In [1]:
```
import pandas as pd
from scipy.io.arff import loadarff
data = loadarff('./arritmias.txt')
df = pd.DataFrame(data[0])
print("Setup complete.")
```

```
Setup complete.
```

1) A partir da base de dados de arritimias cardíacas:

a) Descreva a base de dados, descrevendo os atributos, númericos e categóricos, classificando segundo a escala (nominal ou razão) e a cardinalidade (discreta,contínua, binária).

Númericos e categóricos

In [2]:
```
df.dtypes.value_counts()
```

Out[2]:
```
float64    272
object       8
dtype: int64
```

In [3]:
```
numericos = df.select_dtypes(include='float64')
categoricos = df.select_dtypes(include='O')
```

Separando os atributos em listas

In [4]:
```
numericos
```

Out[4]:

	Age	Height	Weight	QRS_duration	P-R	Q-T	T_interval	P_interval	QRS	T	...	Amp_V6_1	Amp_
0	75.0	190.0	80.0	91.0	193.0	371.0	174.0	121.0	-16.0	13.0	...	-0.3	
1	56.0	165.0	64.0	81.0	174.0	401.0	149.0	39.0	25.0	37.0	...	-0.5	
2	54.0	172.0	95.0	138.0	163.0	386.0	185.0	102.0	96.0	34.0	...	0.9	
3	55.0	175.0	94.0	100.0	202.0	380.0	179.0	143.0	28.0	11.0	...	0.1	
4	75.0	190.0	80.0	88.0	181.0	360.0	177.0	103.0	-16.0	13.0	...	-0.4	
...	...	...	...	...	...	...	...	...	...	...	...	...	
447	53.0	160.0	70.0	80.0	199.0	382.0	154.0	117.0	-37.0	4.0	...	0.0	
448	37.0	190.0	85.0	100.0	137.0	361.0	201.0	73.0	86.0	66.0	...	-0.5	
449	36.0	166.0	68.0	108.0	176.0	365.0	194.0	116.0	-85.0	-19.0	...	1.2	
450	32.0	155.0	55.0	93.0	106.0	386.0	218.0	63.0	54.0	29.0	...	0.2	
451	78.0	160.0	70.0	79.0	127.0	364.0	138.0	78.0	28.0	79.0	...	-0.3	

452 rows × 272 columns

In [5]:
```
categoricos
```

Out[5]:

	Sex	Existence_ragged_R_wave	Existence_diphasic_derivation_R_wave	Existence_ragged_P_wave	Existence
0	b'0'	b'0'	b'0'	b'0'	
1	b'1'	b'0'	b'0'	b'0'	
2	b'0'	b'0'	b'0'	b'0'	
3	b'0'	b'0'	b'0'	b'0'	
4	b'0'	b'0'	b'0'	b'0'	
...	...	...	...	...	
447	b'1'	b'0'	b'0'	b'0'	
448	b'0'	b'0'	b'0'	b'0'	
449	b'0'	b'0'	b'0'	b'0'	
450	b'1'	b'0'	b'0'	b'0'	
451	b'1'	b'0'	b'0'	b'0'	

452 rows × 8 columns

Notamos que são 272 atributos númericos e 8 do tipo "O"

In [6]:
```
df.select_dtypes(include='O')
```

Out[6]:

	Sex	Existence_ragged_R_wave	Existence_diphasic_derivation_R_wave	Existence_ragged_P_wave	Existence
0	b'0'	b'0'	b'0'	b'0'	
1	b'1'	b'0'	b'0'	b'0'	
2	b'0'	b'0'	b'0'	b'0'	
3	b'0'	b'0'	b'0'	b'0'	
4	b'0'	b'0'	b'0'	b'0'	
...	...	...	...	...	
447	b'1'	b'0'	b'0'	b'0'	
448	b'0'	b'0'	b'0'	b'0'	
449	b'0'	b'0'	b'0'	b'0'	
450	b'1'	b'0'	b'0'	b'0'	
451	b'1'	b'0'	b'0'	b'0'	

452 rows × 8 columns

Os 8 atributos "O" são categóricos e estão salvos em "Bytes"

In [7]:
```
df.Class.value_counts()
```

Out[7]:
```
b'1'     245
b'10'     50
b'2'      44
b'6'      25
b'16'     22
b'3'      15
b'4'      15
b'5'      13
b'9'       9
b'15'      5
b'14'      4
b'7'       3
b'8'       2
Name: Class, dtype: int64
```

Dentre estes 8 atributos:

. 7 são binários
. 1 é a definição da condição diagnosticada, são 13 valores possiveis

Nominal ou Razão

Atributos com escala Nominal: São os categóricos

Atributos com escala de Razão: São os númericos com Zero Absoluto

Nominal

In [8]:
```
categoricos
```

Out[8]:

	Sex	Existence_ragged_R_wave	Existence_diphasic_derivation_R_wave	Existence_ragged_P_wave	Existence
0	b'0'	b'0'	b'0'	b'0'	
1	b'1'	b'0'	b'0'	b'0'	
2	b'0'	b'0'	b'0'	b'0'	
3	b'0'	b'0'	b'0'	b'0'	
4	b'0'	b'0'	b'0'	b'0'	
...	...	...	...	...	
447	b'1'	b'0'	b'0'	b'0'	
448	b'0'	b'0'	b'0'	b'0'	
449	b'0'	b'0'	b'0'	b'0'	
450	b'1'	b'0'	b'0'	b'0'	
451	b'1'	b'0'	b'0'	b'0'	

452 rows × 8 columns

Razão

In [9]:
```
mask = numericos >= 0

positive_columns = numericos.loc[:, mask.all()]
positive_columns
```

Out[9]:

	Age	Height	Weight	QRS_duration	P-R	Q-T	T_interval	P_interval	Q_wave	R_wave	...	Amp_V3_3
0	75.0	190.0	80.0	91.0	193.0	371.0	174.0	121.0	0.0	52.0	...	8.4
1	56.0	165.0	64.0	81.0	174.0	401.0	149.0	39.0	0.0	48.0	...	5.8
2	54.0	172.0	95.0	138.0	163.0	386.0	185.0	102.0	0.0	40.0	...	5.8
3	55.0	175.0	94.0	100.0	202.0	380.0	179.0	143.0	0.0	72.0	...	9.0
4	75.0	190.0	80.0	88.0	181.0	360.0	177.0	103.0	0.0	48.0	...	8.5
...	...	...	...	...	...	...	...	...	...	...	...	...
447	53.0	160.0	70.0	80.0	199.0	382.0	154.0	117.0	0.0	52.0	...	1.3
448	37.0	190.0	85.0	100.0	137.0	361.0	201.0	73.0	0.0	44.0	...	12.2
449	36.0	166.0	68.0	108.0	176.0	365.0	194.0	116.0	16.0	40.0	...	18.3
450	32.0	155.0	55.0	93.0	106.0	386.0	218.0	63.0	0.0	56.0	...	8.8
451	78.0	160.0	70.0	79.0	127.0	364.0	138.0	78.0	0.0	44.0	...	20.7

452 rows × 174 columns

Cardinalidade

Temos apenas 2 tipos de dados. Sabemos que, dos 8 dados "Object", 7 são binários(6 perguntas "Sim" ou "Não" e uma indicação de sexo) e 1 é Discreto, o diagnóstico do paciente, com 16 resultados possíveis.

