Aluno: João Emanuel da Silva Lins

Matrícula: 162080263

Arvores de Decisão - Diabetes

Disponível em https://www.kaggle.com/uciml/pima-indians-diabetes-database (https://www.kaggle.com/uciml/pima-indians-diabetes-database)

Attributes:

Pregnancies: Number of times pregnant - Gravidez

Glucose: Plasma glucose concentration a 2 hours in an oral glucose tolerance test - Glicose

BloodPressure: Diastolic blood pressure (mm Hg) - Pressão Arterial

SkinThickness: Triceps skin fold thickness (mm) - Espessura do tríceps

Insulin: 2-Hour serum insulin (mu U/ml) - Insulina

BMI: Body mass index (weight in kg/(height in m)^2) - IMC

DiabetesPedigreeFunction: Diabetes pedigree function - Função que leva em conta doenças na familia

Age: Age (years)

Outcome: Class variable (0 or 1) - 0 : Não tem Diabetes, 1: Possui Diabetes

In [1]:

```
import pandas as pd
```

In [2]:

```
df = pd.read_csv("diabetes.csv")
df.rename(columns={"Outcome": "Class"} , inplace=True)
df.head()
```

Out[2]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

In [3]:

```
df.describe().T
```

Out[3]:

	count	mean	std	min	25%	50%	7
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00(
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25(
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00(
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00(
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25(
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60(
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62(
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00(
Class	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00(

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Class                     768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Preparação dos dados

limpeza dos dados missing

In [5]:

```
len(df)
```

Out[5]:

768

In [6]:

```
df2 = df
df2 = df2.dropna()
len(df2)
```

Out[6]:

768

In [7]:

```
df2.head()
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0
3	1	89	66	23	94	28.1	0
4	0	137	40	35	168	43.1	2

Aplicar o algoritmo de Classificação - Árvore de Decisão

In [8]:

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
```

In [9]:

```
# particionar os conjuntos de treino e teste
from sklearn.model_selection import train_test_split

diabetes_data = df2.loc[:,["Pregnancies", "Glucose", "BloodPressure", "SkinThick
ness", "Insulin",
                           "BMI", "DiabetesPedigreeFunction", "Age"]]
diabetes_target = df2["Class"]
```

In [10]:

```
diabetes_data[:3]
```

Out[10]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunc
0	6	148	72	35	0	33.6	0
1	1	85	66	29	0	26.6	0
2	8	183	64	0	0	23.3	0

In [11]:

```
diabetes_target[:3]
```

Out[11]:

```
0    1
1    0
2    1
Name: Class, dtype: int64
```

In [12]:

```
X_train, X_test, y_train, y_test = train_test_split(
    diabetes_data, diabetes_target, test_size=0.33, random_state=42)

X_train[:3]
```

Out[12]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
464	10	115	98	0	0	24.0	
223	7	142	60	33	190	28.8	
393	4	116	72	12	87	22.1	

In [13]:

```
print("# dados de treino = ", len(X_train))
print("# dados de teste = ", len(X_test))
```

```
# dados de treino =  514
# dados de teste =  254
```

aplicar o algoritmo de arvores de decisao

In [14]:

```
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.709
```

In [15]:

```
import sklearn.metrics as metrics

metrics.confusion_matrix(y_test, tree.predict(X_test))
```

Out[15]:

```
array([[127,  41],
       [ 33,  53]], dtype=int64)
```

Previsao

In [16]:

```
import numpy as np
```

In [17]:

```
# 0 = não tem diabetes, 1 = tem diabetes

ocorrencias = [
#Pregnancies Glucose BloodPressure    SkinThickness Insulin    BMI    DiabetesPe
digreeFunction  Age
[   3,          150,         75,             36,        0,     36.1,    0.62,
55  ],
[   0,          90,          90,             40,        90,    30,      0.7,
32  ],
[   1,          120,         75,             28,        70,    29,      0.5,
27  ]
]


saida =  '{:03.1f}\t\t{:03.1f}\t{:03.1f}\t\t{:03.1f}\t\t{:03.1f}\t{:03.1f}\t\t{:
03.1f}\t{:03.1f}\t{:s}'

print("Pregnancies \tGlucose\tBloodPressure\tSkinThickness Insulin    BMI    Dia
betesPFun    Age")
for ocorrencia in ocorrencias:
    ocorrencia = np.array(ocorrencia).reshape(1, -1)
    classe = "Não tem diabetes" if tree.predict(ocorrencia) == 0 else "Possui Di
abetes"
    #print(classe)
    print(saida.format(ocorrencia[0][0], ocorrencia[0][1], ocorrencia[0][2], oco
rrencia[0][3],
                       ocorrencia[0][4], ocorrencia[0][5], ocorrencia[0][6], oco
rrencia[0][7], classe))
```

```
Pregnancies     Glucose BloodPressure   SkinThickness Insulin    BMI
DiabetesPFun    Age
3.0             150.0   75.0            36.0          0.0     36.1
0.6     55.0    Possui Diabetes
0.0             90.0    90.0            40.0          90.0    30.0
0.7     32.0    Não tem diabetes
1.0             120.0   75.0            28.0          70.0    29.0
0.5     27.0    Não tem diabetes
```

