 1,46
Raiz(1,46) = **1,20**

Pesadelo x Atraso
0,8 0,1 0,5 0,0
0,0 0,2 0,2 0,8

$0,8^2 + 0,1^2 + 0,4^2 + 0,8^2$
0,64 + 0,01 + 0,16 + 0,64 = 1,45
Raiz(1,45) = **1,20**

Importante ressaltar que nesse algoritmo não é criado um modelo, e sim a memorização dos dados vizinhos e calcula as distâncias para determinar a classe.

Obs.: É impressindível a utilização do StandardScaler para que escalas maiores não prejudiquem os cálculos.

In [144…
```
from sklearn.neighbors import KNeighborsClassifier
```

In [145…
```
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pick
```

In [146…
```
X_census_treinamento.shape, y_census_treinamento.shape
```

Out[146… ((26048, 108), (26048,))

In [147…
```
X_census_teste.shape, y_census_teste.shape
```

Out[147… ((6513, 108), (6513,))

In [148…
```
knn_census = KNeighborsClassifier(n_neighbors=10)
knn_census.fit(X_census_treinamento, y_census_treinamento)
```

Out[148…

▾ KNeighborsClassifier

▶ Parameters

In [149…
```
previsoes = knn_census.predict(X_census_teste)
previsoes
```

Out[149…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' >50K'],
      dtype=object)
```

In [150…
```
y_census_teste
```

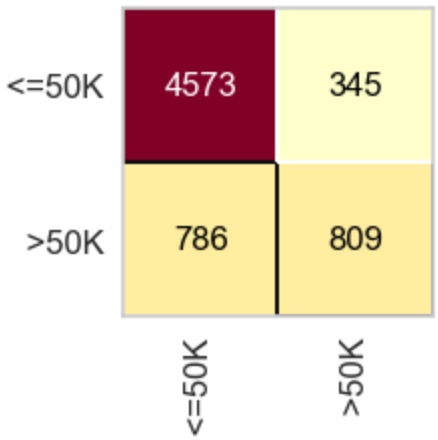
Out[150…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' >50K'],
      dtype=object)
```