Para lidarmos com os 272 "Floats" teremos que diferenciar entre os atributos que realmente são números reais e os atributos que, apesar de estarem salvos como floats, são inteiros(se um inteiro for salvo com um '.0', ao lermos, será interpertrado e considerado um 'float', por exemplo: 8.0)

In [10]:
```
# Primeiro, vamos separar as colunas numéricas
cols_nums = df.select_dtypes(include=['float'])
total_numericos = cols_nums.shape[1]
cols_nums.shape
```

Out[10]:
```
(452, 272)
```

In [11]:
```
list_cols_real = list() # Lista de colunas com valores reais

for col in cols_nums.columns: # Iteramos por cada coluna
    #print(cols_nums[col])
    #if not cols_nums[col] == int(cols_nums[col]):
        #print(cols_nums[col])
    for row in cols_nums[col]: # Para cada linha nessa coluna
        #print(row)
        try: # Se o valor analisado não for igual ao seu inteiro, temos um float
            if not row == int(row):
                #print(col," - ",row)
                if col not in list_cols_real:
                    list_cols_real.append(col)
        except ValueError:
            #print(col," - ",row)
            pass
```

In [12]:
```
print(len(list_cols_real))
```

```
116
```

Temos 116 valores que são contínuos

In [13]:
```
total_numericos-len(list_cols_real)
```

Out[13]:
```
156
```

Temos 156 valores que são discretos mais o valor do resultado do diagnóstico

Conclusão

157 atributos *Discretos*

116 atributos *Contínuos*

7 atributos *Binários*

```
b. Descreva cada um dos atributos segundo frequência,mínimo e máximo
valor, desvios padrão, conforme o caso
```

Atributos númericos

In [14]:
```
nums = df.select_dtypes(include=['float'])
```

In [15]:
```
nums.describe()
```

Out[15]:

	Age	Height	Weight	QRS_duration	P-R	Q-T	T_interval	P_interval	
count	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452.000000	452
mean	46.471239	166.188053	68.170354	88.920354	155.152655	367.207965	169.949115	90.004425	33
std	16.466631	37.170340	16.590803	15.364394	44.842283	33.385421	35.633072	25.826643	45
min	0.000000	105.000000	6.000000	55.000000	0.000000	232.000000	108.000000	0.000000	-172
25%	36.000000	160.000000	59.000000	80.000000	142.000000	350.000000	148.000000	79.000000	3
50%	47.000000	164.000000	68.000000	86.000000	157.000000	367.000000	162.000000	91.000000	40
75%	58.000000	170.000000	79.000000	94.000000	175.000000	384.000000	179.000000	102.000000	66
max	83.000000	780.000000	176.000000	188.000000	524.000000	509.000000	381.000000	205.000000	169

8 rows × 272 columns

Atributos categóricos

In [16]:
```
cats = df.select_dtypes(include=['O'])
```

In [17]:
```
cats.describe()
```

Out[17]:

	Sex	Existence_ragged_R_wave	Existence_diphasic_derivation_R_wave	Existence_ragged_P_wave	Existe
count	452	452	452	452	
unique	2	2	2	2	
top	b'1'	b'0'	b'0'	b'0'	
freq	249	451	447	447	

c. Avalie os resultados dos processos abaixo, caso sejam utilizados na base de dados, após o processo de classificação com J48 ter sido utilizado

i) Limpeza de dados(Outilier, Missing)
ii) Normalização
iii) Discretização

Primeiramente, vamos converter os bytes para inteiros

In [18]:
```
bytes_columns = df.select_dtypes(include=['O']).columns

for column in df[bytes_columns]:
    if column == "Sex":
        df[column] = df[column].apply(lambda sex: 1 if sex == b'0' else 0)
    else:
        df[column] = df[column].apply(lambda x: int(x))
```

Missing Values

Vamos começar identificando quantos missing values temos e em quais atributos se encontram

Iremos lidar com os "Missing Values" antes de lidarmos com os "Outliers" pois o algoritmo utilizado para a detecção de "Outliers" não suporta valores do tipo NaN

In [19]:
```
df.isnull().sum()[lambda x: x != 0]
```

Out[19]:
```
T              8
P             22
QRST           1
J            376
Heart_rate     1
dtype: int64
```

In [20]:
```
# Agora vamos ver quantos % relativo ao total está faltando
TOTAL = df.shape[0]
missing = df.isnull().sum()[lambda x: x != 0]
missing.apply(lambda x: (x/TOTAL)*100)
```

Out[20]:
```
T             1.769912
P             4.867257
QRST          0.221239
J            83.185841
Heart_rate    0.221239
dtype: float64
```

80% da coluna "J" são "Missing Values", Vamos removê-la

In [21]:
```
df.drop('J',axis=1,inplace=True)
```

Vamos remover também as linhas das colunas que possuem apenas um valor missing

In [22]:
```
df = df.dropna(subset=["QRST","Heart_rate"])
```

Para as colunas "QRST" e "Heart_rate" apenas um elemento de cada consta como "Missing Value", o mesmo não se aplica para as colunas "T" e "P", que possuem ~5% e ~2% das linhas sem os dados. Logo, foi decidido que para os atributos que possuem apenas um elemento sem dado marcado, iríamos remover a linha, visto que terá um impacto mínimo no conjunto de dados, para os atributos que possuem uma

quantidade maior do que 1%, iremos utilizar um algoritmo. "J" foi removida justamente por possuir **muitos** missing values, em uma quantidade que inviabilizou qualquer ação na coluna

Para as colunas "T" e "P" iremos usar o knn imputer da scikit learn

In [23]:
```
from sklearn.impute import KNNImputer
```

In [24]:
```
imputer = KNNImputer()
```

In [25]:
```
df[['T', 'P']] = imputer.fit_transform(df[['T','P']].to_numpy())
```

In [26]:
```
df.isnull().sum()[lambda x: x!=0]
```

Out[26]:
```
Series([], dtype: int64)
```

Lidamos com todos os missing values

Outliers

Vamos analisar os Outliers primeiro, pois podem ser gerados mais missing values com a análise

Será utilizado LocalOutlierFactor, um algoritmo não supervisionado que mede a densidade do conjunto de amostras e compara com o dado, caso um dado se encontre distante de uma área de densidade, pode ser um Outlier

In [27]:
```
from sklearn.neighbors import LocalOutlierFactor
```

Vamos criar uma função que itera por cada atributo desse DataFrame, buscando Outliers

