Dataset Description

The Data description is as follows:

1. diagnosis: The diagnosis of breast tissues (1 = malignant, 0 = benign) where malignant denotes that the disease is harmful
2. mean_radius: mean of distances from center to points on the perimeter
3. mean_texture: standard deviation of gray-scale values
4. mean_perimeter: mean size of the core tumor
5. mean_area: mean area of the core tumor
6. mean_smoothness: mean of local variation in radius lengths

All feature values are recoded with four significant digits.

In [ ]:
```
# This Python 3 environment comes with many helpful analytics libraries installe
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/doc
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.metrics import accuracy_score, precision_score, recall_score, roc_a
from sklearn.metrics import f1_score, confusion_matrix, precision_recall_curve,
from sklearn.metrics import ConfusionMatrixDisplay

from sklearn.model_selection import train_test_split
```

Carregando o dataset

In [ ]:
```
#data = pd.read_csv('Breast_cancer_data.csv')
data = pd.read_csv('data.csv')
data = data.drop('id', axis=1)
data['diagnosis'] = data['diagnosis'].map({'M':1, 'B':0})
data
```

Out[ ]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_n
0	1	17.99	10.38	122.80	1001.0	0.1
1	1	20.57	17.77	132.90	1326.0	0.0
2	1	19.69	21.25	130.00	1203.0	0.1
3	1	11.42	20.38	77.58	386.1	0.1
4	1	20.29	14.34	135.10	1297.0	0.1
...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.1
565	1	20.13	28.25	131.20	1261.0	0.0
566	1	16.60	28.08	108.30	858.1	0.0
567	1	20.60	29.33	140.10	1265.0	0.1
568	0	7.76	24.54	47.92	181.0	0.0

569 rows × 31 columns

Checando se existem valores nulos

In [ ]:
```
## check null entries
data.isnull().sum()
```

Out[ ]:
```
diagnosis                0
radius_mean              0
texture_mean             0
perimeter_mean           0
area_mean                0
smoothness_mean          0
compactness_mean         0
concavity_mean           0
concave points_mean      0
symmetry_mean            0
fractal_dimension_mean   0
radius_se                0
texture_se               0
perimeter_se             0
area_se                  0
smoothness_se            0
compactness_se           0
concavity_se             0
concave points_se        0
symmetry_se              0
fractal_dimension_se     0
radius_worst             0
texture_worst            0
perimeter_worst          0
area_worst               0
smoothness_worst         0
compactness_worst        0
concavity_worst          0
concave points_worst     0
symmetry_worst           0
fractal_dimension_worst  0
dtype: int64
```

Removendo ítens duplicados

In [ ]:
```
## remove duplicate entries
data.drop_duplicates(inplace = True)
data
```

Out[ ]:

	diagnosis	radius_mean	texture_mean	perimeter_mean	area_mean	smoothness_n
0	1	17.99	10.38	122.80	1001.0	0.1
1	1	20.57	17.77	132.90	1326.0	0.0
2	1	19.69	21.25	130.00	1203.0	0.1
3	1	11.42	20.38	77.58	386.1	0.1
4	1	20.29	14.34	135.10	1297.0	0.1
...	...	...	...	...	...	
564	1	21.56	22.39	142.00	1479.0	0.1
565	1	20.13	28.25	131.20	1261.0	0.0
566	1	16.60	28.08	108.30	858.1	0.0
567	1	20.60	29.33	140.10	1265.0	0.1
568	0	7.76	24.54	47.92	181.0	0.0

569 rows × 31 columns

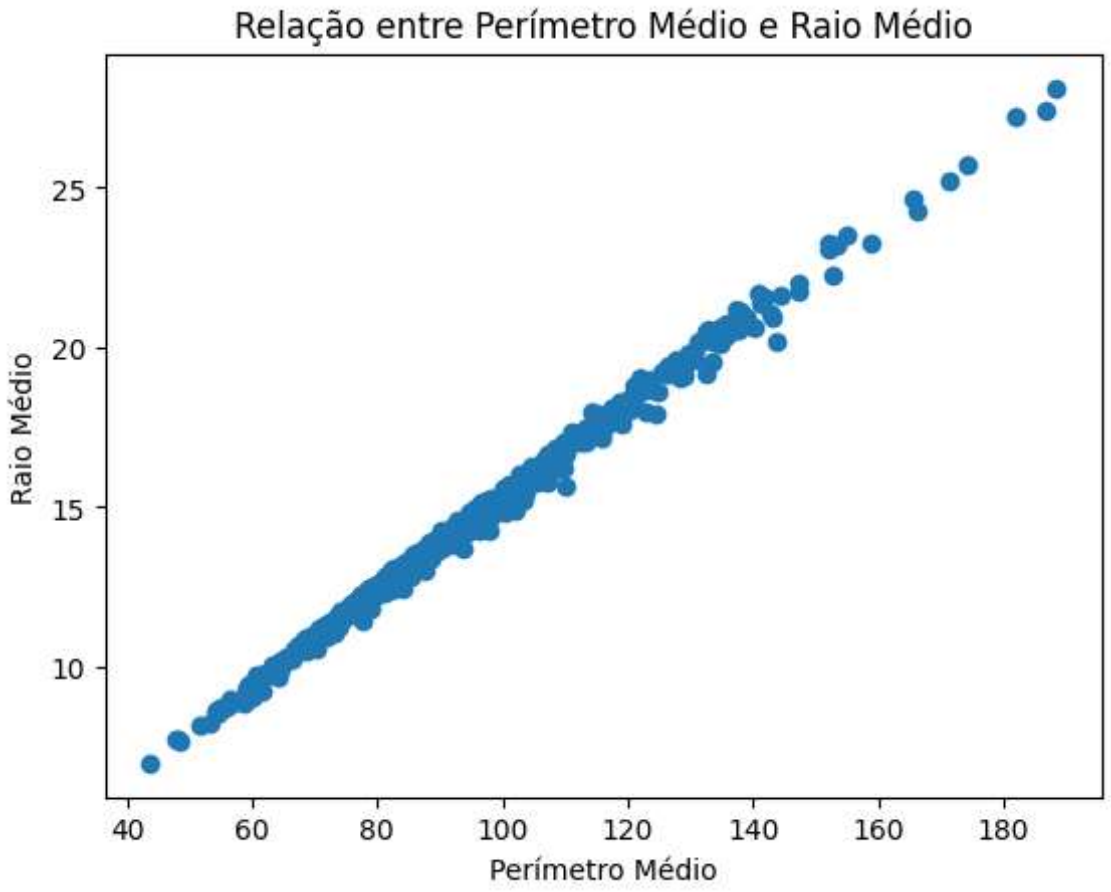
Porque temos Raio, Perímetro e Área como features independetes? Não são features correlacionadas? Vamos investigar!

Perímetro X Raio

In [ ]:
```
print(f"Correlação entre o perímetro médio e o raio médio: {data['perimeter_mean
```

```
plt.scatter(data.perimeter_mean, data.radius_mean)
plt.xlabel("Perímetro Médio")
plt.ylabel("Raio Médio")
plt.title("Relação entre Perímetro Médio e Raio Médio")
plt.show()
```