In [151…
```
accuracy_score(y_census_teste, previsoes)
```

Out[151… 0.8263473053892215

In [152…
```
plt.figure(figsize=(2,2))
cm = ConfusionMatrix(knn_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

Out[152… 0.8263473053892215

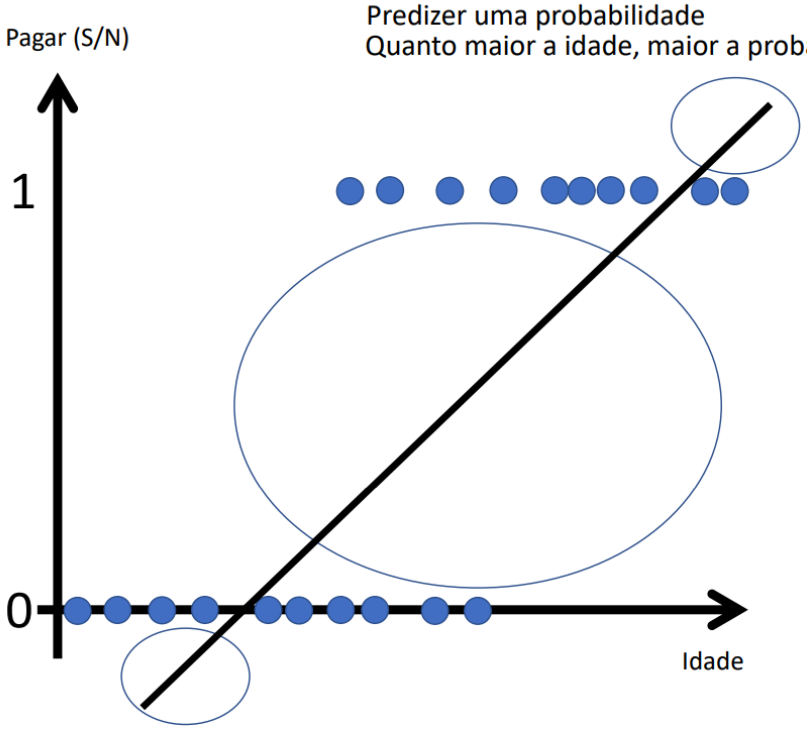


In [153…
```
print(classification_report(y_census_teste, previsoes))
```

	precision	recall	f1-score	support
<=50K	0.85	0.93	0.89	4918
>50K	0.70	0.51	0.59	1595
accuracy			0.83	6513
macro avg	0.78	0.72	0.74	6513
weighted avg	0.82	0.83	0.82	6513

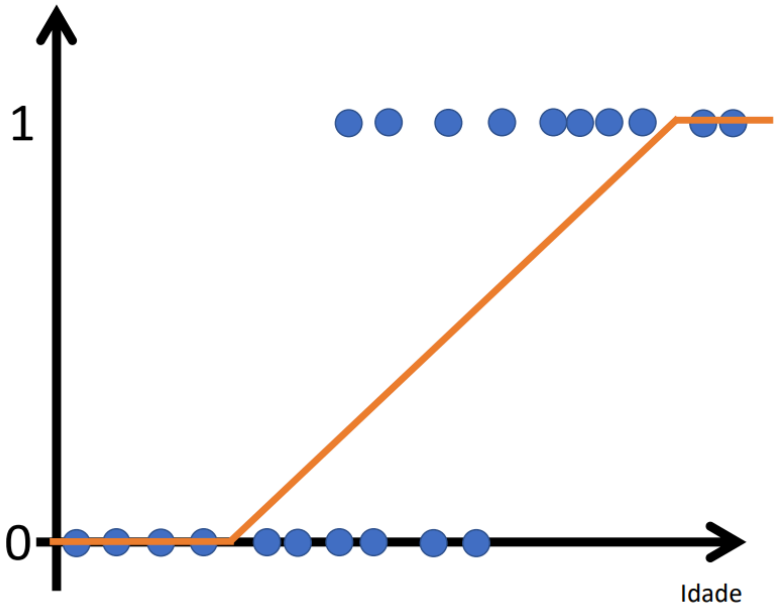
Regressão Logística

Existe algum tipo de correlação entre a idade e o pagamento?
Predizer uma probabilidade
Quanto maior a idade, maior a probabilidade de pagar



Função sigmoide

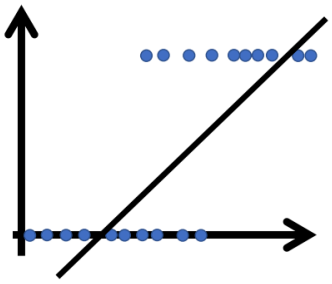
Encontrar a melhor linha para encaixar nos dados



$$y = b_0 + b_1 * x$$

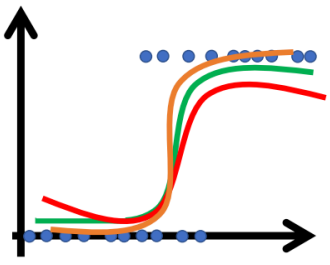
x = idade

Equação da reta



$$p = \frac{1}{1 + e^{-y}}$$

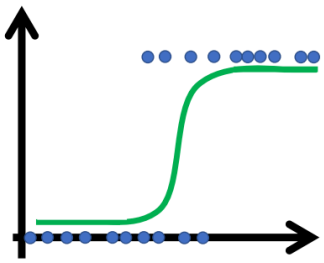
Função sigmoide



$$\log(\frac{p}{1-p}) = b_0 + b_1 * x$$

Transformação “logit”

História do crédito	Dívida	Garantias	Renda anual	Risco
2	0	0	0	Alto
1	0	0	1	Alto
1	1	0	2	Alto
1	1	0	2	Baixo
1	1	1	2	Baixo
2	1	0	0	Alto
0	1	0	2	Baixo
0	0	1	2	Baixo
0	0	0	0	Alto
0	0	0	2	Baixo
2	0	0	1	Alto



Encontrar os parâmetros B_0, B_1, B_2 ... B_N

B_0 = -0,12

B_1 (história do crédito) = -0,71
B_2 (dívida) = 0,24
B_3 (garantias) = -0,54
B_4 (renda anual) = 1.07

$$y = b_0 + b_1 * x$$
Equação da reta

$$p = \frac{1}{1 + e^{-y}}$$
Função sigmoide

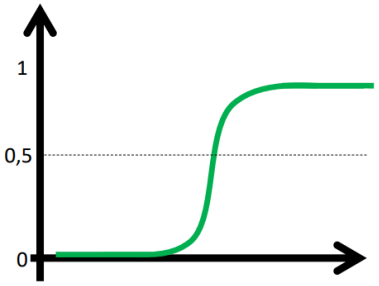
B_0 = -0,12

B_1 (história do crédito) = -0,71
B_2 (dívida) = 0,24
B_3 (garantias) = -0,54