In [28]:
```
def train_model(column):
    clf = LocalOutlierFactor(n_neighbors=int(nums[column].shape[0]*0.99)) # Numero de vi
    # Pelo que eu entendi do algoritmo e da documentação, iremos marcar os que estiverem
    y_pred = clf.fit_predict(nums[column].values.reshape(-1,1))

    return y_pred
```

In [29]:
```
def find_outliers(l):
    positions = []
    for i, item in enumerate(l):
        if item == -1:
            positions.append(i)

    return positions
```

In [30]:
```
nums = df.select_dtypes(include=['float'])
count = 0 # Para evitar cobrir muito espaço em pdf
for i in nums.columns:
    # Primeiro treinamos para aquela coluna
    print(i)
    try:
        tmp_train = train_model(i)

        # Depois salvamos quais são os valores que destoaram do modelo
        tmp_list_out = find_outliers(tmp_train)
        print(tmp_list_out)
    except ValueError:
```

```
    #if count == 20: break

    #count += 1
```

```
Age
[]
Height
[140, 314]
Weight
[212]
QRS_duration
[]
P-R
[391]
Q-T
[]
T_interval
[27]
P_interval
[]
QRS
[]
T
[]
P
[269]
QRST
[]
Heart_rate
[314]
Q_wave
[75, 101]
R_wave
[]
S_wave
[]
R_lin_wave
[115, 192, 212, 420]
S_lin_wave
[]
Number_itrinsic_deflections
[84]
DII_Q_wave
[29, 189]
DII_R_wave
[]
DII_S_wave
[]
DII_R_lin_wave
[0]
DII_S_lin__wave
[140, 310]
DII_Number_itrinsic_deflections
[]
DII_Existence_ragged_R_wave
[]
DII_Existence_diphasic_derivation_R_wave
[]
DII_Existence_ragged_P_wave
[212]
DII_Existence_diphasic_drivation_P_wave
[111, 206]
DII_Existence_ragged_T_wave
[35, 306]
DII_Existence_diphasic_derivation_T_wave
[]
DIII_Q_wave
[]
```

DIII_R_wave
[]
DIII_S_wave
[386]
DIII_R_lin__wave
[]
DIII_S_lin__wave
[162]
DIII_Number_itrinsic_deflections
[]
DIII_Existence_ragged_R_wave
[438]
DIII_Existence_diphasic_derivation_R_wave
[]
DIII_Existence_ragged_P_wave
[380]
DIII_Existence_diphasic_drivation_P_wave
[]
DIII_Existence_ragged_T_wave
[87, 107, 249, 265]
DIII_Existence_diphasic_derivation_T_wave
[140, 370]
AVR_1
[]
AVR_2
[]
AVR_3
[]
AVR_4
[425]
AVR_5
[316]
AVR_6
[]
AVR_7
[]
AVR_8
[289, 364]
AVR_9
[132, 212]
AVR_10
[45, 192]
AVR_11
[140, 186]
AVR_12
[206, 212, 368, 379]
AVL_1
[]
AVL_2
[]
AVL_3
[]
AVL_4
[]
AVL_5
[]
AVL_6
[]
AVL_7
[]
AVL_8
[]
AVL_9
[348]

AVL_10
[217]
AVL_11
[132, 265, 289]
AVL_12
[140]
AVF_1
[]
AVF_2
[]
AVF_3
[386]
AVF_4
[]
AVF_5
[140, 352, 364]
AVF_6
[]
AVF_7
[28, 332]
AVF_8
[]
AVF_9
[]
AVF_10
[132]
AVF_11
[97]
AVF_12
[368]
V1_1
[]
V1_2
[291, 295]
V1_3
[]
V1_4
[]
V1_5
[59, 157, 447]
V1_6
[]
V1_7
[203, 314, 401]
V1_8
[]
V1_9
[13, 365, 383, 399]
V1_10
[107, 149, 314, 428]
V1_11
[]
V1_12
[]
V2_1
[]
V2_2
[291, 295]
V2_3
[]
V2_4
[428]
V2_5
[311, 325, 377]

V2_6
[]
V2_7
[84, 186, 250, 297]
V2_8
[291, 295, 395, 401]
V2_9
[]
V2_10
[25, 107, 197, 314]
V2_11
[107, 217, 294, 354]
V2_12
[]
V3_1
[]
V3_2
[295]
V3_3
[]
V3_4
[2, 295, 377]
V3_5
[2, 377]
V3_6
[295]
V3_7
[]
V3_8
[249, 295]
V3_9
[25, 84, 334]
V3_10
[25, 107, 217, 399]
V3_11
[28, 186, 320]
V3_12
[]
V4_1
[]
V4_2
[]
V4_3
[]
V4_4
[]
V4_5
[361, 424]
V4_6
[]
V4_7
[203, 354]
V4_8
[]
V4_9
[]
V4_10
[]
V4_11
[85, 217]
V4_12
[]
V5_1
[75, 217, 352]

V5_2
[]
V5_3
[417]
V5_4
[84]
V5_5
[]
V5_6
[84, 206]
V5_7
[]
V5_8
[87, 188, 206]
V5_9
[]
V5_10
[265]
V5_11
[]
V5_12
[42, 75, 255]
V6_1
[75, 217]
V6_2
[]
V6_3
[]
V6_4
[90, 278, 345]
V6_5
[]
V6_6
[84, 206]
V6_7
[439]
V6_8
[184]
V6_9
[401, 439]
V6_10
[]
V6_11
[]
V6_12
[]
Amp_JJ_wave
[386]
Amp_Q_wave
[]
Amp_R_wave
[]
Amp_S_wave
[140, 401]
Amp_R_lin_wave
[115, 192, 212, 420]
Amp_S_lin_wave
[]
Amp_P_wave
[401]
Amp_T_wave
[386]
QRSA
[386]

QRSTA
[]
Amp_DII_1
[386]
Amp_DII_2
[]
Amp_DII_3
[]
Amp_DII_4
[447]
Amp_DII_5
[140]
Amp_DII_6
[140, 310]
Amp_DII_7
[]
Amp_DII_8
[]
Amp_DII_9
[]
Amp_DII_10
[]
Amp_DIII_1
[386]
Amp_DIII_2
[325]
Amp_DIII_3
[4]
Amp_DIII_4
[]
Amp_DIII_5
[140]
Amp_DIII_6
[162, 333, 417, 425]
Amp_DIII_7
[447]
Amp_DIII_8
[197, 422]
Amp_DIII_9
[]
Amp_DIII_10
[4]
Amp_AVR_1
[386]
Amp_AVR_2
[]
Amp_AVR_3
[209, 401, 447]
Amp_AVR_4
[]
Amp_AVR_5
[]
Amp_AVR_6
[316]
Amp_AVR_7
[401]
Amp_AVR_8
[386]
Amp_AVR_9
[386]
Amp_AVR_10
[]
Amp_AVL_1
[132, 386]

```
Amp_AVL_2
[203, 314]
Amp_AVL_3
[]
Amp_AVL_4
[]
Amp_AVL_5
[82, 101]
Amp_AVL_6
[]
Amp_AVL_7
[447]
Amp_AVL_8
[386]
Amp_AVL_9
[386]
Amp_AVL_10
[]
Amp_AVF_1
[422]
Amp_AVF_2
[]
Amp_AVF_3
[]
Amp_AVF_4
[447]
Amp_AVF_5
[140, 321]
Amp_AVF_6
[140, 352, 364]
Amp_AVF_7
[447]
Amp_AVF_8
[197, 209, 422]
Amp_AVF_9
[]
Amp_AVF_10
[]
Amp_V1_1
[386]
Amp_V1_2
[]
Amp_V1_3
[295, 401]
Amp_V1_4
[386]
Amp_V1_5
[59, 377, 447]
Amp_V1_6
[59, 157, 447]
Amp_V1_7
[197]
Amp_V1_8
[386]
Amp_V1_9
[295, 386]
Amp_V1_10
[295, 422]
Amp_V2_1
[84, 386]
Amp_V2_2
[]
Amp_V2_3
[357, 408]
```

```
Amp_V2_4
[4]
Amp_V2_5
[2, 377, 430]
Amp_V2_6
[311, 325, 377]
Amp_V2_7
[447]
Amp_V2_8
[]
Amp_V2_9
[]
Amp_V2_10
[]
Amp_V3_1
[84, 422]
Amp_V3_2
[203, 255]
Amp_V3_3
[]
Amp_V3_4
[]
Amp_V3_5
[2, 348, 377]
Amp_V3_6
[2, 377]
Amp_V3_7
[84]
Amp_V3_8
[]
Amp_V3_9
[84]
Amp_V3_10
[]
Amp_V4_1
[84]
Amp_V4_2
[75, 203, 217]
Amp_V4_3
[]
Amp_V4_4
[107]
Amp_V4_5
[107, 334]
Amp_V4_6
[361, 424]
Amp_V4_7
[173]
Amp_V4_8
[84]
Amp_V4_9
[]
Amp_V4_10
[447]
Amp_V5_1
[386]
Amp_V5_2
[75, 217, 352]
Amp_V5_3
[]
Amp_V5_4
[447]
Amp_V5_5
[84]
```

```
Amp_V5_6
[]
Amp_V5_7
[197]
Amp_V5_8
[]
Amp_V5_9
[]
Amp_V5_10
[]
Amp_V6_1
[84, 386]
Amp_V6_2
[217]
Amp_V6_3
[]
Amp_V6_4
[401, 447]
Amp_V6_5
[90, 278, 345]
Amp_V6_6
[]
Amp_V6_7
[197]
Amp_V6_8
[]
Amp_V6_9
[]
Amp_V6_10
[]
```

In [31]:
```
pd.set_option("display.max_columns", None)
```

Vamos analisar os casos para "Height"

In [32]:
```
nums.iloc[[140,314]]
```

Out[32]:

	Age	Height	Weight	QRS_duration	P-R	Q-T	T_interval	P_interval	QRS	T	P	QRST	Heart_r
141	1.0	780.0	6.0	85.0	165.0	237.0	150.0	106.0	88.0	30.0	30.0	52.0	13
316	0.0	608.0	10.0	83.0	126.0	232.0	128.0	60.0	125.0	21.0	-50.0	102.0	16

Observação 1

Certamente são dados errados(Note que o 316, que é 314 na lista mostrada, causou diversos outliers em outros atributos)

In [33]:
```
height = df[df["Height"] > 200]
```

In [34]:
```
height
```

Out[34]:

	Age	Sex	Height	Weight	QRS_duration	P-R	Q-T	T_interval	P_interval	QRS	T	P	QRST	H
141	1.0	0	780.0	6.0	85.0	165.0	237.0	150.0	106.0	88.0	30.0	30.0	52.0	
316	0.0	1	608.0	10.0	83.0	126.0	232.0	128.0	60.0	125.0	21.0	-50.0	102.0	

In [35]:
```
df = df.drop([141, 316])
```

Observação 2

Diversas colunas parecem ter valores constantes, que não são importantes para os algoritmos de Machine Learning, os atributos devem ser indicadores que ajudam a determinar o alvo. Se um atributo sempre contém o mesmo valor, ele não contribui para saber o valor de destino.

In [36]:
```
df.columns[df.nunique() <= 1]
```

Out[36]:
```
Index(['S_lin_wave', 'AVL_5', 'AVL_7', 'AVL_12', 'AVF_9', 'V4_9', 'V4_10',
       'V5_5', 'V5_7', 'V5_9', 'V5_11', 'V6_5', 'V6_10', 'V6_11',
       'Amp_S_lin_wave', 'Amp_AVL_6', 'Amp_V5_6', 'Amp_V6_6'],
      dtype='object')
```

In [37]:
```
len(df.columns[df.nunique() <= 1])
```

Out[37]:
```
18
```

In [38]:
```
df.drop(df.columns[df.nunique() <= 1], axis=1, inplace=True)
```

In [39]:
```
numerals = df.select_dtypes(include='float')
numerals.columns[numerals.nunique() <= 2]
```

Out[39]:
```
Index(['DII_S_lin__wave', 'DII_Existence_ragged_R_wave',
       'DII_Existence_diphasic_derivation_R_wave',
       'DII_Existence_ragged_P_wave',
       'DII_Existence_diphasic_drivation_P_wave',
       'DII_Existence_ragged_T_wave',
       'DII_Existence_diphasic_derivation_T_wave',
       'DIII_Existence_ragged_R_wave',
       'DIII_Existence_diphasic_derivation_R_wave',
       'DIII_Existence_ragged_P_wave',
       'DIII_Existence_diphasic_drivation_P_wave',
       'DIII_Existence_ragged_T_wave',
       'DIII_Existence_diphasic_derivation_T_wave', 'AVR_5', 'AVR_7', 'AVR_8',
       'AVR_9', 'AVR_10', 'AVR_11', 'AVR_12', 'AVL_8', 'AVL_9', 'AVL_10',
       'AVL_11', 'AVF_7', 'AVF_8', 'AVF_10', 'AVF_11', 'AVF_12', 'V1_7',
       'V1_8', 'V1_9', 'V1_10', 'V1_11', 'V1_12', 'V2_7', 'V2_8', 'V2_9',
       'V2_10', 'V2_11', 'V2_12', 'V3_7', 'V3_8', 'V3_9', 'V3_10', 'V3_11',
       'V3_12', 'V4_7', 'V4_8', 'V4_11', 'V4_12', 'V5_8', 'V5_10', 'V5_12',
       'V6_7', 'V6_8', 'V6_9', 'V6_12', 'Amp_DII_6', 'Amp_AVR_6'],
      dtype='object')
```

In [40]:
```
df.drop(numerals.columns[numerals.nunique() <= 2], axis=1, inplace=True)
```

In [41]:
```
df.shape
```

Out[41]:
```
(448, 201)
```

Foi feita uma análise de todos os outros atributos apontados como "Outliers", porém não foram encontrados dados que destoassem do resto de forma à atrapalhar os algoritmos de classificação