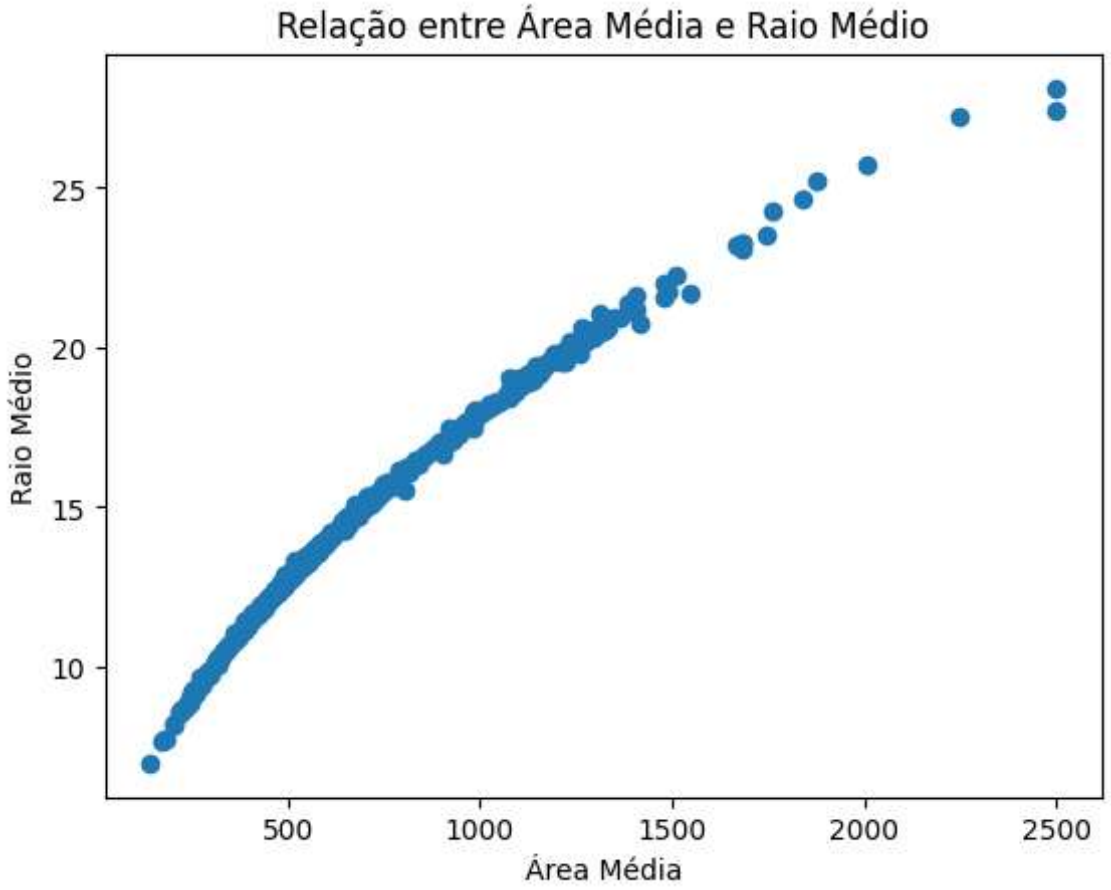
```
Correlação entre o perímetro médio e o raio médio: 0.9978552814938106
```



Área x Raio

In [ ]:
```
print(f"Correlação entre o área média e o raio médio: {data['area_mean'].corr(da

plt.scatter(data.area_mean, data.radius_mean)
plt.xlabel("Área Média")
plt.ylabel("Raio Médio")
plt.title("Relação entre Área Média e Raio Médio")
plt.show()
```

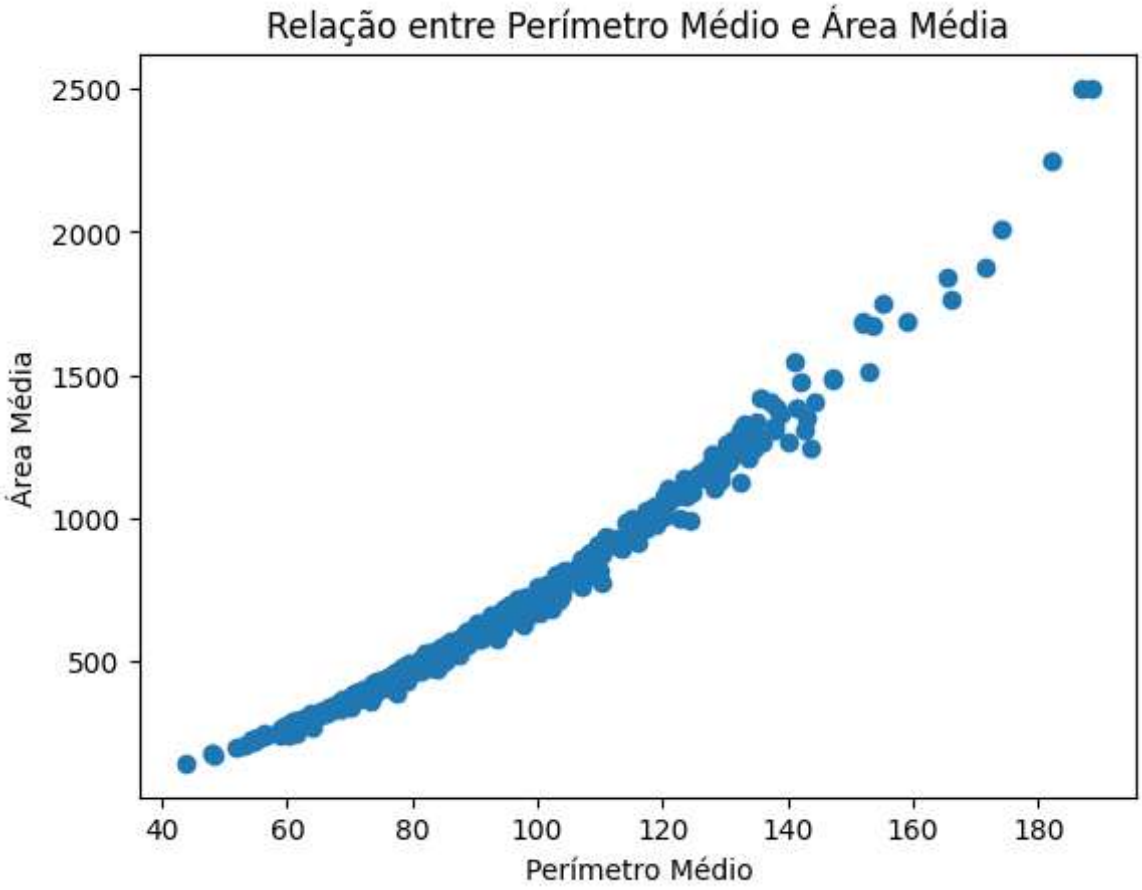
```
Correlação entre o área média e o raio médio: 0.9873571700566127
```



Perímetro x Área

In [ ]:
```
print(f"Correlação entre o perímetro médio e o área média: {data['perimeter_mear


plt.scatter(data.perimeter_mean, data.area_mean)
plt.xlabel("Perímetro Médio")
plt.ylabel("Área Média")
plt.title("Relação entre Perímetro Médio e Área Média")
plt.show()
```

```
Correlação entre o perímetro médio e o área média: 0.9865068039913906
```



Vamos então descartar a coluna "Perímetro" e a coluna "Área"

- Estas duas colunas não complementam a variabilidade dos dados e só dificultarão o treinamento do modelo
- Retirando essas colunas nós também retiramos a necessidade de obtê-las para uma inferência futura

In [ ]:
```
data.drop(['perimeter_mean'], axis=1, inplace=True)
data.drop(['area_mean'], axis=1, inplace=True)
data
```

Out[ ]:

	diagnosis	radius_mean	texture_mean	smoothness_mean	compactness_mean	con
0	1	17.99	10.38	0.11840	0.27760	
1	1	20.57	17.77	0.08474	0.07864	
2	1	19.69	21.25	0.10960	0.15990	
3	1	11.42	20.38	0.14250	0.28390	
4	1	20.29	14.34	0.10030	0.13280	
...	...	...	...	...	...	
564	1	21.56	22.39	0.11100	0.11590	
565	1	20.13	28.25	0.09780	0.10340	
566	1	16.60	28.08	0.08455	0.10230	
567	1	20.60	29.33	0.11780	0.27700	
568	0	7.76	24.54	0.05263	0.04362	

569 rows × 29 columns

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
#X_train = sc.fit_transform(X_train)
#X_test = sc.transform(X_test)
```

Tudo certo! Agora vamos separar os dados de treinamento e de validação

In [ ]:
```
from sklearn.model_selection import train_test_split

X = data.iloc[:,1:].values
y = data.iloc[:,0].values

X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.8, random

print(f"O conjunto de treinamento contém {len(X_train)} ítens e o conjunto de te
```