B_4 (renda anual) = 1.07

História = Boa (0)
Dívida = Alta (0)
Garantias = Nenhuma (1)
Renda = > 35 (2)

$$y = b_0 + b_1 * \text{história} + b_2 * \text{dívida} + b_3 * \text{garantias} + b_4 * \text{renda}$$
$$y = -0{,}12 + (-0{,}71 * 0) + 0{,}24 * 0 + (-0{,}54 * 1) + 1{,}07 * 2$$
$$y = 1{,}48$$

$$p = \frac{1}{1 + e^{-(1{,}48)}}$$
$$p = 0{,}81$$



Risco alto = 0
Risco baixo = 1

In [154…
```
from sklearn.linear_model import LogisticRegression
```

In [155…
```
with open('census.pkl', 'rb') as f:
  X_census_treinamento, y_census_treinamento, X_census_teste, y_census_teste = pick
```

In [156…
```
X_census_treinamento.shape, y_census_treinamento.shape
```

Out[156…
```
((26048, 108), (26048,))
```

In [157…
```
X_census_teste.shape, y_census_teste.shape
```

Out[157…
```
((6513, 108), (6513,))
```

In [158…
```
logistic_census = LogisticRegression(random_state = 1,max_iter=100)
logistic_census.fit(X_census_treinamento, y_census_treinamento)
```

Out[158…

▾ LogisticRegression

▶ Parameters

In [159…
```
previsoes = logistic_census.predict(X_census_teste)
previsoes
```

Out[159…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' <=50K'],
      dtype=object)
```

In [160…
```
y_census_teste
```

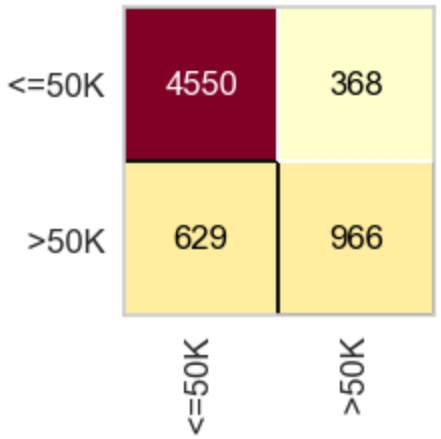
Out[160…
```
array([' <=50K', ' <=50K', ' <=50K', ..., ' >50K', ' <=50K', ' >50K'],
      dtype=object)
```

In [161…
```
accuracy_score(y_census_teste, previsoes)
```

Out[161…
```
0.84692154153232
```

In [162…
```
plt.figure(figsize=(2,2))
cm = ConfusionMatrix(logistic_census)
cm.fit(X_census_treinamento, y_census_treinamento)
cm.score(X_census_teste, y_census_teste)
```

Out[162…
```
0.84692154153232
```



In [ ]: