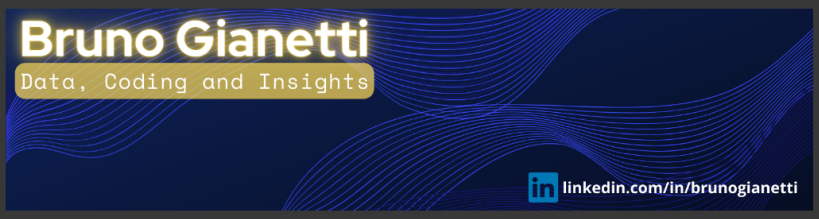



```
!pip install seaborn==0.9.0
```

```
Requirement already satisfied: seaborn==0.9.0 in /usr/local/lib/python3.6/dist-packages (0.9.0)
Requirement already satisfied: scipy>=0.14.0 in /usr/local/lib/python3.6/dist-packages (from seaborn==0.9.0) (1.1.0)
Requirement already satisfied: matplotlib>=1.4.3 in /usr/local/lib/python3.6/dist-packages (from seaborn==0.9.0) (2.1.2)
Requirement already satisfied: numpy>=1.9.3 in /usr/local/lib/python3.6/dist-packages (from seaborn==0.9.0) (1.14.6)
Requirement already satisfied: pandas>=0.15.2 in /usr/local/lib/python3.6/dist-packages (from seaborn==0.9.0) (0.22.0)
Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.3->seaborn==0.9.0) (2.5.3)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.3->seaborn==0.9.0) (1.11.0)
Requirement already satisfied: pytz in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.3->seaborn==0.9.0) (2018.7)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.3->seaborn==0.9.0) (2.3.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.6/dist-packages (from matplotlib>=1.4.3->seaborn==0.9.0) (0.10.0)
```

```
[ ] import seaborn as sns

    print(sns.__version__)
```

```
0.9.0
```

```
[ ] !pip install graphviz==0.9
    !pip install pydot
```

```
Requirement already satisfied: graphviz==0.9 in /usr/local/lib/python3.6/dist-packages (0.9)
Requirement already satisfied: pydot in /usr/local/lib/python3.6/dist-packages (1.3.0)
Requirement already satisfied: pyparsing>=2.1.4 in /usr/local/lib/python3.6/dist-packages (from pydot) (2.3.0)
```

```
[ ] !apt-get install graphviz
```

```
Reading package lists... Done
Building dependency tree
Reading state information... Done
graphviz is already the newest version (2.40.1-2).
0 upgraded, 0 newly installed, 0 to remove and 8 not upgraded.
```

```
[ ] import pandas as pd

    uri = "https://gist.githubusercontent.com/guilhermesilveira/e99a526b2e7ccc6c3b70f53db43a87d2/raw/1605fc74aa778066bf2e6695e24d53cf65f2f447/machine-learning-carros-simulacao.csv"
    dados = pd.read_csv(uri).drop(columns=["Unnamed: 0"], axis=1)
    dados.head()
```

	preco	vendido	idade_do_modelo	km_por_ano
0	30941.02	1	18	35085.22134
1	40557.96	1	20	12622.05362
2	89627.50	0	12	11440.79806
3	95276.14	0	3	43167.32682
4	117384.68	1	4	12770.11290

```
[ ] # situação horrível de "azar" onde as classes estão ordenadas por padrão

    dados_azar = dados.sort_values("vendido", ascending=True)
    x_azar = dados_azar[["preco", "idade_do_modelo","km_por_ano"]]
    y_azar = dados_azar["vendido"]
    dados_azar.head()
```

	preco	vendido	idade_do_modelo	km_por_ano
4999	74023.29	0	12	24812.80412
5322	84843.49	0	13	23095.63834
5319	83100.27	0	19	36240.72746
5316	87932.13	0	16	32249.56426
5315	77937.01	0	15	28414.50704

```
[ ] from sklearn.model_selection import cross_validate
    from sklearn.dummy import DummyClassifier
    import numpy as np

    SEED = 301
    np.random.seed(SEED)

    modelo = DummyClassifier()
    results = cross_validate(modelo, x_azar, y_azar, cv = 10, return_train_score=False)
    media = results['test_score'].mean()
    desvio_padrao = results['test_score'].std()
    print("Accuracy com dummy stratified, 10 = [%.2f, %.2f]" % ((media - 2 * desvio_padrao)*100, (media + 2 * desvio_padrao) * 100))
```

```
Accuracy com dummy stratified, 10 = [49.79, 53.45]
```

```
[ ] from sklearn.model_selection import cross_validate
    from sklearn.tree import DecisionTreeClassifier

    SEED = 301
    np.random.seed(SEED)

    modelo = DecisionTreeClassifier(max_depth=2)
    results = cross_validate(modelo, x_azar, y_azar, cv = 10, return_train_score=False)
    media = results['test_score'].mean()
    desvio_padrao = results['test_score'].std()
    print("Accuracy com cross validation, 10 = [%.2f, %.2f]" % ((media - 2 * desvio_padrao)*100, (media + 2 * desvio_padrao) * 100))
```

```
Accuracy com cross validation, 10 = [73.83, 77.73]
```

```
[ ] # gerando dados elatorios de modelo de carro para simulacao de agrupamento ao usar nosso estimador

    np.random.seed(SEED)
    dados['modelo'] = dados.idade_do_modelo + np.random.randint(-2, 3, size=10000)
    dados.modelo = dados.modelo + abs(dados.modelo.min()) + 1
    dados.head()
```

	preco	vendido	idade_do_modelo	km_por_ano	modelo

0	30941.02	1	18	35085.22134	18
1	40557.96	1	20	12622.05362	24
2	89627.50	0	12	11440.79806	14
3	95276.14	0	3	43167.32682	6
4	117384.68	1	4	12770.11290	5

```
def imprime_resultados(results):
  media = results['test_score'].mean() * 100
  desvio = results['test_score'].std() * 100
  print("Accuracy médio %.2f" % media)
  print("Intervalo [%.2f, %.2f]" % (media - 2 * desvio, media + 2 * desvio))
```

```
# GroupKFold em um pipeline com StandardScaler e SVC

from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GroupKFold

SEED = 301
np.random.seed(SEED)

scaler = StandardScaler()
modelo = SVC()

pipeline = Pipeline([('transformacao',scaler), ('estimador',modelo)])

cv = GroupKFold(n_splits = 10)
results = cross_validate(pipeline, x_azar, y_azar, cv = cv, groups = dados.modelo, return_train_score=False)
imprime_resultados(results)
```

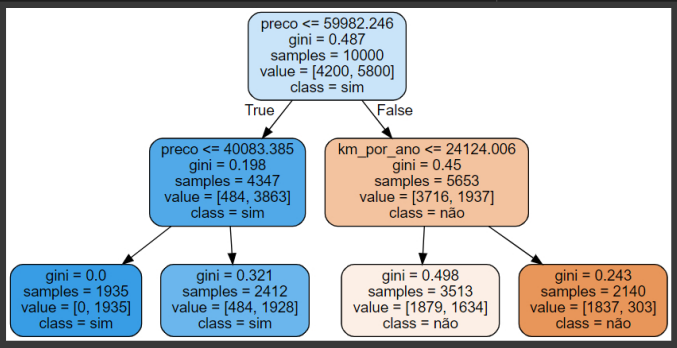
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-27-3b39d7bb16cb> in <module>()
     14
     15 cv = GroupKFold(n_splits = 10)
---> 16 results = cross_validate(pipeline, x_azar, y_azar, cv = cv, groups = dados.modelo, return_train_score=False)
     17 imprime_resultados(results)

/usr/local/lib/python3.6/dist-packages/pandas/core/generic.py in __getattr__(self, name)
   3612             if name in self._info_axis:
   3613                 return self[name]
-> 3614             return object.__getattribute__(self, name)
   3615
   3616     def __setattr__(self, name, value):

AttributeError: 'DataFrame' object has no attribute 'modelo'
```

PESQUISAR NO STACK OVERFLOW

```
# GroupKFold para analisar como o modelo se comporta com novos grupos

from sklearn.model_selection import GroupKFold

SEED = 301
np.random.seed(SEED)

cv = GroupKFold(n_splits = 10)
modelo = DecisionTreeClassifier(max_depth=2)
results = cross_validate(modelo, x_azar, y_azar, cv = cv, groups = dados.modelo, return_train_score=False)
imprime_resultados(results)
```

```
Accuracy médio 75.78
Intervalo [73.67, 77.90]
```

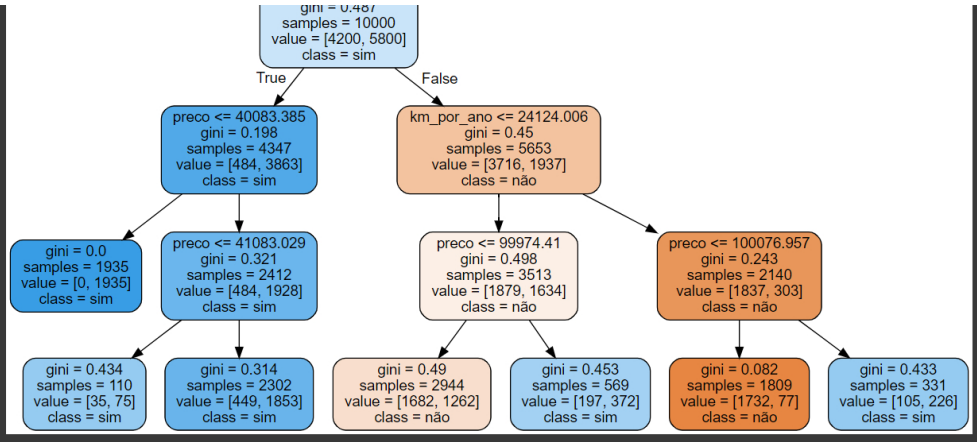
```
from sklearn.tree import export_graphviz
import graphviz

modelo.fit(x_azar, y_azar)
features = x_azar.columns
dot_data = export_graphviz(modelo, out_file=None, filled=True, rounded=True,
                           class_names=["não","sim"],
                           feature_names = features)
graph = graphviz.Source(dot_data)
graph
```




```
# GroupKFold para analisar como o modelo se comporta com novos grupos

from sklearn.model_selection import GroupKFold

SEED = 301
np.random.seed(SEED)

cv = GroupKFold(n_splits = 10)
modelo = DecisionTreeClassifier(max_depth=3)
results = cross_validate(modelo, x_azar, y_azar, cv = cv, groups = dados.modelo, return_train_score=False)
imprime_resultados(results)
```

```
Accuracy médio 78.67
Intervalo [76.40, 80.94]
```

```
from sklearn.tree import export_graphviz
import graphviz

modelo.fit(x_azar, y_azar)
features = x_azar.columns
dot_data = export_graphviz(modelo, out_file=None, filled=True, rounded=True,
                           class_names=["não","sim"],
                           feature_names = features)
graph = graphviz.Source(dot_data)
graph
```




```
[ ] # GroupKFold para analisar como o modelo se comporta com novos grupos

    from sklearn.model_selection import GroupKFold

    SEED = 301
    np.random.seed(SEED)

    cv = GroupKFold(n_splits = 10)
    modelo = DecisionTreeClassifier(max_depth=10)
    results = cross_validate(modelo, x_azar, y_azar, cv = cv, groups = dados.modelo, return_train_score=False)
    imprime_resultados(results)
```

```
Accuracy médio 77.19
Intervalo [75.26, 79.13]
```

Explorando hiper parâmetros em uma dimensão

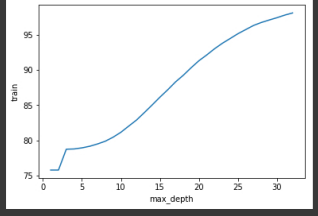
```
[ ] def roda_arvore_de_decisao(max_depth):
      SEED = 301
      np.random.seed(SEED)

      cv = GroupKFold(n_splits = 10)
      modelo = DecisionTreeClassifier(max_depth=max_depth)
      results = cross_validate(modelo, x_azar, y_azar, cv = cv, groups = dados.modelo, return_train_score=True)
      train_score = results['train_score'].mean() * 100
      test_score = results['test_score'].mean() * 100
      print("Arvore max_depth = %d, treino = %.2f, teste = %.2f" % (max_depth, train_score, test_score))
      tabela = [max_depth, train_score, test_score]
      return tabela

    resultados = [roda_arvore_de_decisao(i) for i in range(1,33)]
    resultados = pd.DataFrame(resultados, columns= ["max_depth","train","test"])
    resultados.head()
```

```
Arvore max_depth = 1, treino = 75.79, teste = 75.78
Arvore max_depth = 2, treino = 75.79, teste = 75.78
Arvore max_depth = 3, treino = 78.75, teste = 78.67
Arvore max_depth = 4, treino = 78.79, teste = 78.63
Arvore max_depth = 5, treino = 78.94, teste = 78.56
Arvore max_depth = 6, treino = 79.17, teste = 78.12
Arvore max_depth = 7, treino = 79.50, teste = 77.96
Arvore max_depth = 8, treino = 79.89, teste = 77.86
Arvore max_depth = 9, treino = 80.46, teste = 77.38
Arvore max_depth = 10, treino = 81.16, teste = 77.19
Arvore max_depth = 11, treino = 82.03, teste = 76.97
Arvore max_depth = 12, treino = 82.89, teste = 76.49
Arvore max_depth = 13, treino = 83.95, teste = 75.81
Arvore max_depth = 14, treino = 85.03, teste = 75.66
Arvore max_depth = 15, treino = 86.14, teste = 75.16
Arvore max_depth = 16, treino = 87.19, teste = 75.11
Arvore max_depth = 17, treino = 88.31, teste = 74.74
Arvore max_depth = 18, treino = 89.26, teste = 74.33
Arvore max_depth = 19, treino = 90.32, teste = 74.34
Arvore max_depth = 20, treino = 91.32, teste = 74.22
Arvore max_depth = 21, treino = 92.14, teste = 73.80
Arvore max_depth = 22, treino = 93.02, teste = 73.81
Arvore max_depth = 23, treino = 93.80, teste = 73.38
Arvore max_depth = 24, treino = 94.47, teste = 73.43
Arvore max_depth = 25, treino = 95.16, teste = 73.14
Arvore max_depth = 26, treino = 95.74, teste = 73.04
Arvore max_depth = 27, treino = 96.33, teste = 72.91
Arvore max_depth = 28, treino = 96.75, teste = 72.66
Arvore max_depth = 29, treino = 97.10, teste = 72.73
Arvore max_depth = 30, treino = 97.43, teste = 72.81
Arvore max_depth = 31, treino = 97.80, teste = 72.86
Arvore max_depth = 32, treino = 98.10, teste = 72.52
```

	max_depth	train	test
0	1	75.791169	75.784219
1	2	75.791169	75.784219
2	3	78.750993	78.672633
3	4	78.787628	78.632803
4	5	78.941007	78.555912

```
[ ] import seaborn as sns

    sns.lineplot(x = "max_depth", y = "train", data = resultados)
```

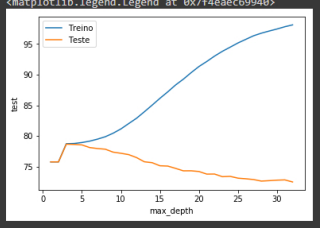
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4eb15d5748>
```



OVERFIT: ficou "perfeito" para o treino mas ruim para o teste

```
[ ] import matplotlib.pyplot as plt

    sns.lineplot(x = "max_depth", y = "train", data = resultados)
    sns.lineplot(x = "max_depth", y = "test", data = resultados)
    plt.legend(["Treino", "Teste"])
```

```
<matplotlib.legend.Legend at 0x7f4eaec69940>
```




```
[ ] resultados.sort_values("test", ascending=False).head()
```

	max_depth	train	test
2	3	78.750993	78.672633
3	4	78.787628	78.632803
4	5	78.941007	78.555912
5	6	79.170115	78.123266
6	7	79.496806	77.963185

Explorando hiper parâmetros em 2 dimensões

```
[ ] def roda_arvore_de_decisao(max_depth, min_samples_leaf):
      SEED = 301
      np.random.seed(SEED)

      cv = GroupKFold(n_splits = 10)
      modelo = DecisionTreeClassifier(max_depth=max_depth, min_samples_leaf = min_samples_leaf)
      results = cross_validate(modelo, x_azar, y_azar, cv = cv, groups = dados.modelo, return_train_score=True)
      train_score = results['train_score'].mean() * 100
      test_score = results['test_score'].mean() * 100
      print("Arvore max_depth = %d, min_samples_leaf = %d, treino = %.2f, teste = %.2f" % (max_depth, min_samples_leaf, train_score, test_score))
      tabela = [max_depth, min_samples_leaf, train_score, test_score]
      return tabela

    def busca():
      resultados = []
      for max_depth in range(1,33):
        for min_samples_leaf in [32, 64, 128, 256]:
          tabela = roda_arvore_de_decisao(max_depth, min_samples_leaf)
          resultados.append(tabela)
      resultados = pd.DataFrame(resultados, columns= ["max_depth","min_samples_leaf","train","test"])
      return resultados

    resultados = busca()
    resultados.head()
```

```
Arvore max_depth = 1, min_samples_leaf = 32, treino = 75.79, teste = 75.78
Arvore max_depth = 1, min_samples_leaf = 64, treino = 75.79, teste = 75.78
Arvore max_depth = 1, min_samples_leaf = 128, treino = 75.79, teste = 75.78
Arvore max_depth = 1, min_samples_leaf = 256, treino = 75.79, teste = 75.78
Arvore max_depth = 2, min_samples_leaf = 32, treino = 75.79, teste = 75.78
Arvore max_depth = 2, min_samples_leaf = 64, treino = 75.79, teste = 75.78
Arvore max_depth = 2, min_samples_leaf = 128, treino = 75.79, teste = 75.78
Arvore max_depth = 2, min_samples_leaf = 256, treino = 75.79, teste = 75.78
Arvore max_depth = 3, min_samples_leaf = 32, treino = 78.75, teste = 78.67
Arvore max_depth = 3, min_samples_leaf = 64, treino = 78.75, teste = 78.67
Arvore max_depth = 3, min_samples_leaf = 128, treino = 78.75, teste = 78.67
Arvore max_depth = 3, min_samples_leaf = 256, treino = 78.75, teste = 78.67
Arvore max_depth = 4, min_samples_leaf = 32, treino = 78.75, teste = 78.67
Arvore max_depth = 4, min_samples_leaf = 64, treino = 78.75, teste = 78.67
Arvore max_depth = 4, min_samples_leaf = 128, treino = 78.75, teste = 78.67
Arvore max_depth = 4, min_samples_leaf = 256, treino = 78.75, teste = 78.67
Arvore max_depth = 5, min_samples_leaf = 32, treino = 78.82, teste = 78.50
Arvore max_depth = 5, min_samples_leaf = 64, treino = 78.79, teste = 78.45
Arvore max_depth = 5, min_samples_leaf = 128, treino = 78.77, teste = 78.44
Arvore max_depth = 5, min_samples_leaf = 256, treino = 78.77, teste = 78.44
Arvore max_depth = 6, min_samples_leaf = 32, treino = 78.96, teste = 78.02
Arvore max_depth = 6, min_samples_leaf = 64, treino = 78.92, teste = 78.09
Arvore max_depth = 6, min_samples_leaf = 128, treino = 78.85, teste = 78.27
Arvore max_depth = 6, min_samples_leaf = 256, treino = 78.79, teste = 78.27
Arvore max_depth = 7, min_samples_leaf = 32, treino = 79.24, teste = 77.94
Arvore max_depth = 7, min_samples_leaf = 64, treino = 79.18, teste = 78.05
Arvore max_depth = 7, min_samples_leaf = 128, treino = 79.01, teste = 78.21
Arvore max_depth = 7, min_samples_leaf = 256, treino = 78.85, teste = 78.08
Arvore max_depth = 8, min_samples_leaf = 32, treino = 79.47, teste = 77.93
Arvore max_depth = 8, min_samples_leaf = 64, treino = 79.30, teste = 77.99
Arvore max_depth = 8, min_samples_leaf = 128, treino = 79.05, teste = 78.37
Arvore max_depth = 8, min_samples_leaf = 256, treino = 78.89, teste = 77.94
Arvore max_depth = 9, min_samples_leaf = 32, treino = 79.80, teste = 77.32
Arvore max_depth = 9, min_samples_leaf = 64, treino = 79.46, teste = 77.55
Arvore max_depth = 9, min_samples_leaf = 128, treino = 79.12, teste = 78.30
Arvore max_depth = 9, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 10, min_samples_leaf = 32, treino = 80.05, teste = 77.45
Arvore max_depth = 10, min_samples_leaf = 64, treino = 79.55, teste = 77.64
Arvore max_depth = 10, min_samples_leaf = 128, treino = 79.16, teste = 78.29
Arvore max_depth = 10, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 11, min_samples_leaf = 32, treino = 80.21, teste = 77.20
Arvore max_depth = 11, min_samples_leaf = 64, treino = 79.60, teste = 77.53
Arvore max_depth = 11, min_samples_leaf = 128, treino = 79.19, teste = 78.31
Arvore max_depth = 11, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 12, min_samples_leaf = 32, treino = 80.34, teste = 77.00
Arvore max_depth = 12, min_samples_leaf = 64, treino = 79.72, teste = 77.29
Arvore max_depth = 12, min_samples_leaf = 128, treino = 79.21, teste = 78.22
Arvore max_depth = 12, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 13, min_samples_leaf = 32, treino = 80.51, teste = 76.85
Arvore max_depth = 13, min_samples_leaf = 64, treino = 79.74, teste = 77.34
Arvore max_depth = 13, min_samples_leaf = 128, treino = 79.21, teste = 78.22
Arvore max_depth = 13, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 14, min_samples_leaf = 32, treino = 80.60, teste = 76.69
Arvore max_depth = 14, min_samples_leaf = 64, treino = 79.77, teste = 77.26
Arvore max_depth = 14, min_samples_leaf = 128, treino = 79.21, teste = 78.22
Arvore max_depth = 14, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 15, min_samples_leaf = 32, treino = 80.70, teste = 76.88
```

```
[ ] resultados.sort_values("test", ascending=False).head()
```

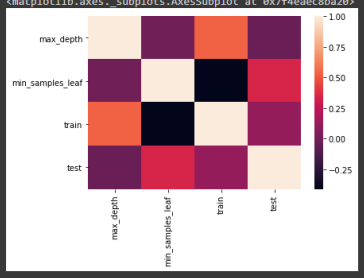
	max_depth	min_samples_leaf	train	test
15	4	256	78.750993	78.672633
12	4	32	78.750993	78.672633
14	4	128	78.750993	78.672633
13	4	64	78.750993	78.672633
8	3	32	78.750993	78.672633

```
[ ] corr = resultados.corr()
    corr
```

	max_depth	min_samples_leaf	train	test
max_depth	1.000000	0.000000	0.536705	-0.027675
min_samples_leaf	0.000000	1.000000	-0.409676	0.349011
train	0.536705	-0.409676	1.000000	0.116466
test	-0.027675	0.349011	0.116466	1.000000

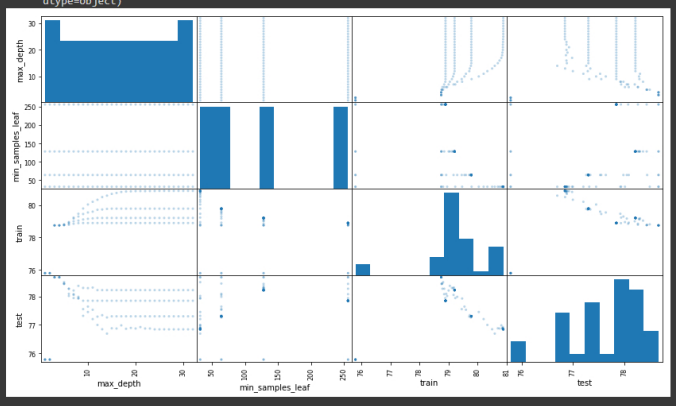
```
[ ] sns.heatmap(corr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4eaec8ba20>
```