```
O conjunto de treinamento contém 455 ítens e o conjunto de teste contém 114 iten
s.
```

Tudo certo mesmo? Vamos dar uma olhada nos valores médios de cada coluna para tentar entender

In [ ]:
```
data.mean()
```

Out[ ]:
```
diagnosis                 0.372583
radius_mean              14.127292
texture_mean             19.289649
smoothness_mean           0.096360
compactness_mean          0.104341
concavity_mean            0.088799
concave points_mean       0.048919
symmetry_mean             0.181162
fractal_dimension_mean    0.062798
radius_se                 0.405172
texture_se                1.216853
perimeter_se              2.866059
area_se                  40.337079
smoothness_se             0.007041
compactness_se            0.025478
concavity_se              0.031894
concave points_se         0.011796
symmetry_se               0.020542
fractal_dimension_se      0.003795
radius_worst             16.269190
texture_worst            25.677223
perimeter_worst         107.261213
area_worst              880.583128
smoothness_worst          0.132369
compactness_worst         0.254265
concavity_worst           0.272188
concave points_worst      0.114606
symmetry_worst            0.290076
fractal_dimension_worst   0.083946
dtype: float64
```

Vamos realizar uma trasnformação nos dados para que fiquem com a média próxima de zero e amplitudes mais próximas

In [ ]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [ ]:
```
pd.DataFrame(X_train, columns = data.columns[1:]).mean()
```

Out[ ]:
```
radius_mean               -8.976946e-16
texture_mean              -3.351654e-15
smoothness_mean           -1.465128e-15
compactness_mean           5.317404e-16
concavity_mean             6.614977e-16
concave points_mean        4.760539e-16
symmetry_mean             -5.686782e-15
fractal_dimension_mean     6.948044e-16
radius_se                  4.904502e-17
texture_se                 5.768280e-16
perimeter_se               8.107068e-16
area_se                    1.828818e-16
smoothness_se              1.791607e-15
compactness_se             4.155406e-16
concavity_se               3.255028e-16
concave points_se          6.022655e-16
symmetry_se               -1.517711e-16
fractal_dimension_se       1.345810e-15
radius_worst              -4.289609e-16
texture_worst              4.924022e-16
perimeter_worst           -1.517955e-15
area_worst                 1.324947e-16
smoothness_worst           5.923375e-15
compactness_worst         -4.703693e-16
concavity_worst           -1.369234e-15
concave points_worst       2.563273e-16
symmetry_worst            -2.123332e-15
fractal_dimension_worst    1.992423e-15
dtype: float64
```

Agora sim, tudo pronto!

Vamos instanciar as métricas para podermos avaliar os modelos posteriormente. Relembrando:

- Acertos
  - "True Positive" (TP): Previsão --> Maligno ; Real --> Maligno
  - "True Negative" (TN): Previsão --> Benigno ; Real --> Benigno

- Erros
  - "False Positive" (FP): Previsão --> Maligno ; Real --> Benigno
  - "False Negative" (FN): Previsão --> Benigno ; Real --> Maligno

Métricas:

- Acurácia = $\frac{T_P+T_N}{T_P+T_N+F_P+F_N}$

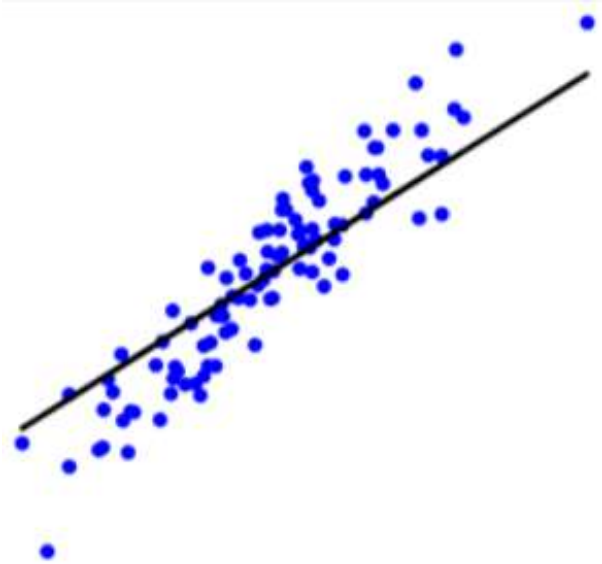
- Recall = $\frac{T_P}{T_P+F_N}$

- Precisão = $\frac{T_P}{T_P+F_P}$

- F1 - Score = $\frac{2P_{rec}R_{ec}}{P_{rec}+R_{ec}}$

In [ ]:
```
from sklearn.metrics import confusion_matrix, accuracy_score, recall_score, prec

acc = {}
rec = {}
prec = {}
f1 = {}
```

Regressão linear

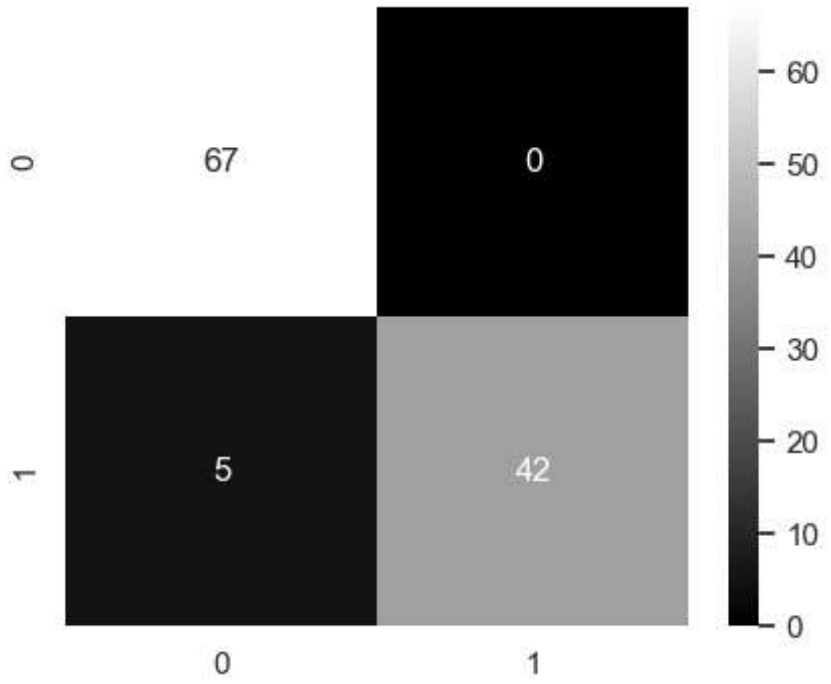


In [ ]:
```
# Importando o módulo de regressão linear
from sklearn.linear_model import LinearRegression

# Treinando o modelo
regr = LinearRegression()
regr.fit(X_train, y_train)


# Prevendo os resultados
y_pred = regr.predict(X_test)
y_pred = [1 if x >= 0.5 else 0 for x in y_pred]


# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)

acc['reg_linear'] = accuracy_score(y_test, y_pred)
rec['reg_linear'] = recall_score(y_test, y_pred)
prec['reg_linear'] = precision_score(y_test, y_pred)
f1['reg_linear'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100*acc['reg_linear']} %")
print(f"Recall: {100*rec['reg_linear']} %")
print(f"Precisão: {100*prec['reg_linear']} %")
print(f"F1: {100*f1['reg_linear']} %")

print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
```

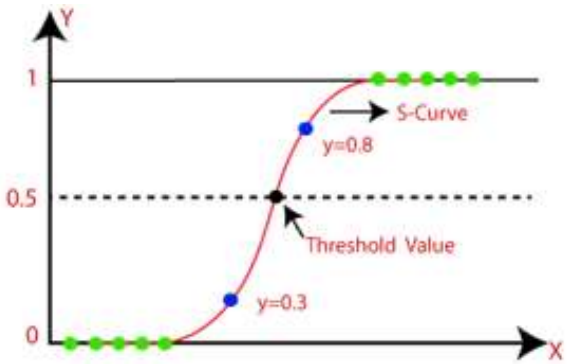
```
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
Acurácia: 95.6140350877193 %
Recall: 89.36170212765957 %
Precisão: 100.0 %
F1: 94.3820224719101 %
```

Matriz de confusão:



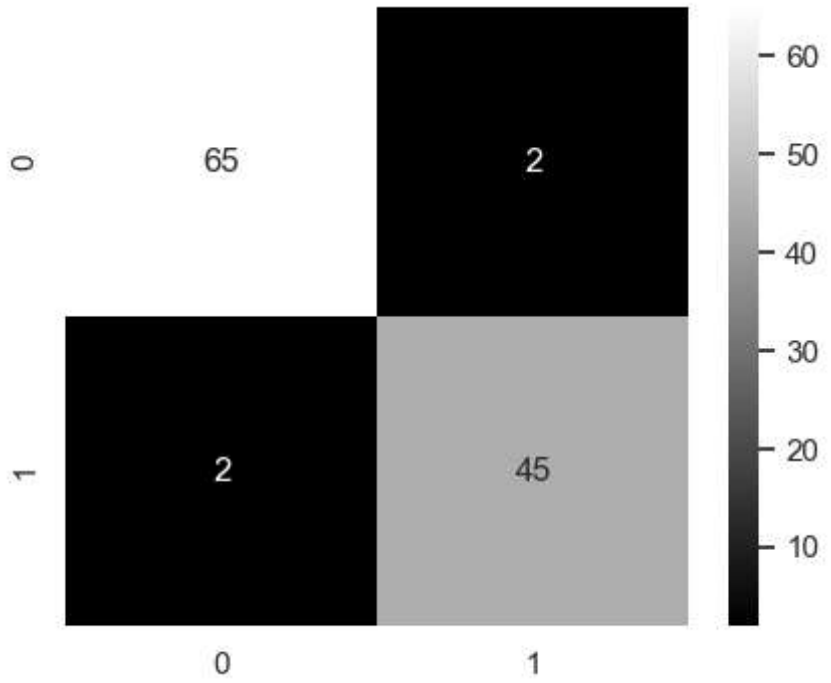
Regressão logística



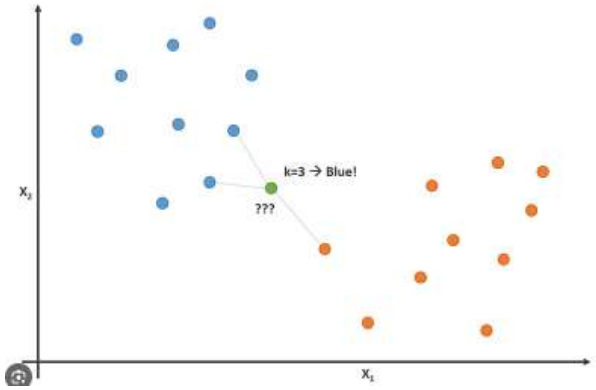
In [ ]:
```
# Treinando modelo
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression()
classifier.fit(X_train, y_train)