Vamos dividir em treino e teste. E testar como um modela de árvore de decisão para avaliar o desempenho

In [42]:
```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score

train, teste = train_test_split(df, test_size=0.1,random_state=42)

train_X = train.drop("Class", axis=1)
train_y = train.Class
```

```
test_X = teste.drop("Class", axis=1)
test_y = teste.Class

forest_model = RandomForestClassifier(random_state=1)
forest_model.fit(train_X, train_y)
preds = forest_model.predict(test_X)
print(accuracy_score(test_y, preds))
print(f1_score(test_y, preds, average='weighted'))
```

```
0.7333333333333333
0.6886772486772486
```

Aplicando a normalização e testando nosso modelo

In [43]:
```
train, teste = train_test_split(df, test_size=0.1, random_state=42)

train_X = train.drop("Class", axis=1)
train_y = train.Class

test_X = teste.drop("Class", axis=1)
test_y = teste.Class

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
normalized_x_train = pd.DataFrame(
    scaler.fit_transform(train_X),
    columns = train_X.columns
)

forest_model = RandomForestClassifier(random_state=1)
forest_model.fit(normalized_x_train, train_y)

normalized_x_test = pd.DataFrame(
    scaler.transform(test_X),
    columns = train_X.columns
)

preds = forest_model.predict(normalized_x_test)
print(accuracy_score(test_y, preds))
print(f1_score(test_y, preds, average='weighted'))
```

```
0.7333333333333333
0.6886772486772486
```

In [44]:
```
df.shape
```

Out[44]:
```
(448, 201)
```

In [45]:
```
df['Class'].describe()
```

Out[45]:
```
count    448.000000
mean       3.875000
std        4.421576
min        1.000000
25%        1.000000
50%        1.000000
75%        6.000000
max       16.000000
Name: Class, dtype: float64
```

In [46]:
```
df['Class'].value_counts()
```

Out[46]:
```
1     244
10     50
2      44
6      25
16     22
3      15
4      15
5      11
9       9
15      5
14      4
8       2
7       2
Name: Class, dtype: int64
```

2 - Seleção de variáveis e aplicação dos modelos

Utilizando as técnicas de Seleção de variáveis:

.Random Forest
.Select K Best
.Mutual Information

iremos aplicar os algoritmos de classificação:

.Logistic Regression
.K-Nearest Neighbors
.Random Forest
.Gaussian Naive Bayes

Em diferentes cenários propostos pelas técnicas de seleção de variáveis, assim procurando os melhores modelos

In [47]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB

from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score

from sklearn.model_selection import RandomizedSearchCV

import numpy as np
```

Random Forest

In [48]:
```
from sklearn.ensemble import RandomForestRegressor

# Separar os dados em features (X) e target (y)
X = df.drop('Class', axis=1)
y = df['Class']

# Criar o modelo Random Forest
orestRegressor()
```

```
# Treinar o modelo
rf.fit(X, y)

# Obter a importância das variáveis
feature_importance = rf.feature_importances_

# Criar um DataFrame com as variáveis e suas importâncias
importance_df = pd.DataFrame({'Variable': X.columns, 'Importance': feature_importance})

# Ordenar as variáveis por importância
importance_df = importance_df.sort_values(by='Importance', ascending=False)
importance_df
```

Out[48]:

	Variable	Importance
146	Amp_V1_5	0.104637
4	QRS_duration	0.069399
13	Heart_rate	0.058485
150	Amp_V1_9	0.031362
8	P_interval	0.030705
...	...	...
84	V6_4	0.000000
177	Amp_V4_6	0.000000
62	V2_5	0.000000
67	V3_4	0.000000
74	V4_5	0.000000

200 rows × 2 columns

Random Forest classifier

In [49]:

```
def selecting_best_rf_rf():
    hash_table = {}
    columns2select = [20, 30, 50, 70, 100, 150, 200]
    for qtd in columns2select:
        atribute_columns = list(importance_df.head(qtd)['Variable'])
        tmp_df = train_X[atribute_columns]

        foest_model = RandomForestClassifier(random_state=3)

        forest_model.fit(tmp_df, train_y)

        preds_tmp = forest_model.predict(test_X[atribute_columns])

        hash_table[qtd] = [accuracy_score(test_y, preds_tmp), f1_score(test_y, preds_tmp
    return hash_table

def table_rf_rf():
    table = selecting_best_rf_rf()

    for value in table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(table[value][0])
        print('F1:       ', end='')
        print(table[value][1])
```

```
        print()

table_rf_rf()
```

```
Número de atributos selecionados:  20
Acurácia: 0.6888888888888889
F1:       0.6035978835978835

Número de atributos selecionados:  30
Acurácia: 0.7111111111111111
F1:       0.6264004096262161

Número de atributos selecionados:  50
Acurácia: 0.7333333333333333
F1:       0.6572998805256869

Número de atributos selecionados:  70
Acurácia: 0.7333333333333333
F1:       0.6572998805256869

Número de atributos selecionados:  100
Acurácia: 0.7555555555555555
F1:       0.6895543000627746

Número de atributos selecionados:  150
Acurácia: 0.7555555555555555
F1:       0.6855026455026454

Número de atributos selecionados:  200
Acurácia: 0.7111111111111111
F1:       0.6380406212664277
```

KNN

Primeiro, vamos encontrar o melhor número de "vizinhos" no modelo

In [50]:
```
def find_best_n(df):
    best_score = -np.inf
    best_n_knn = np.inf
    print(df.shape)
    for n in range(1,df.columns.shape[0]):
        temp_model = KNeighborsClassifier(n_neighbors=n)
        print(n,end=" ")

        temp_model.fit(train_X, train_y)

        temp_pred = temp_model.predict(test_X)

        temp_score = f1_score(test_y, temp_pred, average='weighted')

        #print(temp_score)
        if temp_score > best_score:
                best_score = temp_score
                best_n_knn = n

    #print('-'*30)
    #print("Best performing number of n_neighbors is",best_n_knn,"scoring",round(best_sc
    return best_n_knn
```

In [51]:
```
tmp_gain_df = importance_df[importance_df['Importance']>0]
atribute_columns = list(tmp_gain_df.head(tmp_gain_df.shape[0])['Variable'])
tmp_df = train_X[atribute_columns]
_best_n(tmp_df)
```

```
(403, 182)
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 3
3 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62
63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 9
2 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 1
16 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 1
38 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 1
60 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180 181
```

Agora, rodamos o KNN nas variáveis selecionadas pelo RandomForest

In [52]:

```
def selecting_best_rf_knn():
    hash_table = {}
    tmp_gain_df = importance_df[importance_df['Importance']>0]
    columns2select = [20, 30, 50, 70, 100, 150, tmp_gain_df.shape[0]]
    #best_n = find_best_n(tmp_gain_df)
    for qtd in columns2select:
        atribute_columns = list(tmp_gain_df.head(qtd)['Variable'])
        tmp_df = train_X[atribute_columns]