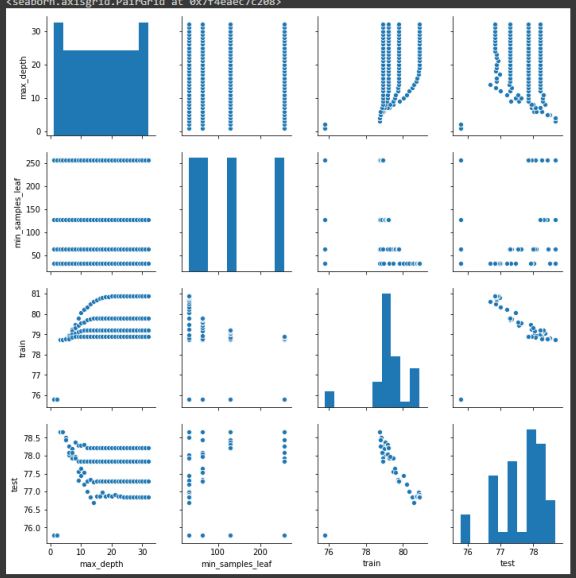

```
[ ] pd.scatter_matrix(resultados, figsize = (14, 8), alpha = 0.3)
```

```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:1: FutureWarning: pandas.scatter_matrix is deprecated. Use pandas.plotting.scatter_matrix instead
  """Entry point for launching an IPython kernel.
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f4eaec42160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eaeb21b70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eaeadb7f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eaea975c0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f4eaead15c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eaead15f8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eaea3c0f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae9f6128>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae9ae160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae9c2f98>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae990940>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae946e10>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae8ffe80>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae8bada0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae8705c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f4eae826780>]],
      dtype=object)
```




```
[ ] sns.pairplot(resultados)
```

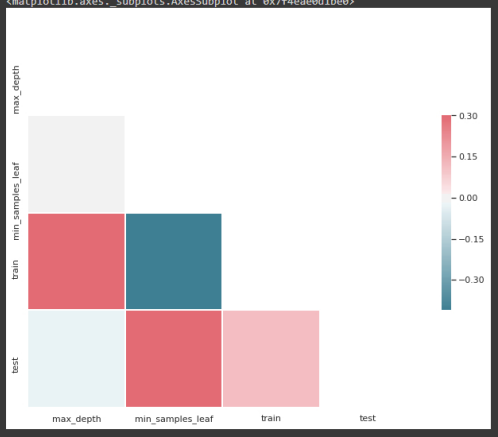
```
<seaborn.axisgrid.PairGrid at 0x7f4eaec7c208>
```




```
[ ] sns.set(style="white")

    # Generate a mask for the upper triangle
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    # Set up the matplotlib figure
    f, ax = plt.subplots(figsize=(11, 9))
```

```
# Generate a custom diverging colormap
cmap = sns.diverging_palette(220, 10, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4eae0d1be0>
```



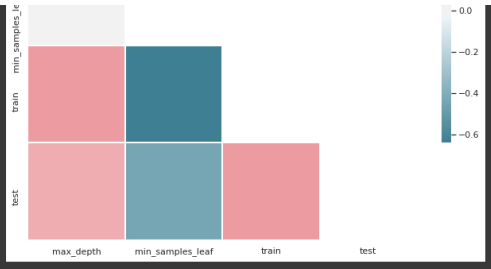

```
[ ] def busca():
      resultados = []
      for max_depth in range(1,33):
        for min_samples_leaf in [128, 192, 256, 512]:
          tabela = roda_arvore_de_decisao(max_depth, min_samples_leaf)
          resultados.append(tabela)
      resultados = pd.DataFrame(resultados, columns= ["max_depth","min_samples_leaf","train","test"])
      return resultados

    resultados = busca()
    resultados.head()
```

```
Arvore max_depth = 1, min_samples_leaf = 128, treino = 75.79, teste = 75.78
Arvore max_depth = 1, min_samples_leaf = 192, treino = 75.79, teste = 75.78
Arvore max_depth = 1, min_samples_leaf = 256, treino = 75.79, teste = 75.78
Arvore max_depth = 1, min_samples_leaf = 512, treino = 75.79, teste = 75.78
Arvore max_depth = 2, min_samples_leaf = 128, treino = 75.79, teste = 75.78
Arvore max_depth = 2, min_samples_leaf = 192, treino = 75.79, teste = 75.78
Arvore max_depth = 2, min_samples_leaf = 256, treino = 75.79, teste = 75.78
Arvore max_depth = 2, min_samples_leaf = 512, treino = 75.79, teste = 75.78
Arvore max_depth = 3, min_samples_leaf = 128, treino = 78.75, teste = 78.67
Arvore max_depth = 3, min_samples_leaf = 192, treino = 78.75, teste = 78.67
Arvore max_depth = 3, min_samples_leaf = 256, treino = 78.75, teste = 78.67
Arvore max_depth = 3, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 4, min_samples_leaf = 128, treino = 78.75, teste = 78.67
Arvore max_depth = 4, min_samples_leaf = 192, treino = 78.75, teste = 78.67
Arvore max_depth = 4, min_samples_leaf = 256, treino = 78.75, teste = 78.67
Arvore max_depth = 4, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 5, min_samples_leaf = 128, treino = 78.77, teste = 78.44
Arvore max_depth = 5, min_samples_leaf = 192, treino = 78.77, teste = 78.44
Arvore max_depth = 5, min_samples_leaf = 256, treino = 78.77, teste = 78.44
Arvore max_depth = 5, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 6, min_samples_leaf = 128, treino = 78.85, teste = 78.27
Arvore max_depth = 6, min_samples_leaf = 192, treino = 78.79, teste = 78.31
Arvore max_depth = 6, min_samples_leaf = 256, treino = 78.79, teste = 78.27
Arvore max_depth = 6, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 7, min_samples_leaf = 128, treino = 79.01, teste = 78.21
Arvore max_depth = 7, min_samples_leaf = 192, treino = 78.92, teste = 78.10
Arvore max_depth = 7, min_samples_leaf = 256, treino = 78.85, teste = 78.08
Arvore max_depth = 7, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 8, min_samples_leaf = 128, treino = 79.05, teste = 78.37
Arvore max_depth = 8, min_samples_leaf = 192, treino = 78.98, teste = 78.20
Arvore max_depth = 8, min_samples_leaf = 256, treino = 78.89, teste = 77.94
Arvore max_depth = 8, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 9, min_samples_leaf = 128, treino = 79.12, teste = 78.30
Arvore max_depth = 9, min_samples_leaf = 192, treino = 78.99, teste = 78.11
Arvore max_depth = 9, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 9, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 10, min_samples_leaf = 128, treino = 79.16, teste = 78.29
Arvore max_depth = 10, min_samples_leaf = 192, treino = 79.01, teste = 78.05
Arvore max_depth = 10, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 10, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 11, min_samples_leaf = 128, treino = 79.19, teste = 78.31
Arvore max_depth = 11, min_samples_leaf = 192, treino = 79.01, teste = 78.05
Arvore max_depth = 11, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 11, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 12, min_samples_leaf = 128, treino = 79.21, teste = 78.22
Arvore max_depth = 12, min_samples_leaf = 192, treino = 79.01, teste = 78.05
Arvore max_depth = 12, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 12, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 13, min_samples_leaf = 128, treino = 79.21, teste = 78.22
Arvore max_depth = 13, min_samples_leaf = 192, treino = 79.01, teste = 78.05
Arvore max_depth = 13, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 13, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 14, min_samples_leaf = 128, treino = 79.21, teste = 78.22
Arvore max_depth = 14, min_samples_leaf = 192, treino = 79.01, teste = 78.05
Arvore max_depth = 14, min_samples_leaf = 256, treino = 78.90, teste = 77.85
Arvore max_depth = 14, min_samples_leaf = 512, treino = 77.53, teste = 77.50
Arvore max_depth = 15, min_samples_leaf = 128, treino = 79.21, teste = 78.22
```

```
[ ] corr = resultados.corr()
```

```
[ ] sns.set(style="white")

    # Generate a mask for the upper triangle
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    # Set up the matplotlib figure
    f, ax = plt.subplots(figsize=(11, 9))

    # Generate a custom diverging colormap
    cmap = sns.diverging_palette(220, 10, as_cmap=True)

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
                square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f4eae1ae9b0>
```




```
[ ] resultados.sort_values("test", ascending=False).head()
```

	max_depth	min_samples_leaf	train	test
13	4	192	78.750993	78.672633
8	3	128	78.750993	78.672633
9	3	192	78.750993	78.672633
10	3	256	78.750993	78.672633
14	4	256	78.750993	78.672633