# Prevendo os resultados
y_pred = classifier.predict(X_test)



# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)
```

```
acc['logistic'] = accuracy_score(y_test, y_pred)
rec['logistic'] = recall_score(y_test, y_pred)
prec['logistic'] = precision_score(y_test, y_pred)
f1['logistic'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100*acc['logistic']} %")
print(f"Recall: {100*rec['logistic']} %")
print(f"Precisão: {100*prec['logistic']} %")
print(f"F1: {100*f1['logistic']} %")


print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
Acurácia: 96.49122807017544 %
Recall: 95.74468085106383 %
Precisão: 95.74468085106383 %
F1: 95.74468085106385 %


Matriz de confusão:
```

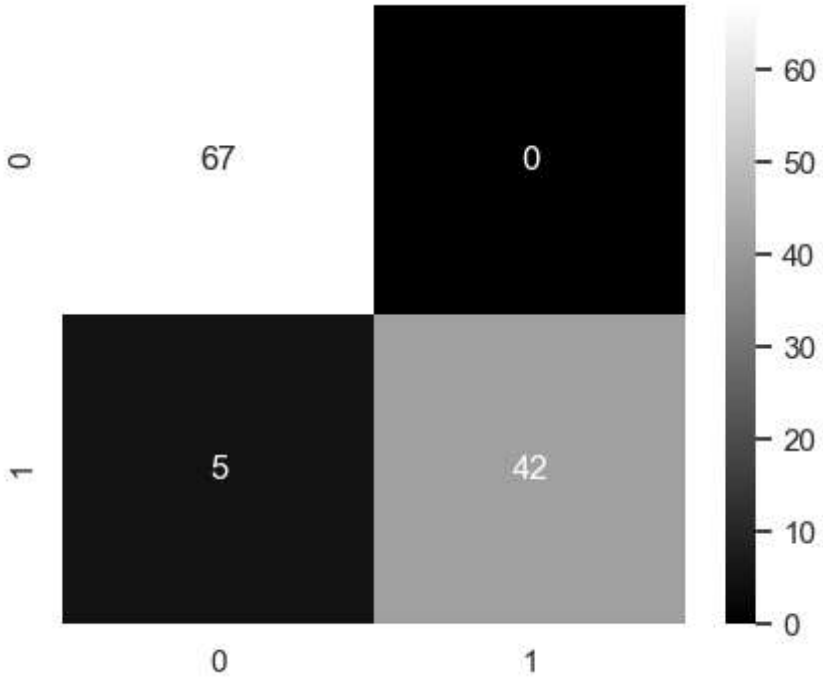


K Neighbors (vizinho mais próximo)

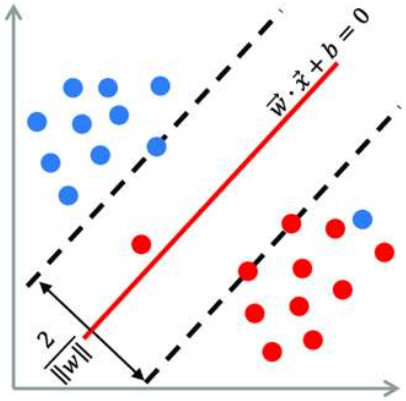


In [ ]:
```
# Treinando modelo
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 5)
knn.fit(X_train, y_train)

# Prevendo os resultados
y_pred = knn.predict(X_test)

# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)

acc['k_neighbors'] = accuracy_score(y_test, y_pred)
rec['k_neighbors'] = recall_score(y_test, y_pred)
prec['k_neighbors'] = precision_score(y_test, y_pred)
f1['k_neighbors'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100* acc['k_neighbors']} %")
print(f"Recall: {100* rec['k_neighbors']} %")
print(f"Precisão: {100* prec['k_neighbors']} %")
print(f"F1: {100* f1['k_neighbors']} %")


print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
Acurácia: 95.6140350877193 %
Recall: 89.36170212765957 %
Precisão: 100.0 %
F1: 94.3820224719101 %


Matriz de confusão:
```



Linear SVM

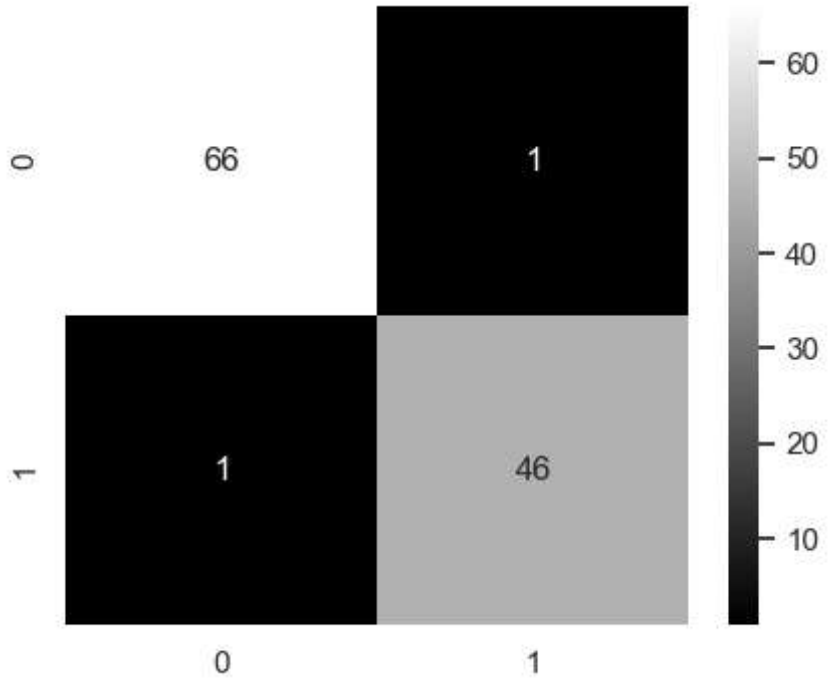


In [ ]:
```
# Treinando modelo
from sklearn.svm import SVC
svm = SVC(kernel = 'linear')
svm.fit(X_train, y_train)

# Prevendo os resultados
y_pred = svm.predict(X_test)


# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)

acc['SVM_linear'] = accuracy_score(y_test, y_pred)
rec['SVM_linear'] = recall_score(y_test, y_pred)
prec['SVM_linear'] = precision_score(y_test, y_pred)
f1['SVM_linear'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100*acc['SVM_linear']} %")
print(f"Recall: {100*rec['SVM_linear']} %")
print(f"Precisão: {100*prec['SVM_linear']} %")
print(f"F1: {100*f1['SVM_linear']} %")
```

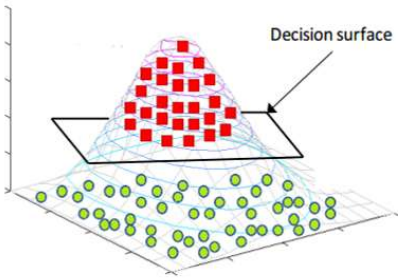
```
print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
Acurácia: 98.24561403508771 %
Recall: 97.87234042553192 %
Precisão: 97.87234042553192 %
F1: 97.87234042553192 %
```