        #best_n_knn = find_best(tmp_df)
        knn_model = KNeighborsClassifier(n_neighbors=best_n)

        knn_model.fit(tmp_df, train_y)

        knn_pred = knn_model.predict(test_X[atribute_columns])

        hash_table[qtd] = [accuracy_score(test_y, knn_pred), f1_score(test_y, knn_pred,

    return hash_table

def table_rf_knn():
    table = selecting_best_rf_knn()

    for value in table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(table[value][0])
        print('F1:       ', end='')
        print(table[value][1])
        print()

table_rf_knn()
```

```
Número de atributos selecionados:  20
Acurácia: 0.6444444444444445
F1:       0.5924338624338624

Número de atributos selecionados:  30
Acurácia: 0.5777777777777777
F1:       0.47962962962962963

Número de atributos selecionados:  50
Acurácia: 0.6444444444444445
F1:       0.5343790849673202

Número de atributos selecionados:  70
Acurácia: 0.6888888888888889
F1:       0.6

Número de atributos selecionados:  100
Acurácia: 0.6222222222222222
F1:       0.5326495726495726

Número de atributos selecionados:  150
Acurácia: 0.6666666666666666
F1:       0.5782642343836374

Número de atributos selecionados:  182
Acurácia: 0.7333333333333333
F1:       0.6706974506974507
```

Logistic Regression

In [53]:

```
def selecting_best_rf_log():
    hash_table = {}
    tmp_gain_df = importance_df[importance_df['Importance']>0]

    columns2select = [20, 30, 50, 70, 100, 150, tmp_gain_df.shape[0]]

    for qtd in columns2select:
        atribute_columns = list(tmp_gain_df.head(qtd)['Variable'])
        tmp_df = train_X[atribute_columns]

        log_model = LogisticRegression(random_state=0,solver = "saga")

        log_model.fit(tmp_df, train_y)

        log_pred = log_model.predict(test_X[atribute_columns])

        hash_table[qtd] = [accuracy_score(test_y, log_pred), f1_score(test_y, log_pred,

    return hash_table

def table_rf_log():
    table = selecting_best_rf_log()

    for value in table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(table[value][0])
        print('F1:       ', end='')
        print(table[value][1])
        print()
table_rf_log()
```

```
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
Número de atributos selecionados:  20
Acurácia: 0.6
F1:       0.5311891751236014

Número de atributos selecionados:  30
Acurácia: 0.6444444444444445
F1:       0.5517460317460318

Número de atributos selecionados:  50
Acurácia: 0.6444444444444445
F1:       0.5507936507936507

Número de atributos selecionados:  70
Acurácia: 0.6888888888888889
F1:       0.6311111111111111

Número de atributos selecionados:  100
Acurácia: 0.7111111111111111
F1:       0.6785112205801861

Número de atributos selecionados:  150
Acurácia: 0.7777777777777778
F1:       0.7556632321544602

Número de atributos selecionados:  182
Acurácia: 0.7777777777777778
F1:       0.7556632321544602

/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: ConvergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

Gaussian Naive Bayes

In [54]:
```python
def selecting_best_rf_nb():
    hash_table = {}
    tmp_gain_df = importance_df[importance_df['Importance']>0]
    columns2select = [20, 30, 50, 70, 100, 150, tmp_gain_df.shape[0]]

    for qtd in columns2select:
        atribute_columns = list(tmp_gain_df.head(qtd)['Variable'])
        tmp_df = train_X[atribute_columns]

        nb_model = GaussianNB(var_smoothing = 0.5) # var_smoothing foi decidido no 'olho
```

Loading [MathJax]/extensions/Safe.js

```
        nb_model.fit(tmp_df, train_y)

        nb_pred = nb_model.predict(test_X[atribute_columns])

        hash_table[qtd] = [accuracy_score(test_y, nb_pred), f1_score(test_y, nb_pred, av

    return hash_table

def table_rf_nb():
    table = selecting_best_rf_nb()

    for value in table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(table[value][0])
        print('F1:       ', end='')
        print(table[value][1])
        print()

table_rf_nb()
```

```
Número de atributos selecionados:  20
Acurácia: 0.6444444444444445
F1:       0.5286346047540077

Número de atributos selecionados:  30
Acurácia: 0.6666666666666666
F1:       0.5524709784411277

Número de atributos selecionados:  50
Acurácia: 0.6666666666666666
F1:       0.5616161616161617

Número de atributos selecionados:  70
Acurácia: 0.7111111111111111
F1:       0.6398809523809524

Número de atributos selecionados:  100
Acurácia: 0.7111111111111111
F1:       0.64831541218638

Número de atributos selecionados:  150
Acurácia: 0.8
F1:       0.7231746031746032

Número de atributos selecionados:  182
Acurácia: 0.7555555555555555
F1:       0.6819047619047619
```

K best

Random Forest

In [55]:
```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import f_classif

def selecting_best_kbest_rf():
    hash_table = {}
    columns2select = [20, 30, 40, 50, 60, 100, 150, 200]
    for qtd in columns2select:
        selector = SelectKBest(f_classif, k=qtd)
        k_best_data = pd.DataFrame(selector.fit_transform(train_X, train_y), columns=sel
```

```
        #k_best_data

        forest_model = RandomForestClassifier(random_state=3)

        forest_model.fit(k_best_data, train_y)

        preds_tmp = forest_model.predict(test_X[selector.get_feature_names_out()])

        hash_table[qtd] = [accuracy_score(test_y, preds_tmp), f1_score(test_y, preds_tmp
        #print('Para as ', value, 'primeiras colunas')
        #print(accuracy_score(test_y, preds_tmp))
        #print(f1_score(test_y, preds_tmp, average='weighted'))
        #print()
        #print('-'*30)
    return hash_table

def table_kbest_rf():
    hash_table = selecting_best_kbest_rf()
    for value in hash_table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(hash_table[value][0])
        print('F1:       ', end='')
        print(hash_table[value][1])
        print()

table_kbest_rf()
```

```
Número de atributos selecionados:  20
Acurácia: 0.6888888888888889
F1:       0.5958597883597883

Número de atributos selecionados:  30
Acurácia: 0.7333333333333333
F1:       0.6696296296296296

Número de atributos selecionados:  40
Acurácia: 0.7333333333333333
F1:       0.6664550264550264

Número de atributos selecionados:  50
Acurácia: 0.7777777777777778
F1:       0.7478781244298487

Número de atributos selecionados:  60
Acurácia: 0.7777777777777778
F1:       0.7429934535019282

Número de atributos selecionados:  100
Acurácia: 0.7777777777777778
F1:       0.7429934535019282

Número de atributos selecionados:  150
Acurácia: 0.7555555555555555
F1:       0.711776522284997

Número de atributos selecionados:  200
Acurácia: 0.7333333333333333
F1:       0.6833272616879174
```