Explorando 3 dimensões de hiper parâmetros

```
[ ] def roda_arvore_de_decisao(max_depth, min_samples_leaf, min_samples_split):
      SEED = 301
      np.random.seed(SEED)

      cv = GroupKFold(n_splits = 10)
      modelo = DecisionTreeClassifier(max_depth=max_depth, min_samples_leaf = min_samples_leaf, min_samples_split = min_samples_split)
      results = cross_validate(modelo, x_azar, y_azar, cv = cv, groups = dados.modelo, return_train_score=True)
      fit_time = results['fit_time'].mean()
      score_time = results['score_time'].mean()
      train_score = results['train_score'].mean() * 100
      test_score = results['test_score'].mean() * 100
      tabela = [max_depth, min_samples_leaf, min_samples_split, train_score, test_score, fit_time, score_time]
      return tabela

    def busca():
      resultados = []
      for max_depth in range(1,33):
        for min_samples_leaf in [32, 64, 128, 256]:
          for min_samples_split in [32, 64, 128, 256]:
            tabela = roda_arvore_de_decisao(max_depth, min_samples_leaf, min_samples_split)
            resultados.append(tabela)
      resultados = pd.DataFrame(resultados, columns= ["max_depth","min_samples_leaf","min_samples_split","train","test", "fit_time", "score_time"])
      return resultados

    resultados = busca()
    resultados.head()
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-32-69e067b2978a> in <module>()
     23   return resultados
     24
---> 25 resultados = busca()
     26 resultados.head()
     27

 18 frames
/usr/local/lib/python3.6/dist-packages/numpy/lib/arraysetops.py in _unique1d(ar, return_index, return_inverse, return_counts)
    281         aux = ar[perm]
    282     else:
--> 283         ar.sort()
    284         aux = ar
    285     flag = np.concatenate(([True], aux[1:] != aux[:-1]))

KeyboardInterrupt:
```

PESQUISAR NO STACK OVERFLOW

```
[ ] corr = resultados.corr()
```

```
[ ] sns.set(style="white")

    # Generate a mask for the upper triangle
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True

    # Set up the matplotlib figure
    f, ax = plt.subplots(figsize=(11, 9))

    # Generate a custom diverging colormap
    cmap = sns.diverging_palette(220, 10, as_cmap=True)

    # Draw the heatmap with the mask and correct aspect ratio
    sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3, center=0,
                square=True, linewidths=.5, cbar_kws={"shrink": .5})
```

```
[ ] resultados.sort_values("test", ascending=False).head()
```

Explorando espaço de hiper parâmetros com GridSearchCV

```
[ ] from sklearn.model_selection import GridSearchCV

    SEED=301
    np.random.seed(SEED)

    espaco_de_parametros = {
        "max_depth" : [3, 5],
        "min_samples_split" : [32, 64, 128],
        "min_samples_leaf" : [32, 64, 128],
        "criterion" : ["gini", "entropy"]
    }

    busca = GridSearchCV(DecisionTreeClassifier(),
```

```
                    espaco_de_parametros,
                    cv = GroupKFold(n_splits = 10))
busca.fit(x_azar, y_azar, groups = dados.modelo)
resultados = pd.DataFrame(busca.cv_results_)
resultados.head()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('mean_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split0_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split1_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split2_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split3_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split4_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split5_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split6_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split7_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split8_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split9_train_score'), which will not be available by default any more in 0.21. If you need training scores,
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('std_train_score'), which will not be available by default any more in 0.21. If you need training scores, ple
  warnings.warn(*warn_args, **warn_kwargs)
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_criterion	param_max_depth	param_min_samples_leaf	param_min_samples_split	params	rank_test_score	...	split7_test_score	split7_train_score	split8_t
0	0.011155	0.001260	0.7868	0.78751	gini	3	32	32	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.781818	0.788124	
1	0.010868	0.001203	0.7868	0.78751	gini	3	32	64	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.781818	0.788124	
2	0.010933	0.001298	0.7868	0.78751	gini	3	32	128	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.781818	0.788124	
3	0.010633	0.001206	0.7868	0.78751	gini	3	64	32	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.781818	0.788124	
4	0.010424	0.001201	0.7868	0.78751	gini	3	64	64	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.781818	0.788124	

5 rows × 34 columns

```
print(busca.best_params_)
print(busca.best_score_ * 100)
```

```
{'criterion': 'gini', 'max_depth': 3, 'min_samples_leaf': 32, 'min_samples_split': 32}
78.68
```

```
melhor = busca.best_estimator_
melhor
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=32, min_samples_split=32,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
from sklearn.metrics import accuracy_score

# evitar essa abordagem pois estará sendo otimista

predicoes = melhor.predict(x_azar)
accuracy = accuracy_score(predicoes, y_azar) * 100
print("Accuracy para os dados foi %.2f%%" % accuracy)
```

```
Accuracy para os dados foi 78.75%
```

Como ter uma estimativa sem esse vício nos dados que eu já vi?

No caso de cross validation com busca de hiper parâmetros, fazemos uma nova validação cruzada. Chama-se nested cross validation

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(busca, x_azar, y_azar, cv = GroupKFold(n_splits=10), groups = dados.modelo)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_validation.py:542: FutureWarning: From version 0.22, errors during fit will result in a cross validation score of NaN by default. Use error_score='raise' if you want an ex
  FutureWarning)
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-38-5d0a4e0c67ad> in <module>()
      1 from sklearn.model_selection import cross_val_score
      2
----> 3 scores = cross_val_score(busca, x_azar, y_azar, cv = GroupKFold(n_splits=10), groups = dados.modelo)

                                  16 frames
/usr/local/lib/python3.6/dist-packages/sklearn/model_selection/_split.py in _iter_test_indices(self, X, y, groups)
    502     def _iter_test_indices(self, X, y, groups):
    503         if groups is None:
--> 504             raise ValueError("The 'groups' parameter should not be None.")
    505         groups = check_array(groups, ensure_2d=False, dtype=None)
    506

ValueError: The 'groups' parameter should not be None.
```

PESQUISAR NO STACK OVERFLOW

Infelizmente como o Pandas não suporta nested validation com group k fold não conseguimos prever o resultado para novos grupos

```
from sklearn.model_selection import GridSearchCV, KFold

SEED=301
np.random.seed(SEED)

espaco_de_parametros = {
    "max_depth" : [3, 5],
```

```
    "min_samples_split" : [32, 64, 128],
    "min_samples_leaf" : [32, 64, 128],
    "criterion" : ["gini", "entropy"]
}

busca = GridSearchCV(DecisionTreeClassifier(),
                     espaco_de_parametros,
                     cv = KFold(n_splits = 5, shuffle=True))
busca.fit(x_azar, y_azar)
resultados = pd.DataFrame(busca.cv_results_)
resultados.head()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('mean_train_score'), which will not be available by default any more in 0.21. If you need training scores, plea
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split0_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split1_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split2_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split3_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split4_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('std_train_score'), which will not be available by default any more in 0.21. If you need training scores, pleas
  warnings.warn(*warn_args, **warn_kwargs)
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_criterion	param_max_depth	param_min_samples_leaf	param_min_samples_split	params	rank_test_score	...	split2_test_score	split2_train_score	split3_tes
0	0.010018	0.001712	0.787	0.787525	gini	3	32	32	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.8025	0.783625	
1	0.009543	0.001429	0.787	0.787525	gini	3	32	64	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.8025	0.783625	
2	0.009719	0.001431	0.787	0.787525	gini	3	32	128	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.8025	0.783625	
3	0.009586	0.001437	0.787	0.787525	gini	3	64	32	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.8025	0.783625	
4	0.009464	0.001390	0.787	0.787525	gini	3	64	64	{'criterion': 'gini', 'max_depth': 3, 'min_sam...	1	...	0.8025	0.783625	

5 rows × 24 columns