Verificando os atributos mais relevantes

In [18]:

```
print(list(df.columns[:-1]))
tree.feature_importances_
```

```
['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n', 'BMI', 'DiabetesPedigreeFunction', 'Age']
```

Out[18]:

```
array([0.03669111, 0.39394111, 0.10523598, 0.04884907, 0.09388844,
       0.16631872, 0.06580802, 0.08926756])
```

In [19]:

```
dfi = pd.DataFrame()
dfi['atributo'] = list(df.columns[:-1])
dfi['importancia'] = tree.feature_importances_
dfi
```

Out[19]:

	atributo	importancia
0	Pregnancies	0.036691
1	Glucose	0.393941
2	BloodPressure	0.105236
3	SkinThickness	0.048849
4	Insulin	0.093888
5	BMI	0.166319
6	DiabetesPedigreeFunction	0.065808
7	Age	0.089268

In [20]:

```
dfi.sort_values(by="importancia", ascending=False)
```

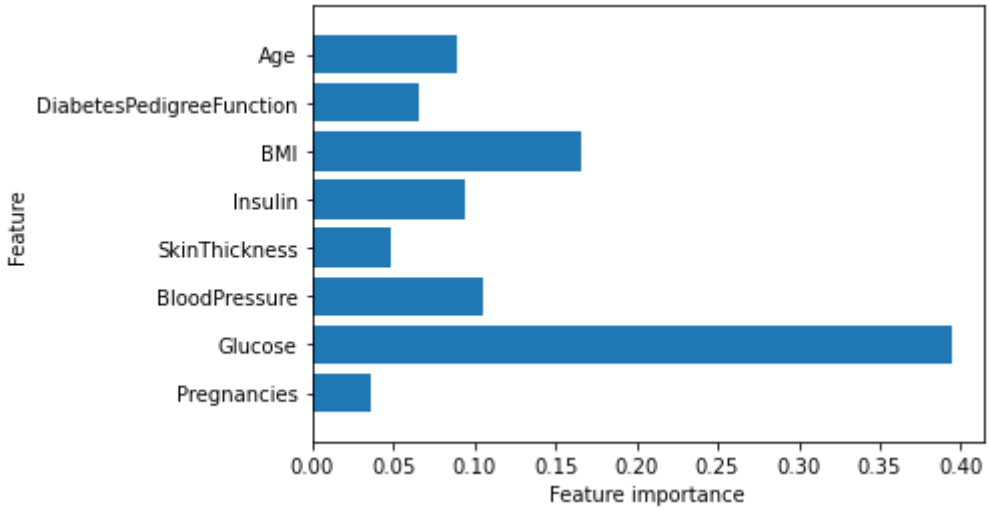
Out[20]:

	atributo	importancia
1	Glucose	0.393941
5	BMI	0.166319
2	BloodPressure	0.105236
4	Insulin	0.093888
7	Age	0.089268
6	DiabetesPedigreeFunction	0.065808
3	SkinThickness	0.048849
0	Pregnancies	0.036691

In [21]:

```
# mostrar os atributos mais relavantes (features)
import matplotlib.pyplot as plt
%matplotlib inline

def plot_feature_importances_cancer(model):
    columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insu
lin",
                       "BMI", "DiabetesPedigreeFunction", "Age"]
    n_features = len(columns)
    plt.barh(range(n_features), tree.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), columns)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)
    plt.figure(figsize=(12,10))
    print ("Atributos mais relavantes")
    plt.show()


plot_feature_importances_cancer(tree)
```

```
Atributos mais relavantes
```




```
<Figure size 864x720 with 0 Axes>
```

In [22]:

```
columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insulin"
,
                        "BMI", "DiabetesPedigreeFunction", "Age"]
dict_features =  {}
for name, feature, in zip(columns,  tree.feature_importances_):
    dict_features[name] = feature
dict_features
```