KNN

In [56]:
```
def selecting_best_kbest_knn():
    hash_table = {}
    columns2select = [20, 30, 50, 70, 100, 150, 200]
    for qtd in columns2select:
        selector = SelectKBest(f_classif, k=qtd)
        k_best_data = pd.DataFrame(selector.fit_transform(train_X, train_y), columns=sel

        knn_model = KNeighborsClassifier(n_neighbors=3)

        knn_model.fit(k_best_data, train_y)

        knn_pred = knn_model.predict(test_X[selector.get_feature_names_out()])

        hash_table[qtd] = [accuracy_score(test_y, knn_pred), f1_score(test_y, knn_pred,

    return hash_table

def table_kbest_knn():
    table = selecting_best_kbest_knn()

    for value in table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(table[value][0])
        print('F1:       ', end='')
        print(table[value][1])
        print()

table_kbest_knn()
```

```
Número de atributos selecionados:  20
Acurácia: 0.6888888888888889
F1:       0.5951388888888889

Número de atributos selecionados:  30
Acurácia: 0.6666666666666666
F1:       0.5556902356902357

Número de atributos selecionados:  50
Acurácia: 0.6222222222222222
F1:       0.5401058201058202

Número de atributos selecionados:  70
Acurácia: 0.6666666666666666
F1:       0.5705050505050505

Número de atributos selecionados:  100
Acurácia: 0.6222222222222222
F1:       0.5556695156695157

Número de atributos selecionados:  150
Acurácia: 0.6444444444444445
F1:       0.5768981481481482

Número de atributos selecionados:  200
Acurácia: 0.7333333333333333
F1:       0.6706974506974507
```

Logistic Regression

In [57]:
```
def selecting_best_kbest_log():
    hash_table = {}
    columns2select = [20, 30, 40, 50, 60, 100, 150, 200]
```

```python
    for qtd in columns2select:
        selector = SelectKBest(f_classif, k=qtd)
        k_best_data = pd.DataFrame(selector.fit_transform(train_X, train_y), columns=sel

        log_model = LogisticRegression(random_state=0,solver = "saga")

        log_model.fit(k_best_data, train_y)

        log_pred = log_model.predict(test_X[selector.get_feature_names_out()])

        hash_table[qtd] = [accuracy_score(test_y, log_pred), f1_score(test_y, log_pred,

    return hash_table

def table_kbest_log():
    hash_table = selecting_best_kbest_log()
    for value in hash_table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(hash_table[value][0])
        print('F1:       ', end='')
        print(hash_table[value][1])
        print()

table_kbest_log()
```

```
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

```
Número de atributos selecionados:  20
Acurácia: 0.7111111111111111
F1:       0.6247685185185184

Número de atributos selecionados:  30
Acurácia: 0.7111111111111111
F1:       0.6448677248677248

Número de atributos selecionados:  40
Acurácia: 0.6888888888888889
F1:       0.6198351981958539

Número de atributos selecionados:  50
Acurácia: 0.6888888888888889
F1:       0.6232258064516129

Número de atributos selecionados:  60
Acurácia: 0.7333333333333333
F1:       0.6862962962962962

Número de atributos selecionados:  100
Acurácia: 0.6888888888888889
F1:       0.6369312169312169

Número de atributos selecionados:  150
Acurácia: 0.7555555555555555
F1:       0.7144900565590221

Número de atributos selecionados:  200
Acurácia: 0.7777777777777778
F1:       0.7556632321544602
```

```
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

Gaussian Naive Bayes

In [58]:

```
def selecting_best_kbest_nb():
    hash_table = {}
    columns2select = [20, 30, 40, 50, 60, 100, 150, 200]
    for qtd in columns2select:
        selector = SelectKBest(f_classif, k=qtd)
        k_best_data = pd.DataFrame(selector.fit_transform(train_X, train_y), columns=sel
        #k_best_data

        nb_model = GaussianNB(var_smoothing = 0.5) # var_smoothing foi decidido no 'olho

        nb_model.fit(k_best_data, train_y)

        nb_pred = nb_model.predict(test_X[selector.get_feature_names_out()])

        hash_table[qtd] = [accuracy_score(test_y, nb_pred), f1_score(test_y, nb_pred, av
        #print('Para as ', value, 'primeiras colunas')
        #print(accuracy_score(test_y, preds_tmp))
        #print(f1_score(test_y, preds_tmp, average='weighted'))
        #print()
        #print('-'*30)
    return hash_table


def table_kbest_nb():
    hash_table = selecting_best_kbest_nb()
    for  in hash_table:
```

```
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(hash_table[value][0])
        print('F1:       ', end='')
        print(hash_table[value][1])
        print()

table_kbest_nb()
```

```
Número de atributos selecionados:  20
Acurácia: 0.6888888888888889
F1:       0.5742483660130719

Número de atributos selecionados:  30
Acurácia: 0.6888888888888889
F1:       0.5724709784411277

Número de atributos selecionados:  40
Acurácia: 0.6888888888888889
F1:       0.5813598673300167

Número de atributos selecionados:  50
Acurácia: 0.6888888888888889
F1:       0.5813598673300167

Número de atributos selecionados:  60
Acurácia: 0.7111111111111111
F1:       0.6183969043670536

Número de atributos selecionados:  100
Acurácia: 0.7555555555555555
F1:       0.6953968253968255

Número de atributos selecionados:  150
Acurácia: 0.7333333333333333
F1:       0.6785858585858585

Número de atributos selecionados:  200
Acurácia: 0.7555555555555555
F1:       0.6819047619047619
```

Mutual Info

In [59]:

```
import pandas as pd
from sklearn.feature_selection import mutual_info_classif

# Separar os dados em features (X) e target (y)
X = df.drop('Class', axis=1)
y = df['Class']

# Calcular o ganho de informação entre cada variável e a variável alvo
gain = mutual_info_classif(X, y)

# Criar um DataFrame com as variáveis e seus ganhos de informação
gain_df = pd.DataFrame({'Variable': X.columns, 'Gain': gain})

# Ordenar as variáveis por ganho de informação
gain_df = gain_df.sort_values(by='Gain', ascending=False)

# Exibir as variáveis em ordem decrescente de ganho de informação
gain_df
```

Out[59]:

	Variable	Gain
13	Heart_rate	0.264960
197	Amp_V6_8	0.248316
120	Amp_AVR_8	0.225076
150	Amp_V1_9	0.224810
66	V3_3	0.218437
...	...	...
22	Existence_diphasic_drivation_P_wave	0.000000
109	Amp_DIII_6	0.000000
20	Existence_diphasic_derivation_R_wave	0.000000
176	Amp_V4_5	0.000000
90	Amp_R_lin_wave	0.000000