```
Matriz de confusão:
```



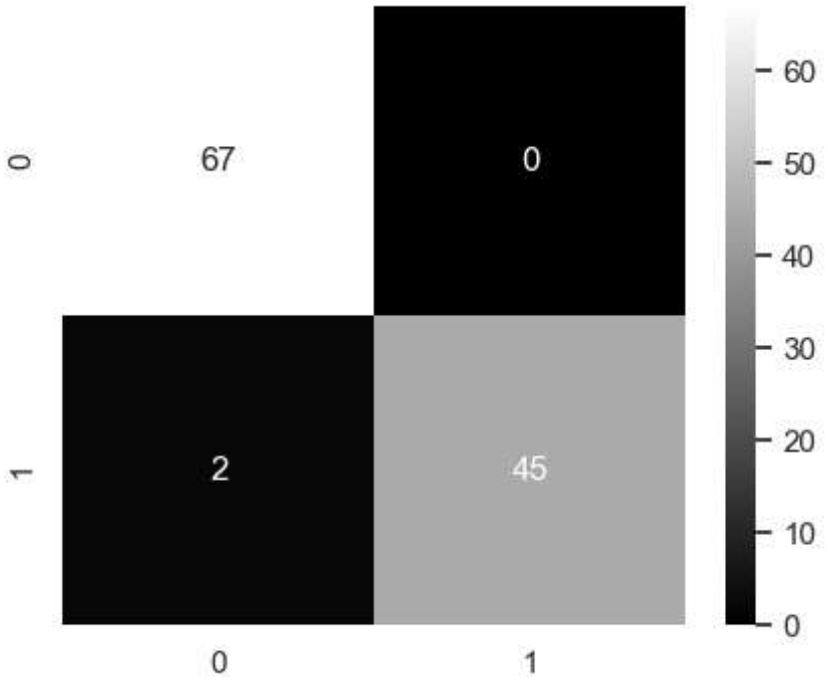
Kernal SVM



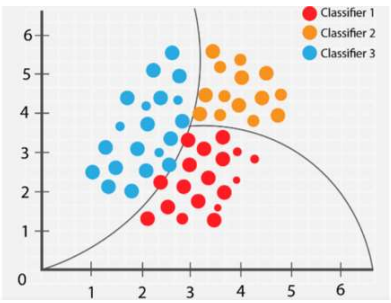
In [ ]:
```
# Treinando modelo
from sklearn.svm import SVC
kernel_svm = SVC(kernel = 'rbf')
kernel_svm.fit(X_train, y_train)

# Prevendo os resultados
y_pred = kernel_svm.predict(X_test)
```

```
# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)

acc['SVM_kernal'] = accuracy_score(y_test, y_pred)
rec['SVM_kernal'] = recall_score(y_test, y_pred)
prec['SVM_kernal'] = precision_score(y_test, y_pred)
f1['SVM_kernal'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100*acc['SVM_kernal']} %")
print(f"Recall: {100*rec['SVM_kernal']} %")
print(f"Precisão: {100*prec['SVM_kernal']} %")
print(f"F1: {100*f1['SVM_kernal']} %")


print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
Acurácia: 98.24561403508771 %
Recall: 95.74468085106383 %
Precisão: 100.0 %
F1: 97.82608695652173 %


Matriz de confusão:
```

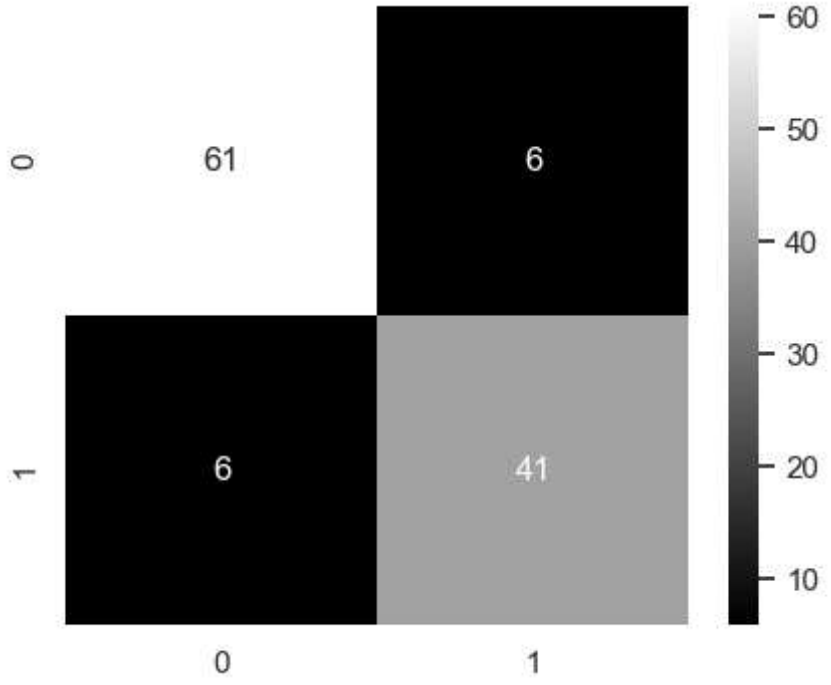


Naive Bayes (Redes Bayesianas)

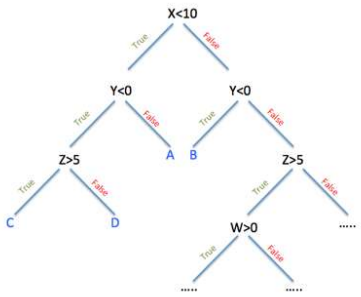


In [ ]:
```
# Treinando modelo
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(X_train, y_train)

# Prevendo os resultados
y_pred = gnb.predict(X_test)


# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)

acc['naive_bayes'] = accuracy_score(y_test, y_pred)
rec['naive_bayes'] = recall_score(y_test, y_pred)
prec['naive_bayes'] = precision_score(y_test, y_pred)
f1['naive_bayes'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100*acc['naive_bayes']} %")
print(f"Recall: {100*rec['naive_bayes']} %")
print(f"Precisão: {100*prec['naive_bayes']} %")
print(f"F1: {100*f1['naive_bayes']} %")


print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
Acurácia: 89.47368421052632 %
Recall: 87.2340425531915 %
Precisão: 87.2340425531915 %
F1: 87.2340425531915 %


Matriz de confusão:
```

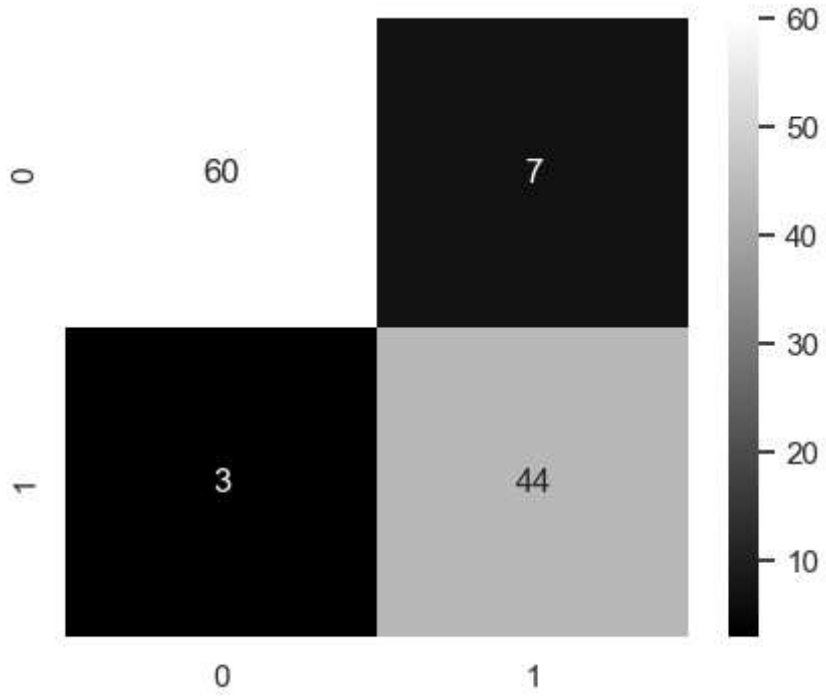


Decision Tree Classifier (árvore de decisão)

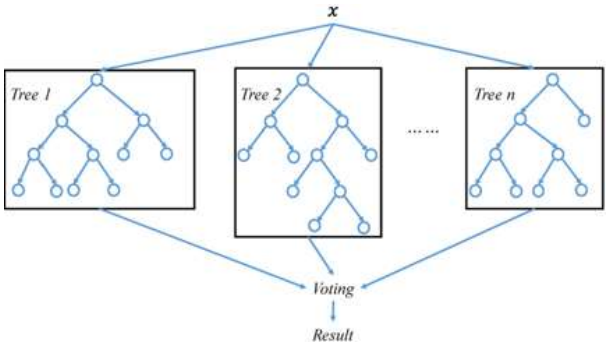


In [ ]:
```
# Treinando modelo
from sklearn.tree import DecisionTreeClassifier
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)