Out[22]:

```
{'Pregnancies': 0.036691112268639484,
 'Glucose': 0.39394110999442356,
 'BloodPressure': 0.10523598192982168,
 'SkinThickness': 0.04884906980811649,
 'Insulin': 0.0938884391553898,
 'BMI': 0.16631871513760388,
 'DiabetesPedigreeFunction': 0.06580801646616855,
 'Age': 0.0892675552398366}
```

In [23]:

```
# ordenar
sorted(dict_features.items(), key=lambda x: -x[1])
```

Out[23]:

```
[('Glucose', 0.39394110999442356),
 ('BMI', 0.16631871513760388),
 ('BloodPressure', 0.10523598192982168),
 ('Insulin', 0.0938884391553898),
 ('Age', 0.0892675552398366),
 ('DiabetesPedigreeFunction', 0.06580801646616855),
 ('SkinThickness', 0.04884906980811649),
 ('Pregnancies', 0.036691112268639484)]
```

Analisando a árvore de decisao

In [24]:

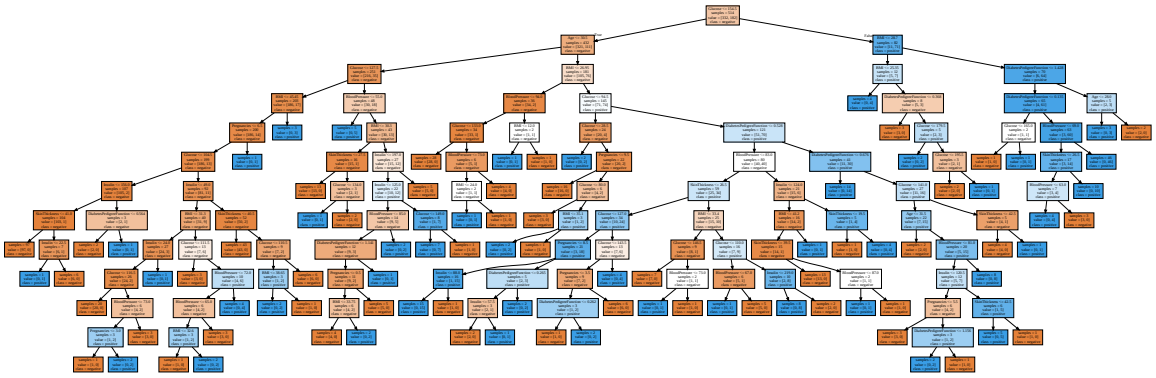
```
#!pip install graphviz
```

In [25]:

```
# Class: negative, positive
Class = ["negative", "positive"]
features = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insuli
n",
                        "BMI", "DiabetesPedigreeFunction", "Age"]
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["negative","positive",],
                feature_names=features, impurity=False, filled=True)
```

In [28]:

```
# instalar o graphviz: https://anaconda.org/anaconda/graphviz
# http://www.graphviz.org/Download_macos.php
# !pip install graphviz

import graphviz
from IPython.display import set_matplotlib_formats, display

with open("tree.dot") as f:
    dot_graph = f.read()
display(graphviz.Source(dot_graph))
```



Exercício: Retirar os 2 atributos menos significativos e mostrar as métricas : "SkinThickness" e "Pregnancies"

In [30]:

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
```

In [31]:

```
# particionar os conjuntos de treino e teste
from sklearn.model_selection import train_test_split

diabetes_data = df2.loc[:,["Glucose", "BloodPressure", "Insulin",
                           "BMI", "DiabetesPedigreeFunction", "Age"]]
diabetes_target = df2["Class"]
```

In [32]:

```
diabetes_data[:3]
```

Out[32]:

	Glucose	BloodPressure	Insulin	BMI	DiabetesPedigreeFunction	Age
0	148	72	0	33.6	0.627	50
1	85	66	0	26.6	0.351	31
2	183	64	0	23.3	0.672	32

In [33]:

```
diabetes_target[:3]
```

Out[33]:

```
0    1
1    0
2    1
Name: Class, dtype: int64
```

In [34]:

```
X_train, X_test, y_train, y_test = train_test_split(
    diabetes_data, diabetes_target, test_size=0.33, random_state=42)

X_train[:3]
```

Out[34]:

	Glucose	BloodPressure	Insulin	BMI	DiabetesPedigreeFunction	Age
464	115	98	0	24.0	1.022	34
223	142	60	190	28.8	0.687	61
393	116	72	87	22.1	0.463	37

In [35]:

```
print("# dados de treino = ", len(X_train))
print("# dados de teste = ", len(X_test))
```

```
# dados de treino =  514
# dados de teste =  254
```

aplicar o algoritmo de arvores de decisao

In [36]:

```
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train, y_train)
print("Accuracy on training set: {:.3f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.3f}".format(tree.score(X_test, y_test)))
```

```
Accuracy on training set: 1.000
Accuracy on test set: 0.654
```

In [37]:

```
import sklearn.metrics as metrics

metrics.confusion_matrix(y_test, tree.predict(X_test))
```

Out[37]:

```
array([[119,  49],
       [ 39,  47]], dtype=int64)
```

Previsao

In [38]:

```
import numpy as np
```

In [43]:

```
# 0 = não tem diabetes, 1 = tem diabetes

ocorrencias = [
# Glucose BloodPressure        Insulin    BMI    DiabetesPedigreeFunction  Age
[   150,         75,              0,     36.1,   0.62,                       55
],
[    90,          90,            90,    30,     0.7,                        32
],
[   120,         75,              70,    29,     0.5,                        27  ]
]


saida =  '{:03.1f}\t{:03.1f}\t\t{:03.1f}\t\t{:03.1f}\t\t{:03.1f}\t{:03.1f}\t{:s}
'

print("Glucose\tBloodPressure\t Insulin    BMI    DiabetesPFun    Age")
for ocorrencia in ocorrencias:
    ocorrencia = np.array(ocorrencia).reshape(1, -1)
    classe = "Não tem diabetes" if tree.predict(ocorrencia) == 0 else "Possui Di
abetes"
    #print(classe)
    print(saida.format(ocorrencia[0][0], ocorrencia[0][1], ocorrencia[0][2], oco
rrencia[0][3],
                       ocorrencia[0][4], ocorrencia[0][5],  classe))
```

```
Glucose BloodPressure    Insulin    BMI    DiabetesPFun    Age
150.0   75.0            0.0             36.1            0.6     55.0
Possui Diabetes
90.0    90.0            90.0            30.0            0.7     32.0
Não tem diabetes
120.0   75.0            70.0            29.0            0.5     27.0
Não tem diabetes
```

Verificando os atributos mais relevantes

In [47]:

```
print(list(diabetes_data.columns[:-1]))
tree.feature_importances_
```

```
['Glucose', 'BloodPressure', 'Insulin', 'BMI', 'DiabetesPedigreeFunc
tion']
```

Out[47]:

```
array([0.3708443 , 0.08933981, 0.08197103, 0.18123307, 0.13047574,
       0.14613604])
```

In [49]:

```
dfi = pd.DataFrame()
dfi['atributo'] = list(diabetes_data.columns[:-1])
dfi['importancia'] = tree.feature_importances_
dfi
```

```
---------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-49-c58065ff843d> in <module>
      1 dfi = pd.DataFrame()
      2 dfi['atributo'] = list(diabetes_data.columns[:-1])
----> 3 dfi['importancia'] = tree.feature_importances_
      4 dfi

~\anaconda3\lib\site-packages\pandas\core\frame.py in __setitem__(se
lf, key, value)
   2936         else:
   2937             # set column
-> 2938             self._set_item(key, value)
   2939
   2940     def _setitem_slice(self, key, value):

~\anaconda3\lib\site-packages\pandas\core\frame.py in _set_item(sel
f, key, value)
   2998
   2999         self._ensure_valid_index(value)
-> 3000         value = self._sanitize_column(key, value)
   3001         NDFrame._set_item(self, key, value)
   3002

~\anaconda3\lib\site-packages\pandas\core\frame.py in _sanitize_colu
mn(self, key, value, broadcast)
   3634
   3635             # turn me into an ndarray
-> 3636             value = sanitize_index(value, self.index, copy=F
alse)
   3637             if not isinstance(value, (np.ndarray, Index)):
   3638                 if isinstance(value, list) and len(value) >
0:

~\anaconda3\lib\site-packages\pandas\core\internals\construction.py
 in sanitize_index(data, index, copy)
    609
    610     if len(data) != len(index):
--> 611         raise ValueError("Length of values does not match le
ngth of index")
    612
    613     if isinstance(data, ABCIndexClass) and not copy:

ValueError: Length of values does not match length of index
```

In [50]:

```
dfi.sort_values(by="importancia", ascending=False)
```

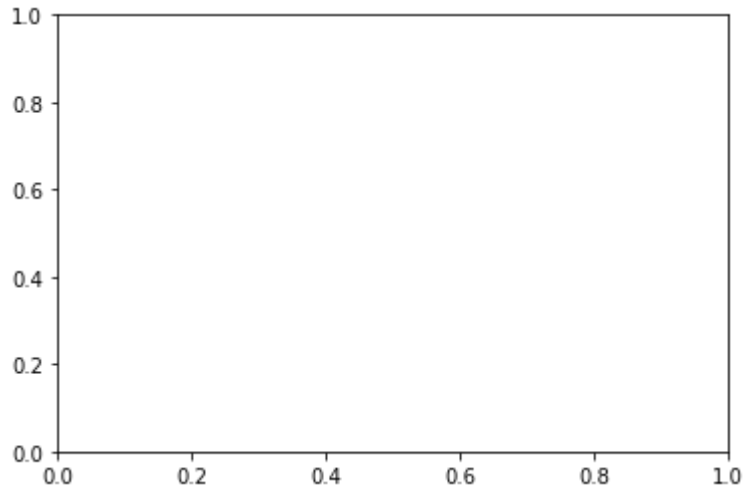
```
---------------------------------------------------------------------
-------
KeyError                                  Traceback (most recent cal
l last)
<ipython-input-50-7d9ea08c7472> in <module>
----> 1 dfi.sort_values(by="importancia", ascending=False)

~\anaconda3\lib\site-packages\pandas\core\frame.py in sort_values(se
lf, by, axis, ascending, inplace, kind, na_position, ignore_index)
   4925
   4926             by = by[0]
-> 4927             k = self._get_label_or_level_values(by, axis=axi
s)
   4928
   4929             if isinstance(ascending, (tuple, list)):

~\anaconda3\lib\site-packages\pandas\core\generic.py in _get_label_o
r_level_values(self, key, axis)
   1690             values = self.axes[axis].get_level_values(key)._
values
   1691         else:
-> 1692             raise KeyError(key)
   1693
   1694         # Check for duplicates

KeyError: 'importancia'
```

In [51]:

```
# mostrar os atributos mais relavantes (features)
import matplotlib.pyplot as plt
%matplotlib inline

def plot_feature_importances_cancer(model):
    columns = ["Pregnancies", "Glucose", "BloodPressure", "SkinThickness", "Insu
lin",
                       "BMI", "DiabetesPedigreeFunction", "Age"]
    n_features = len(columns)
    plt.barh(range(n_features), tree.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), columns)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)
    plt.figure(figsize=(12,10))
    print ("Atributos mais relavantes")
    plt.show()


plot_feature_importances_cancer(tree)
```

```
---------------------------------------------------------------------
-------
ValueError                                Traceback (most recent cal
l last)
<ipython-input-51-91f3ea91105d> in <module>
     17
     18
---> 19 plot_feature_importances_cancer(tree)

<ipython-input-51-91f3ea91105d> in plot_feature_importances_cancer(m
odel)
      7                         "BMI", "DiabetesPedigreeFunction", "A
ge"]
      8     n_features = len(columns)
----> 9     plt.barh(range(n_features), tree.feature_importances_, a
lign='center')
     10     plt.yticks(np.arange(n_features), columns)
     11     plt.xlabel("Feature importance")

~\anaconda3\lib\site-packages\matplotlib\pyplot.py in barh(y, width,
height, left, align, **kwargs)
   2420 @docstring.copy(Axes.barh)
   2421 def barh(y, width, height=0.8, left=None, *, align='center',
**kwargs):
-> 2422     return gca().barh(
   2423         y, width, height=height, left=left, align=align, **k
wargs)
   2424

~\anaconda3\lib\site-packages\matplotlib\axes\_axes.py in barh(self,
y, width, height, left, align, **kwargs)
   2544         """
   2545         kwargs.setdefault('orientation', 'horizontal')
-> 2546         patches = self.bar(x=left, height=height, width=widt
h, bottom=y,
   2547                            align=align, **kwargs)
   2548         return patches

~\anaconda3\lib\site-packages\matplotlib\__init__.py in inner(ax, da
ta, *args, **kwargs)
   1563     def inner(ax, *args, data=None, **kwargs):
   1564         if data is None:
-> 1565             return func(ax, *map(sanitize_sequence, args), *
*kwargs)
   1566
   1567         bound = new_sig.bind(ax, *args, **kwargs)