```
from sklearn.model_selection import cross_val_score

scores = cross_val_score(busca, x_azar, y_azar, cv = KFold(n_splits=5, shuffle=True))
scores
```

```
array([0.7895, 0.7825, 0.7905, 0.7715, 0.7995])
```

```
def imprime_score(scores):
  media = scores.mean() * 100
  desvio = scores.std() * 100
  print("Accuracy médio %.2f" % media)
  print("Intervalo [%.2f, %.2f]" % (media - 2 * desvio, media + 2 * desvio))
```

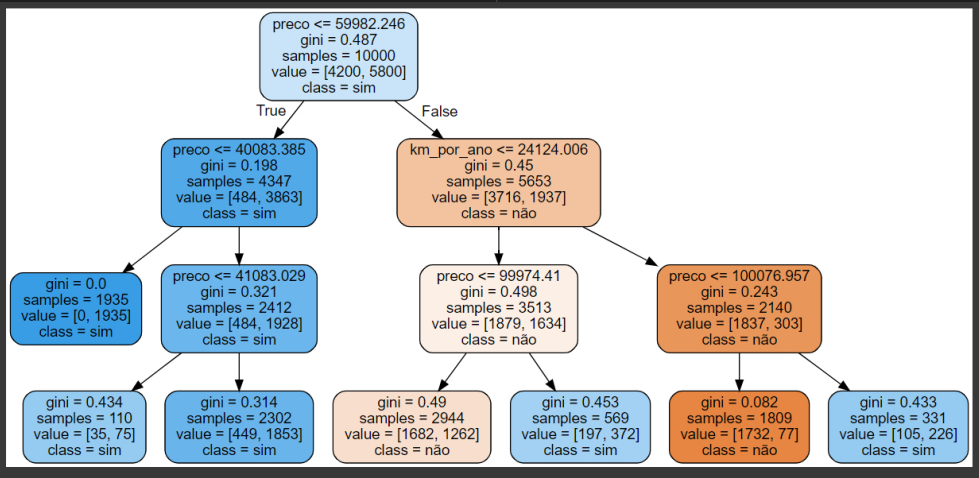
```
imprime_score(scores)
```

```
Accuracy médio 78.67
Intervalo [76.80, 80.54]
```

```
melhor = busca.best_estimator_
print(melhor)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=32, min_samples_split=32,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
from sklearn.tree import export_graphviz
import graphviz

features = x_azar.columns
dot_data = export_graphviz(melhor, out_file=None, filled=True, rounded=True,
                           class_names=["não","sim"],
                           feature_names = features)
graph = graphviz.Source(dot_data)
graph
```



Busca aleatória: RandomSearch

```
[ ] from sklearn.model_selection import RandomizedSearchCV

    SEED=301
    np.random.seed(SEED)

    espaco_de_parametros = {
        "max_depth" : [3, 5],
        "min_samples_split" : [32, 64, 128],
        "min_samples_leaf" : [32, 64, 128],
        "criterion" : ["gini", "entropy"]
    }

    busca = RandomizedSearchCV(DecisionTreeClassifier(),
                       espaco_de_parametros,
                       n_iter = 16,
                       cv = KFold(n_splits = 5, shuffle=True),
                       random_state = SEED)
    busca.fit(x_azar, y_azar)
    resultados = pd.DataFrame(busca.cv_results_)
    resultados.head()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('mean_train_score'), which will not be available by default any more in 0.21. If you need training scores, plea
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split0_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split1_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split2_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split3_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split4_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('std_train_score'), which will not be available by default any more in 0.21. If you need training scores, pleas
  warnings.warn(*warn_args, **warn_kwargs)
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_criterion	param_max_depth	param_min_samples_leaf	param_min_samples_split	params	rank_test_score	...	split2_test_score	split2_train_score	spl
0	0.013421	0.001535	0.7835	0.788050	gini	5	128	128	{'min_samples_split': 128, 'min_samples_leaf':...	13	...	0.7985	0.784750	
1	0.009877	0.001615	0.7870	0.787525	gini	3	32	64	{'min_samples_split': 64, 'min_samples_leaf': ...	1	...	0.8025	0.783625	
2	0.010312	0.001378	0.7870	0.787525	gini	3	128	64	{'min_samples_split': 64, 'min_samples_leaf': ...	1	...	0.8025	0.783625	
3	0.020480	0.001513	0.7839	0.788350	entropy	5	64	32	{'min_samples_split': 32, 'min_samples_leaf': ...	7	...	0.8005	0.785250	
4	0.020049	0.001597	0.7839	0.788350	entropy	5	64	64	{'min_samples_split': 64, 'min_samples_leaf': ...	7	...	0.8005	0.785250	

5 rows × 24 columns

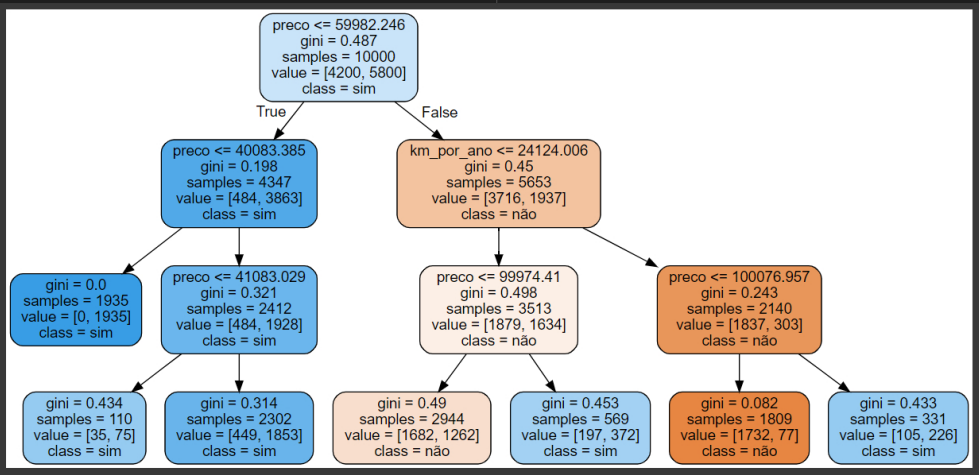
```
[ ] scores = cross_val_score(busca, x_azar, y_azar, cv = KFold(n_splits=5, shuffle=True))
    imprime_score(scores)
```

```
Accuracy médio 78.71
Intervalo [77.49, 79.93]
```

```
[ ] melhor = busca.best_estimator_
    print(melhor)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=32, min_samples_split=64,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
[ ] features = x_azar.columns
    dot_data = export_graphviz(melhor, out_file=None, filled=True, rounded=True,
                               class_names=["não","sim"],
                               feature_names = features)
    graph = graphviz.Source(dot_data)
    graph
```