# Prevendo os resultados
y_pred = dtc.predict(X_test)

# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)

acc['tree'] = accuracy_score(y_test, y_pred)
rec['tree'] = recall_score(y_test, y_pred)
prec['tree'] = precision_score(y_test, y_pred)
f1['tree'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100*acc['tree']} %")
print(f"Recall: {100*rec['tree']} %")
print(f"Precisão: {100*prec['tree']} %")
print(f"F1: {100*f1['tree']} %")


print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
Acurácia: 91.22807017543859 %
Recall: 93.61702127659575 %
Precisão: 86.27450980392157 %
F1: 89.79591836734694 %


Matriz de confusão:
```



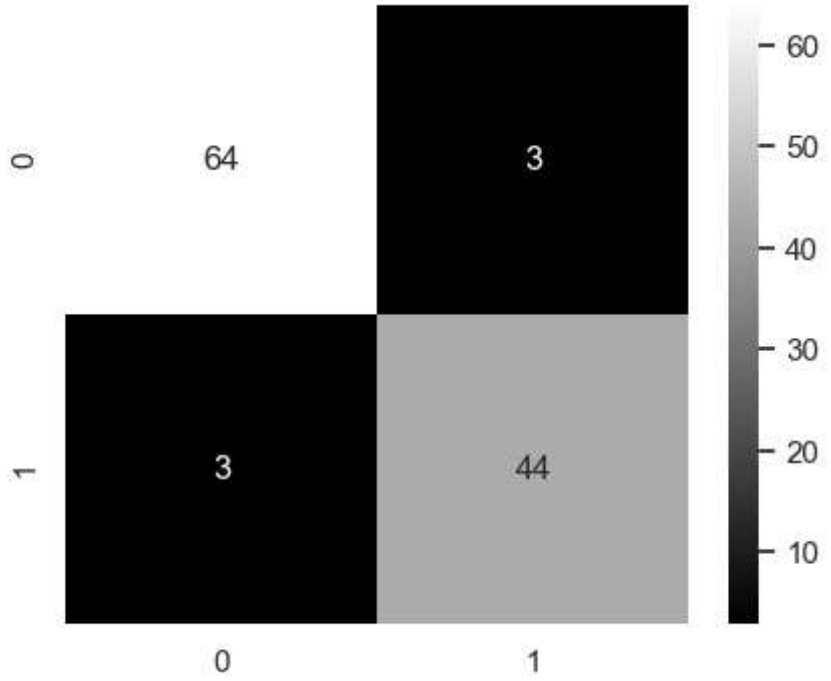
Random Forest



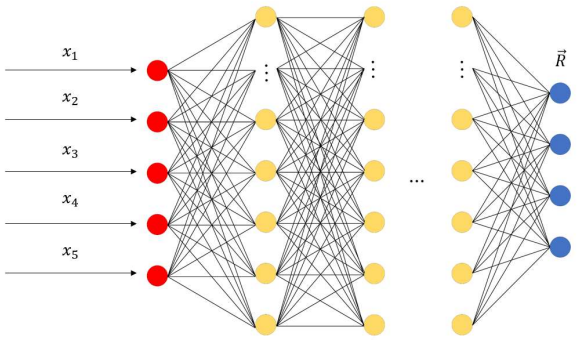
In [ ]:
```
# Treinando modelo
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(n_estimators = 10, criterion = 'entropy')
rfc.fit(X_train, y_train)

# Prevendo os resultados
y_pred = rfc.predict(X_test)

# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)



acc['random_forest'] = accuracy_score(y_test, y_pred)
rec['random_forest'] = recall_score(y_test, y_pred)
prec['random_forest'] = precision_score(y_test, y_pred)
f1['random_forest'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100*acc['random_forest']} %")
print(f"Recall: {100*rec['random_forest']} %")
print(f"Precisão: {100*prec['random_forest']} %")
print(f"F1: {100*f1['random_forest']} %")
```

```
print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
Acurácia: 94.73684210526315 %
Recall: 93.61702127659575 %
Precisão: 93.61702127659575 %
F1: 93.61702127659575 %


Matriz de confusão:
```



Artificial Neural Network (ANN)



In [ ]:
```
# Importando módulos
import tensorflow as tf
import keras
from keras.layers import Dense, Dropout
from keras.models import Sequential


# Criando modelo
model = Sequential()
model.add(Dense(units = 3, activation = 'relu'))
```

```
model.add(Dropout(0.25))
model.add(Dense(units = 3, activation = 'relu'))
model.add(Dropout(0.25))
model.add(Dense(units = 1, activation = 'sigmoid'))

# Compilando modelo
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accu
```

In [ ]:

```
# Treinando o modelo
from keras.callbacks import EarlyStopping
early_stop = EarlyStopping(monitor='val_loss', patience=5, verbose=1)
deep_history = model.fit(X_train, y_train, epochs=100, batch_size = 32, validati

model_path = 'modelo.h5'
model.save(model_path)

model_weights_path= 'modelo_weights.h5'
model.save_weights(model_weights_path)
```

```
Epoch 1/100
15/15 [==============================] - 1s 11ms/step - loss: 0.7002 - accuracy:
0.5077 - val_loss: 0.6850 - val_accuracy: 0.5351
Epoch 2/100
15/15 [==============================] - 0s 3ms/step - loss: 0.7005 - accuracy:
0.5275 - val_loss: 0.6731 - val_accuracy: 0.7456
Epoch 3/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6906 - accuracy:
0.6527 - val_loss: 0.6639 - val_accuracy: 0.7982
Epoch 4/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6833 - accuracy:
0.6857 - val_loss: 0.6557 - val_accuracy: 0.8070
Epoch 5/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6560 - accuracy:
0.7407 - val_loss: 0.6454 - val_accuracy: 0.8246
Epoch 6/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6582 - accuracy:
0.7209 - val_loss: 0.6323 - val_accuracy: 0.8509
Epoch 7/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6458 - accuracy:
0.7604 - val_loss: 0.6195 - val_accuracy: 0.8596
Epoch 8/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6207 - accuracy:
0.7626 - val_loss: 0.6035 - val_accuracy: 0.8684
Epoch 9/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6140 - accuracy:
0.7714 - val_loss: 0.5855 - val_accuracy: 0.8860
Epoch 10/100
15/15 [==============================] - 0s 3ms/step - loss: 0.6079 - accuracy:
0.7890 - val_loss: 0.5702 - val_accuracy: 0.9035
Epoch 11/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5905 - accuracy:
0.7824 - val_loss: 0.5537 - val_accuracy: 0.9035
Epoch 12/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5761 - accuracy:
0.8066 - val_loss: 0.5362 - val_accuracy: 0.9035
Epoch 13/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5608 - accuracy:
0.7956 - val_loss: 0.5178 - val_accuracy: 0.9123
Epoch 14/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5371 - accuracy:
0.8110 - val_loss: 0.4984 - val_accuracy: 0.9211
Epoch 15/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5364 - accuracy:
0.8066 - val_loss: 0.4785 - val_accuracy: 0.9298
Epoch 16/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5090 - accuracy:
0.8220 - val_loss: 0.4589 - val_accuracy: 0.9298
Epoch 17/100
15/15 [==============================] - 0s 3ms/step - loss: 0.5099 - accuracy:
0.8088 - val_loss: 0.4429 - val_accuracy: 0.9211
Epoch 18/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4923 - accuracy:
0.8264 - val_loss: 0.4243 - val_accuracy: 0.9298
Epoch 19/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4697 - accuracy:
0.8308 - val_loss: 0.4050 - val_accuracy: 0.9298
Epoch 20/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4525 - accuracy:
0.8440 - val_loss: 0.3851 - val_accuracy: 0.9298
```