~\anaconda3\lib\site-packages\matplotlib\axes\_axes.py in bar(self,
 x, height, width, bottom, align, **kwargs)
   2338                 yerr = self._convert_dx(yerr, y0, y, self.co
nvert_yunits)
   2339
-> 2340         x, height, width, y, linewidth = np.broadcast_arrays
(
   2341             # Make args iterable too.
   2342             np.atleast_1d(x), height, width, y, linewidth)

<__array_function__ internals> in broadcast_arrays(*args, **kwargs)

~\anaconda3\lib\site-packages\numpy\lib\stride_tricks.py in broadcas
t_arrays(*args, **kwargs)
```

```
    262     args = [np.array(_m, copy=False, subok=subok) for _m in
 args]
    263
--> 264     shape = _broadcast_shape(*args)
    265
    266     if all(array.shape == shape for array in args):

~\anaconda3\lib\site-packages\numpy\lib\stride_tricks.py in _broadca
st_shape(*args)
    189     # use the old-iterator because np.nditer does not handle
size 0 arrays
    190     # consistently
--> 191     b = np.broadcast(*args[:32])
    192     # unfortunately, it cannot handle 32 or more arguments d
irectly
    193     for pos in range(32, len(args), 31):

ValueError: shape mismatch: objects cannot be broadcast to a single
 shape
```



In [56]:

```
columns = [ "Glucose", "BloodPressure", "Insulin",
                      "BMI", "DiabetesPedigreeFunction", "Age"]
dict_features =  {}
for name, feature, in zip(columns,  tree.feature_importances_):
    dict_features[name] = feature
dict_features
```

Out[56]:

```
{'Glucose': 0.37084430266000235,
 'BloodPressure': 0.08933980649950127,
 'Insulin': 0.08197103254530407,
 'BMI': 0.1812330742368067,
 'DiabetesPedigreeFunction': 0.1304757425885096,
 'Age': 0.14613604146987583}
```

In [57]:

```
# ordenar
sorted(dict_features.items(), key=lambda x: -x[1])
```

Out[57]:

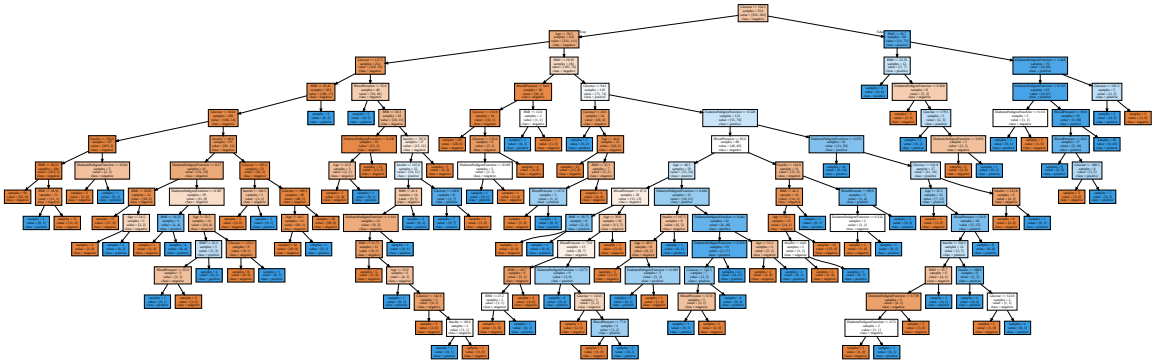
```
[('Glucose', 0.37084430266000235),
 ('BMI', 0.1812330742368067),
 ('Age', 0.14613604146987583),
 ('DiabetesPedigreeFunction', 0.1304757425885096),
 ('BloodPressure', 0.08933980649950127),
 ('Insulin', 0.08197103254530407)]
```

In [58]:

```
# Class: negative, positive
# Foram retirados os atributos "SkinThickness" e "Pregnancies", na qual são as m
enos significativas
# 3% e 4% respectivamente.

Class = ["negative", "positive"]
features = ["Glucose", "BloodPressure", "Insulin",
                       "BMI", "DiabetesPedigreeFunction", "Age"]
from sklearn.tree import export_graphviz
export_graphviz(tree, out_file="tree.dot", class_names=["negative","positive",],
                feature_names=features, impurity=False, filled=True)
```

In [59]:

```
# instalar o graphviz: https://anaconda.org/anaconda/graphviz
# http://www.graphviz.org/Download_macos.php
# !pip install graphviz

import graphviz
from IPython.display import set_matplotlib_formats, display

with open("tree.dot") as f:
    dot_graph = f.read()
display(graphviz.Source(dot_graph))
```



In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```
```