200 rows × 2 columns

Random Forest

In [60]:

```
def selecting_best_mi_rf():

    hash_table = {}

    tmp_gain_df = gain_df[gain_df['Gain']>0]
    columns2select = [20, 30, 40, 50, 100, 150, tmp_gain_df.shape[0]]

    for qtd in columns2select:
        atribute_columns = list(tmp_gain_df.head(qtd)['Variable'])
        tmp_df = train_X[atribute_columns]

        forest_model = RandomForestClassifier(random_state=3)

        forest_model.fit(tmp_df, train_y)

        preds_tmp = forest_model.predict(test_X[atribute_columns])

        hash_table[qtd] = [accuracy_score(test_y, preds_tmp), f1_score(test_y, preds_tmp

    return hash_table


def table_mi_rf():
    hash_table = selecting_best_mi_rf()
    for value in hash_table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(hash_table[value][0])
        print('F1:       ', end='')
        print(hash_table[value][1])
        print()
table_mi_rf()
```

```
Número de atributos selecionados:  20
Acurácia: 0.7777777777777778
F1:       0.7029993928354584

Número de atributos selecionados:  30
Acurácia: 0.7333333333333333
F1:       0.6720939826024572

Número de atributos selecionados:  40
Acurácia: 0.7333333333333333
F1:       0.6720939826024572

Número de atributos selecionados:  50
Acurácia: 0.7777777777777778
F1:       0.740893399254055

Número de atributos selecionados:  100
Acurácia: 0.7555555555555555
F1:       0.711776522284997

Número de atributos selecionados:  150
Acurácia: 0.7333333333333333
F1:       0.6886772486772486

Número de atributos selecionados:  175
Acurácia: 0.7555555555555555
F1:       0.7123056228140976
```

KNN

In [61]:

```
def selecting_best_mi_knn():
    hash_table = {}
    tmp_gain_df = gain_df[gain_df['Gain']>0]
    columns2select = [20, 30, 40, 50, 100, 150, tmp_gain_df.shape[0]]

    for qtd in columns2select:
        atribute_columns = list(tmp_gain_df.head(qtd)['Variable'])
        tmp_df = train_X[atribute_columns]

        knn_model = KNeighborsClassifier(n_neighbors=3)

        knn_model.fit(tmp_df, train_y)

        knn_pred = knn_model.predict(test_X[atribute_columns])

        hash_table[qtd] = [accuracy_score(test_y, knn_pred), f1_score(test_y, knn_pred,

    return hash_table


def table_mi_knn():
    hash_table = selecting_best_mi_knn()

    for value in hash_table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(hash_table[value][0])
        print('F1:       ', end='')
        print(hash_table[value][1])
        print()

table_mi_knn()
```

```
Número de atributos selecionados:  20
Acurácia: 0.6444444444444445
F1:       0.5504629629629629

Número de atributos selecionados:  30
Acurácia: 0.6222222222222222
F1:       0.536957387495022

Número de atributos selecionados:  40
Acurácia: 0.6
F1:       0.5226984126984127

Número de atributos selecionados:  50
Acurácia: 0.6222222222222222
F1:       0.5167827529021559

Número de atributos selecionados:  100
Acurácia: 0.6444444444444445
F1:       0.5346932006633499

Número de atributos selecionados:  150
Acurácia: 0.6444444444444445
F1:       0.5696866096866097

Número de atributos selecionados:  175
Acurácia: 0.7111111111111111
F1:       0.6300371160072653
```

Logistic Regression

In [62]:

```
def selecting_best_mi_log():
    hash_table = {}
    tmp_gain_df = gain_df[gain_df['Gain']>0]
    columns2select = [20, 30, 40, 50, 100, 150, tmp_gain_df.shape[0]]

    for qtd in columns2select:
        atribute_columns = list(tmp_gain_df.head(qtd)['Variable'])
        tmp_df = train_X[atribute_columns]

        log_model = LogisticRegression(random_state=0,solver = "saga")

        log_model.fit(tmp_df, train_y)

        log_pred = log_model.predict(test_X[atribute_columns])

        hash_table[qtd] = [accuracy_score(test_y, log_pred), f1_score(test_y, log_pred,

    return hash_table

def table_mi_log():
    hash_table = selecting_best_mi_log()
    for value in hash_table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(hash_table[value][0])
        print('F1:       ', end='')
        print(hash_table[value][1])
        print()
table_mi_log()
```

```
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

```
Número de atributos selecionados:  20
Acurácia: 0.6666666666666666
F1:       0.5869907407407406

Número de atributos selecionados:  30
Acurácia: 0.6444444444444445
F1:       0.5788871821129885

Número de atributos selecionados:  40
Acurácia: 0.7111111111111111
F1:       0.643915343915344

Número de atributos selecionados:  50
Acurácia: 0.6888888888888889
F1:       0.6336976320582878

Número de atributos selecionados:  100
Acurácia: 0.7555555555555555
F1:       0.7159387797805877

Número de atributos selecionados:  150
Acurácia: 0.7555555555555555
F1:       0.737056530214425

Número de atributos selecionados:  175
Acurácia: 0.7777777777777778
F1:       0.7556632321544602
```

```
/home/kalai/anaconda3/lib/python3.9/site-packages/sklearn/linear_model/_sag.py:352: Conv
ergenceWarning: The max_iter was reached which means the coef_ did not converge
  warnings.warn(
```

Gaussian Naive Bayes

In [63]:

```python
def selecting_best_mi_nb():
    hash_table = {}
    tmp_gain_df = gain_df[gain_df['Gain']>0]
    columns2select = [20, 30, 40, 50, 100, 150, tmp_gain_df.shape[0]]

    for qtd in columns2select:
        atribute_columns = list(tmp_gain_df.head(qtd)['Variable'])
        tmp_df = train_X[atribute_columns]

        nb_model = GaussianNB(var_smoothing = 0.5) # var_smoothing foi decidido no 'olho
```

Loading [MathJax]/extensions/Safe.js

```
        nb_model.fit(tmp_df, train_y)

        nb_pred = nb_model.predict(test_X[atribute_columns])

        hash_table[qtd] = [accuracy_score(test_y, nb_pred), f1_score(test_y, nb_pred, av

    return hash_table

def table_mi_nb():
    hash_table = selecting_best_mi_nb()
    for value in hash_table:
        print('Número de atributos selecionados: ', value)
        print('Acurácia: ', end='')
        print(hash_table[value][0])
        print('F1:       ', end='')
        print(hash_table[value][1])
        print()
table_mi_nb()
```

```
Número de atributos selecionados:  20
Acurácia: 0.6222222222222222
F1:       0.4924867724867725

Número de atributos selecionados:  30
Acurácia: 0.6222222222222222
F1:       0.4924867724867725

Número de atributos selecionados:  40
Acurácia: 0.6444444444444445
F1:       0.5325281803542673

Número de atributos selecionados:  50
Acurácia: 0.6444444444444445
F1:       0.5140096618357488

Número de atributos selecionados:  100
Acurácia: 0.6888888888888889
F1:       0.5742483660130719

Número de atributos selecionados:  150
Acurácia: 0.7333333333333333
F1:       0.6521789321789322

Número de atributos selecionados:  175
Acurácia: 0.7777777777777778
F1:       0.7027932098765431
```