Customizando o espaço de hiper parâmetros

```
[ ] from scipy.stats import randint

    SEED=301
    np.random.seed(SEED)

    espaco_de_parametros = {
        "max_depth" : [3, 5, 10, 15, 20, 30, None],
        "min_samples_split" : randint(32, 128),
        "min_samples_leaf" : randint(32, 128),
        "criterion" : ["gini", "entropy"]
    }
```

```
busca = RandomizedSearchCV(DecisionTreeClassifier(),
                           espaco_de_parametros,
                           n_iter = 16,
                           cv = KFold(n_splits = 5, shuffle=True),
                           random_state = SEED)
busca.fit(x_azar, y_azar)
resultados = pd.DataFrame(busca.cv_results_)
resultados.head()
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('mean_train_score'), which will not be available by default any more in 0.21. If you need training scores, plea
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split0_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split1_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split2_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split3_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split4_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('std_train_score'), which will not be available by default any more in 0.21. If you need training scores, pleas
  warnings.warn(*warn_args, **warn_kwargs)
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_criterion	param_max_depth	param_min_samples_leaf	param_min_samples_split	params	rank_test_score	...	split2_test_score	split2_train_score	split3_tes
0	0.016111	0.001663	0.7870	0.787525	entropy	3	71	100	{'criterion': 'entropy', 'max_depth': 3, 'min_...	1	...	0.8025	0.783625	
1	0.018661	0.001703	0.7765	0.794500	gini	15	93	111	{'criterion': 'gini', 'max_depth': 15, 'min_sa...	14	...	0.7830	0.789375	
2	0.017192	0.001705	0.7793	0.791575	gini	20	124	88	{'criterion': 'gini', 'max_depth': 20, 'min_sa...	8	...	0.7990	0.787750	
3	0.023533	0.001830	0.7685	0.802350	gini	None	46	62	{'criterion': 'gini', 'max_depth': None, 'min_...	16	...	0.7725	0.796875	
4	0.016981	0.001685	0.7794	0.791400	gini	15	126	84	{'criterion': 'gini', 'max_depth': 15, 'min_sa...	7	...	0.7995	0.787500	

5 rows × 24 columns

```
[ ] scores = cross_val_score(busca, x_azar, y_azar, cv = KFold(n_splits=5, shuffle=True))
    imprime_score(scores)
    melhor = busca.best_estimator_
    print(melhor)
```

```
Accuracy médio 78.71
Intervalo [77.49, 79.93]
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=71, min_samples_split=100,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
[ ] resultados_ordenados_pela_media = resultados.sort_values("mean_test_score", ascending=False)
    for indice, linha in resultados_ordenados_pela_media.iterrows():
      print("%.3f +-(%.3f) %s" % (linha.mean_test_score, linha.std_test_score*2, linha.params))
```

```
0.787 +-(0.019) {'criterion': 'entropy', 'max_depth': 3, 'min_samples_leaf': 71, 'min_samples_split': 100}
0.784 +-(0.024) {'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 73, 'min_samples_split': 72}
0.784 +-(0.024) {'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 64, 'min_samples_split': 67}
0.781 +-(0.017) {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 108, 'min_samples_split': 110}
0.780 +-(0.019) {'criterion': 'gini', 'max_depth': 10, 'min_samples_leaf': 125, 'min_samples_split': 59}
0.780 +-(0.012) {'criterion': 'gini', 'max_depth': 15, 'min_samples_leaf': 103, 'min_samples_split': 96}
0.779 +-(0.021) {'criterion': 'gini', 'max_depth': 15, 'min_samples_leaf': 126, 'min_samples_split': 84}
0.779 +-(0.020) {'criterion': 'gini', 'max_depth': 20, 'min_samples_leaf': 124, 'min_samples_split': 88}
0.779 +-(0.009) {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 101, 'min_samples_split': 52}
0.779 +-(0.014) {'criterion': 'entropy', 'max_depth': 15, 'min_samples_leaf': 104, 'min_samples_split': 88}
0.779 +-(0.010) {'criterion': 'gini', 'max_depth': 30, 'min_samples_leaf': 100, 'min_samples_split': 84}
0.778 +-(0.014) {'criterion': 'entropy', 'max_depth': 30, 'min_samples_leaf': 74, 'min_samples_split': 58}
0.777 +-(0.013) {'criterion': 'gini', 'max_depth': 30, 'min_samples_leaf': 88, 'min_samples_split': 78}
0.776 +-(0.009) {'criterion': 'gini', 'max_depth': 15, 'min_samples_leaf': 93, 'min_samples_split': 111}
0.775 +-(0.015) {'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 52, 'min_samples_split': 80}
0.768 +-(0.021) {'criterion': 'gini', 'max_depth': None, 'min_samples_leaf': 46, 'min_samples_split': 62}
```

Uma exploração mais a fundo de forma aleatória

```
[ ] from scipy.stats import randint

    SEED=564
    np.random.seed(SEED)

    espaco_de_parametros = {
        "max_depth" : [3, 5, 10, 15, 20, 30, None],
        "min_samples_split" : randint(32, 128),
        "min_samples_leaf" : randint(32, 128),
        "criterion" : ["gini", "entropy"]
    }

    busca = RandomizedSearchCV(DecisionTreeClassifier(),
                           espaco_de_parametros,
                           n_iter = 64,
                           cv = KFold(n_splits = 5, shuffle=True),
                           random_state = SEED)
    busca.fit(x_azar, y_azar)
    resultados = pd.DataFrame(busca.cv_results_)
    resultados.head()
```

```
[ ] resultados_ordenados_pela_media = resultados.sort_values("mean_test_score", ascending=False)
    for indice, linha in resultados_ordenados_pela_media.iterrows():
      print("%.3f +-(%.3f) %s" % (linha.mean_test_score, linha.std_test_score*2, linha.params))
```

```
[ ] scores = cross_val_score(busca, x_azar, y_azar, cv = KFold(n_splits=5, shuffle=True))
    imprime_score(scores)
    melhor = busca.best_estimator_
    print(melhor)
```

Comparando GridSearchCV com RandomizedSearch (1 comparação)

```
[ ] from sklearn.ensemble import RandomForestClassifier
    import time
```

```
SEED=301
np.random.seed(SEED)

espaco_de_parametros = {
    "n_estimators" : [10, 100],
    "max_depth" : [3, 5],
    "min_samples_split" : [32, 64, 128],
    "min_samples_leaf" : [32, 64, 128],
    "bootstrap" : [True, False],
    "criterion" : ["gini", "entropy"]
}

tic = time.time()
busca = GridSearchCV(RandomForestClassifier(),
                     espaco_de_parametros,
                     cv = KFold(n_splits = 5, shuffle=True))
busca.fit(x_azar, y_azar)
tac = time.time()
tempo_que_passou = tac - tic
print("Tempo %.2f segundos" % tempo_que_passou)


resultados = pd.DataFrame(busca.cv_results_)
resultados.head()
```

```
[ ] resultados_ordenados_pela_media = resultados.sort_values("mean_test_score", ascending=False)
    for indice, linha in resultados_ordenados_pela_media[:5].iterrows():
      print("%.3f +-(%.3f) %s" % (linha.mean_test_score, linha.std_test_score*2, linha.params))
```

```
[ ] tic = time.time()
    scores = cross_val_score(busca, x_azar, y_azar, cv = KFold(n_splits=5, shuffle=True))
    tac = time.time()
    tempo_passado = tac - tic
    print("Tempo %.2f segundos" % tempo_passado)

    imprime_score(scores)
    melhor = busca.best_estimator_
    print(melhor)
```

```
[ ] SEED=301
    np.random.seed(SEED)

    espaco_de_parametros = {
        "n_estimators" : [10, 100],
        "max_depth" : [3, 5],
        "min_samples_split" : [32, 64, 128],
        "min_samples_leaf" : [32, 64, 128],
        "bootstrap" : [True, False],
        "criterion" : ["gini", "entropy"]
    }

    tic = time.time()
    busca = RandomizedSearchCV(RandomForestClassifier(),
                         espaco_de_parametros,
                         n_iter = 20,
                         cv = KFold(n_splits = 5, shuffle=True))
    busca.fit(x_azar, y_azar)
    tac = time.time()
    tempo_que_passou = tac - tic
    print("Tempo %.2f segundos" % tempo_que_passou)


    resultados = pd.DataFrame(busca.cv_results_)
    resultados.head()
```

```
[ ] resultados_ordenados_pela_media = resultados.sort_values("mean_test_score", ascending=False)
    for indice, linha in resultados_ordenados_pela_media[:5].iterrows():
      print("%.3f +-(%.3f) %s" % (linha.mean_test_score, linha.std_test_score*2, linha.params))
```