```
Epoch 21/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4286 - accuracy:
0.8571 - val_loss: 0.3666 - val_accuracy: 0.9298
Epoch 22/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4283 - accuracy:
0.8110 - val_loss: 0.3479 - val_accuracy: 0.9474
Epoch 23/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4255 - accuracy:
0.8286 - val_loss: 0.3320 - val_accuracy: 0.9561
Epoch 24/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4056 - accuracy:
0.8571 - val_loss: 0.3178 - val_accuracy: 0.9561
Epoch 25/100
15/15 [==============================] - 0s 3ms/step - loss: 0.4061 - accuracy:
0.8352 - val_loss: 0.3041 - val_accuracy: 0.9561
Epoch 26/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3752 - accuracy:
0.8527 - val_loss: 0.2889 - val_accuracy: 0.9474
Epoch 27/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3705 - accuracy:
0.8637 - val_loss: 0.2749 - val_accuracy: 0.9474
Epoch 28/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3874 - accuracy:
0.8242 - val_loss: 0.2630 - val_accuracy: 0.9474
Epoch 29/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3762 - accuracy:
0.8396 - val_loss: 0.2529 - val_accuracy: 0.9474
Epoch 30/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3765 - accuracy:
0.8374 - val_loss: 0.2447 - val_accuracy: 0.9474
Epoch 31/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3399 - accuracy:
0.8549 - val_loss: 0.2363 - val_accuracy: 0.9561
Epoch 32/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3309 - accuracy:
0.8681 - val_loss: 0.2289 - val_accuracy: 0.9561
Epoch 33/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3102 - accuracy:
0.8505 - val_loss: 0.2196 - val_accuracy: 0.9561
Epoch 34/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3373 - accuracy:
0.8440 - val_loss: 0.2121 - val_accuracy: 0.9561
Epoch 35/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3421 - accuracy:
0.8637 - val_loss: 0.2044 - val_accuracy: 0.9561
Epoch 36/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3292 - accuracy:
0.8571 - val_loss: 0.1976 - val_accuracy: 0.9561
Epoch 37/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3310 - accuracy:
0.8484 - val_loss: 0.1923 - val_accuracy: 0.9561
Epoch 38/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3336 - accuracy:
0.8615 - val_loss: 0.1867 - val_accuracy: 0.9561
Epoch 39/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3256 - accuracy:
0.8571 - val_loss: 0.1808 - val_accuracy: 0.9561
Epoch 40/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3316 - accuracy:
0.8615 - val_loss: 0.1756 - val_accuracy: 0.9561
```

```
Epoch 41/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3010 - accuracy:
0.8813 - val_loss: 0.1715 - val_accuracy: 0.9561
Epoch 42/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3289 - accuracy:
0.8593 - val_loss: 0.1671 - val_accuracy: 0.9561
Epoch 43/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2980 - accuracy:
0.8681 - val_loss: 0.1620 - val_accuracy: 0.9561
Epoch 44/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3068 - accuracy:
0.8659 - val_loss: 0.1585 - val_accuracy: 0.9561
Epoch 45/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3082 - accuracy:
0.8505 - val_loss: 0.1545 - val_accuracy: 0.9649
Epoch 46/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3050 - accuracy:
0.8615 - val_loss: 0.1505 - val_accuracy: 0.9649
Epoch 47/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3019 - accuracy:
0.8725 - val_loss: 0.1475 - val_accuracy: 0.9649
Epoch 48/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3152 - accuracy:
0.8593 - val_loss: 0.1438 - val_accuracy: 0.9649
Epoch 49/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2837 - accuracy:
0.8769 - val_loss: 0.1400 - val_accuracy: 0.9649
Epoch 50/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3045 - accuracy:
0.8747 - val_loss: 0.1372 - val_accuracy: 0.9649
Epoch 51/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2789 - accuracy:
0.8945 - val_loss: 0.1340 - val_accuracy: 0.9649
Epoch 52/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3008 - accuracy:
0.8747 - val_loss: 0.1312 - val_accuracy: 0.9649
Epoch 53/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2723 - accuracy:
0.8879 - val_loss: 0.1287 - val_accuracy: 0.9649
Epoch 54/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2895 - accuracy:
0.8681 - val_loss: 0.1258 - val_accuracy: 0.9649
Epoch 55/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2837 - accuracy:
0.8725 - val_loss: 0.1234 - val_accuracy: 0.9737
Epoch 56/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3039 - accuracy:
0.8571 - val_loss: 0.1214 - val_accuracy: 0.9737
Epoch 57/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2847 - accuracy:
0.8681 - val_loss: 0.1199 - val_accuracy: 0.9737
Epoch 58/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2973 - accuracy:
0.8593 - val_loss: 0.1177 - val_accuracy: 0.9737
Epoch 59/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2741 - accuracy:
0.8725 - val_loss: 0.1169 - val_accuracy: 0.9737
Epoch 60/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2865 - accuracy:
0.8637 - val_loss: 0.1151 - val_accuracy: 0.9737
```

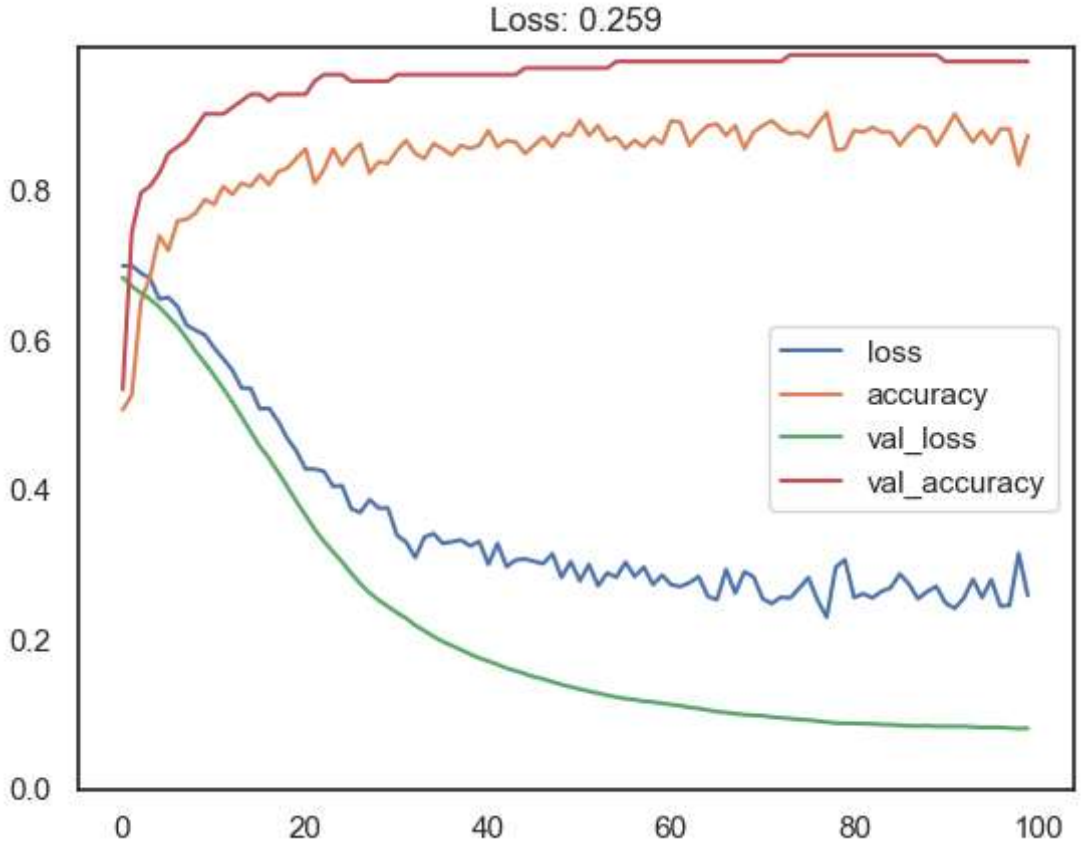
```
Epoch 61/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2736 - accuracy:
0.8945 - val_loss: 0.1133 - val_accuracy: 0.9737
Epoch 62/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2713 - accuracy:
0.8923 - val_loss: 0.1118 - val_accuracy: 0.9737
Epoch 63/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2764 - accuracy:
0.8615 - val_loss: 0.1095 - val_accuracy: 0.9737
Epoch 64/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2847 - accuracy:
0.8769 - val_loss: 0.1082 - val_accuracy: 0.9737
Epoch 65/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2582 - accuracy:
0.8879 - val_loss: 0.1060 - val_accuracy: 0.9737
Epoch 66/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2536 - accuracy:
0.8901 - val_loss: 0.1039 - val_accuracy: 0.9737
Epoch 67/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2941 - accuracy:
0.8747 - val_loss: 0.1028 - val_accuracy: 0.9737
Epoch 68/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2625 - accuracy:
0.8879 - val_loss: 0.1011 - val_accuracy: 0.9737
Epoch 69/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2911 - accuracy:
0.8571 - val_loss: 0.0998 - val_accuracy: 0.9737
Epoch 70/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2843 - accuracy:
0.8791 - val_loss: 0.0991 - val_accuracy: 0.9737
Epoch 71/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2550 - accuracy:
0.8879 - val_loss: 0.0984 - val_accuracy: 0.9737
Epoch 72/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2486 - accuracy:
0.8945 - val_loss: 0.0967 - val_accuracy: 0.9737
Epoch 73/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2570 - accuracy:
0.8835 - val_loss: 0.0955 - val_accuracy: 0.9737
Epoch 74/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2559 - accuracy:
0.8769 - val_loss: 0.0948 - val_accuracy: 0.9825
Epoch 75/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2690 - accuracy:
0.8791 - val_loss: 0.0934 - val_accuracy: 0.9825
Epoch 76/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2833 - accuracy:
0.8725 - val_loss: 0.0928 - val_accuracy: 0.9825
Epoch 77/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2531 - accuracy:
0.8901 - val_loss: 0.0914 - val_accuracy: 0.9825
Epoch 78/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2303 - accuracy:
0.9055 - val_loss: 0.0901 - val_accuracy: 0.9825
Epoch 79/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2971 - accuracy:
0.8549 - val_loss: 0.0886 - val_accuracy: 0.9825
Epoch 80/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3072 - accuracy:
0.8571 - val_loss: 0.0884 - val_accuracy: 0.9825
```