```
[ ] tic = time.time()
    scores = cross_val_score(busca, x_azar, y_azar, cv = KFold(n_splits=5, shuffle=True))
    tac = time.time()
    tempo_passado = tac - tic
    print("Tempo %.2f segundos" % tempo_passado)

    imprime_score(scores)
    melhor = busca.best_estimator_
    print(melhor)
```

```
[ ] SEED=301
    np.random.seed(SEED)

    espaco_de_parametros = {
        "n_estimators" :randint(10, 101),
        "max_depth" : randint(3, 6),
        "min_samples_split" : randint(32, 129),
        "min_samples_leaf" : randint(32, 129),
        "bootstrap" : [True, False],
        "criterion" : ["gini", "entropy"]
    }

    tic = time.time()
    busca = RandomizedSearchCV(RandomForestClassifier(),
                         espaco_de_parametros,
                         n_iter = 80,
                         cv = KFold(n_splits = 5, shuffle=True))
    busca.fit(x_azar, y_azar)
    tac = time.time()
    tempo_que_passou = tac - tic
    print("Tempo %.2f segundos" % tempo_que_passou)


    resultados = pd.DataFrame(busca.cv_results_)
    resultados.head()
```

```
Tempo 124.18 segundos
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('mean_train_score'), which will not be available by default any more in 0.21. If you need training scores, plea
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split0_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split1_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split2_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split3_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split4_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('std_train_score'), which will not be available by default any more in 0.21. If you need training scores, pleas
  warnings.warn(*warn_args, **warn_kwargs)
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_bootstrap	param_criterion	param_max_depth	param_min_samples_leaf	param_min_samples_split	param_n_estimators	...	split2_test_score	split2_train_score	spl
0	0.392737	0.016654	0.7704	0.773900	False	gini	3	50	93	89	...	0.7750	0.761500	
1	0.271048	0.015260	0.7729	0.774950	True	gini	3	88	104	80	...	0.7925	0.771500	
2	0.254733	0.014860	0.7735	0.775875	True	gini	3	126	84	77	...	0.7925	0.771500	

3	0.176373	0.010644	0.7733	0.774700	True	gini	3	57	73	50	...	0.7830	0.764125
4	0.266327	0.014182	0.7754	0.776500	True	gini	4	52	88	68	...	0.7930	0.771625

5 rows × 26 columns

```
[ ] resultados_ordenados_pela_media = resultados.sort_values("mean_test_score", ascending=False)
    for indice, linha in resultados_ordenados_pela_media[:5].iterrows():
      print("%.3f +-(%.3f) %s" % (linha.mean_test_score, linha.std_test_score*2, linha.params))
```

```
0.779 +-(0.025) {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 84, 'min_samples_split': 89, 'n_estimators': 48}
0.778 +-(0.031) {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 5, 'min_samples_leaf': 32, 'min_samples_split': 96, 'n_estimators': 18}
0.778 +-(0.032) {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 4, 'min_samples_leaf': 121, 'min_samples_split': 47, 'n_estimators': 27}
0.777 +-(0.024) {'bootstrap': False, 'criterion': 'gini', 'max_depth': 4, 'min_samples_leaf': 96, 'min_samples_split': 98, 'n_estimators': 11}
0.777 +-(0.029) {'bootstrap': True, 'criterion': 'gini', 'max_depth': 5, 'min_samples_leaf': 63, 'min_samples_split': 88, 'n_estimators': 69}
```

Se eu não posso ou não consigo usar cross validation

```
[ ] # 0.6 treino     => treino
    # 0.2 teste      => dev teste
    # 0.2 validacao  => validacao

    from sklearn.model_selection import train_test_split

    SEED=301
    np.random.seed(SEED)

    x_treino_teste, x_validacao, y_treino_teste, y_validacao = train_test_split(x_azar, y_azar, test_size=0.2, shuffle=True, stratify=y_azar)
    print(x_treino_teste.shape)
    print(x_validacao.shape)
    print(y_treino_teste.shape)
    print(y_validacao.shape)
```

```
(8000, 3)
(2000, 3)
(8000,)
(2000,)
```

```
[ ] from sklearn.model_selection import StratifiedShuffleSplit

    espaco_de_parametros = {
        "n_estimators" :randint(10, 101),
        "max_depth" : randint(3, 6),
        "min_samples_split" : randint(32, 129),
        "min_samples_leaf" : randint(32, 129),
        "bootstrap" : [True, False],
        "criterion" : ["gini", "entropy"]
    }

    split = StratifiedShuffleSplit(n_splits = 1, test_size = 0.25)

    tic = time.time()
    busca = RandomizedSearchCV(RandomForestClassifier(),
                       espaco_de_parametros,
                       n_iter = 5,
                       cv = split)
    busca.fit(x_treino_teste, y_treino_teste)
    tac = time.time()
    tempo_que_passou = tac - tic
    print("Tempo %.2f segundos" % tempo_que_passou)


    resultados = pd.DataFrame(busca.cv_results_)
    resultados.head()
```

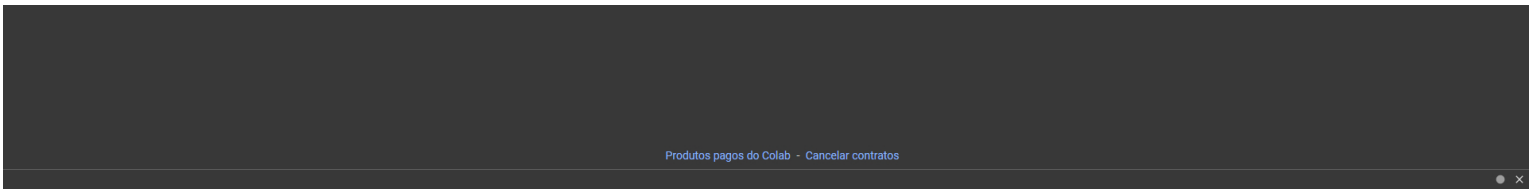
```
Tempo 2.38 segundos
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('mean_train_score'), which will not be available by default any more in 0.21. If you need training scores, plea
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('split0_train_score'), which will not be available by default any more in 0.21. If you need training scores, pl
  warnings.warn(*warn_args, **warn_kwargs)
/usr/local/lib/python3.6/dist-packages/sklearn/utils/deprecation.py:125: FutureWarning: You are accessing a training score ('std_train_score'), which will not be available by default any more in 0.21. If you need training scores, pleas
  warnings.warn(*warn_args, **warn_kwargs)
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_bootstrap	param_criterion	param_max_depth	param_min_samples_leaf	param_min_samples_split	param_n_estimators	params	rank_test_score	split0_test_score
0	0.253041	0.012972	0.7780	0.776000	True	entropy	5	82	111	53	{'bootstrap': True, 'criterion': 'entropy', 'm...	4	0.7780
1	0.391569	0.020130	0.7820	0.777667	False	gini	5	127	71	85	{'bootstrap': False, 'criterion': 'gini', 'max...	1	0.7820
2	0.259195	0.017875	0.7785	0.775833	True	gini	4	104	83	80	{'bootstrap': True, 'criterion': 'gini', 'max_...	3	0.7785
3	0.495695	0.018553	0.7775	0.775833	False	entropy	4	37	94	86	{'bootstrap': False, 'criterion': 'entropy', '...	5	0.7775
4	0.180420	0.012332	0.7805	0.776000	True	gini	5	85	125	50	{'bootstrap': True, 'criterion': 'gini', 'max_...	2	0.7805

```
[ ] tic = time.time()
    scores = cross_val_score(busca, x_validacao, y_validacao, cv = split)
    tac = time.time()
    tempo_passado = tac - tic
    print("Tempo %.2f segundos" % tempo_passado)
    scores
```

```
Tempo 0.57 segundos
array([0.77])
```

```
[ ] Comece a programar ou gere código com IA.
```


Produtos pagos do Colab - Cancelar contratos