```
Epoch 81/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2567 - accuracy:
0.8813 - val_loss: 0.0883 - val_accuracy: 0.9825
Epoch 82/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2617 - accuracy:
0.8791 - val_loss: 0.0879 - val_accuracy: 0.9825
Epoch 83/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2561 - accuracy:
0.8857 - val_loss: 0.0875 - val_accuracy: 0.9825
Epoch 84/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2649 - accuracy:
0.8791 - val_loss: 0.0867 - val_accuracy: 0.9825
Epoch 85/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2695 - accuracy:
0.8791 - val_loss: 0.0865 - val_accuracy: 0.9825
Epoch 86/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2882 - accuracy:
0.8615 - val_loss: 0.0856 - val_accuracy: 0.9825
Epoch 87/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2746 - accuracy:
0.8769 - val_loss: 0.0848 - val_accuracy: 0.9825
Epoch 88/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2550 - accuracy:
0.8879 - val_loss: 0.0847 - val_accuracy: 0.9825
Epoch 89/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2647 - accuracy:
0.8835 - val_loss: 0.0850 - val_accuracy: 0.9825
Epoch 90/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2716 - accuracy:
0.8615 - val_loss: 0.0842 - val_accuracy: 0.9825
Epoch 91/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2491 - accuracy:
0.8813 - val_loss: 0.0844 - val_accuracy: 0.9737
Epoch 92/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2422 - accuracy:
0.9033 - val_loss: 0.0842 - val_accuracy: 0.9737
Epoch 93/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2557 - accuracy:
0.8835 - val_loss: 0.0843 - val_accuracy: 0.9737
Epoch 94/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2810 - accuracy:
0.8659 - val_loss: 0.0835 - val_accuracy: 0.9737
Epoch 95/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2566 - accuracy:
0.8813 - val_loss: 0.0828 - val_accuracy: 0.9737
Epoch 96/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2805 - accuracy:
0.8637 - val_loss: 0.0829 - val_accuracy: 0.9737
Epoch 97/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2452 - accuracy:
0.8835 - val_loss: 0.0830 - val_accuracy: 0.9737
Epoch 98/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2461 - accuracy:
0.8835 - val_loss: 0.0818 - val_accuracy: 0.9737
Epoch 99/100
15/15 [==============================] - 0s 3ms/step - loss: 0.3156 - accuracy:
0.8352 - val_loss: 0.0814 - val_accuracy: 0.9737
Epoch 100/100
15/15 [==============================] - 0s 3ms/step - loss: 0.2593 - accuracy:
0.8747 - val_loss: 0.0817 - val_accuracy: 0.9737
```

In [ ]:
```
## Load model
#model.load_weights(top_model_weights_path)
```

In [ ]:
```
def plot_loss(history):
    historydata = pd.DataFrame(history.history, index=history.epoch)
    plt.figure(figsize=(8, 6))
    historydata.plot(ylim=(0, 1.01*historydata.values.max()))
    plt.title('Loss: %.3f' % history.history['loss'][-1])
plot_loss(deep_history)
```

```
<Figure size 800x600 with 0 Axes>
```

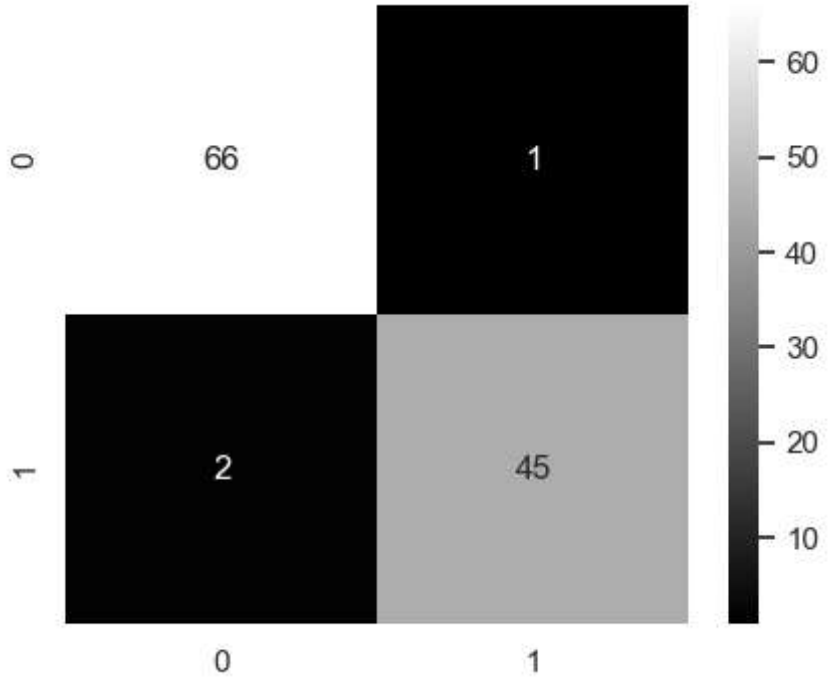


In [ ]:
```
# Prevendo os resultados
y_pred = model.predict(X_test)
y_pred = (y_pred>0.5)


# Analisando a qualidade do modelo
cm = confusion_matrix(y_test, y_pred)

acc['ANN'] = accuracy_score(y_test, y_pred)
rec['ANN'] = recall_score(y_test, y_pred)
prec['ANN'] = precision_score(y_test, y_pred)
f1['ANN'] = f1_score(y_test, y_pred)
print(f"Acurácia: {100*acc['ANN']} %")
print(f"Recall: {100*rec['ANN']} %")
print(f"Precisão: {100*prec['ANN']} %")
print(f"F1: {100*f1['ANN']} %")


print(f"\n\nMatriz de confusão:")
plt.figure(figsize=(5,4))
sns.set(font_scale = 1)
```

```
sns.set_style("white")
_ = sns.heatmap(cm, cmap = 'gist_yarg_r',annot = True, fmt='d')
```

```
4/4 [==============================] - 0s 1ms/step
Acurácia: 97.36842105263158 %
Recall: 95.74468085106383 %
Precisão: 97.82608695652173 %
F1: 96.7741935483871 %
```

Matriz de confusão:



Métricas

Relembrando:

- Acertos
  - "True Positive" (TP): Previsão --> Maligno ; Real --> Maligno
  - "True Negative" (TN): Previsão --> Benigno ; Real --> Benigno

- Erros
  - "False Positive" (FP): Previsão --> Maligno ; Real --> Benigno
  - "False Negative" (FN): Previsão --> Benigno ; Real --> Maligno

- Acurácia = $\frac{T_P+T_N}{T_P+T_N+F_P+F_N}$
- Recall = $\frac{T_P}{T_P+F_N}$
- Precisão = $\frac{T_P}{T_P+F_P}$
- F1 - Score = $\frac{2P_{rec}R_{ec}}{P_{rec}+R_{ec}}$

In [ ]:
```
metrics = pd.DataFrame({
    'Modelo': acc.keys(),
    'Acurácia':  np.array(list(acc.values()))*100,
    'Recall': np.array(list(rec.values()))*100,
    'Precisão': np.array(list(prec.values()))*100,
    'F1': np.array(list(f1.values()))*100
}).sort_values(by = ['Recall'], ascending = False)
metrics.reset_index(drop = True, inplace = True)
metrics
```

Out[ ]:

	Modelo	Acurácia	Recall	Precisão	F1
0	SVM_linear	98.245614	97.872340	97.872340	97.872340
1	logistic	96.491228	95.744681	95.744681	95.744681
2	SVM_kernal	98.245614	95.744681	100.000000	97.826087
3	ANN	97.368421	95.744681	97.826087	96.774194
4	tree	91.228070	93.617021	86.274510	89.795918
5	random_forest	94.736842	93.617021	93.617021	93.617021
6	reg_linear	95.614035	89.361702	100.000000	94.382022
7	k_neighbors	95.614035	89.361702	100.000000	94.382022
8	naive_bayes	89.473684	87.234043	87.234043	87.234043