

# Projeto 1 - Base de Dados Covid

O objetivo do projeto será desenvolver um estudo no dataset `COVID.csv`, base esta que contém informações sobre casos de COVID. Ou seja a partir do diagnóstico de sintomas e informações dos pacientes deve-se desenvolver um modelo para prever casos confirmados de COVID.

A descrição das variáveis contidas no dataset pode ser encontradas a seguir:

- **id**: Identificação do paciente
- **sex**: Sexo do Paciente (0 - Homem / 1 - Mulher)
- **patient\_type**: Se o paciente foi dispensado para casa (1) ou se foi internado (0)
- **intubed**: Se o paciente foi intubado ou não
- **pneumonia**: Se o paciente apresentou pneumonia ou não
- **age**: Idade do Paciente
- **pregnancy**: Se a paciente estava grávida ou não (para pacientes mulheres)
- **diabetes**: Se o paciente tem diabetes ou não
- **copd**: Se o paciente tem COPD ou não
- **asthma**: Se o paciente tem Asma ou não
- **inmsupr**: Se o paciente apresentou Imunossupressão ou não
- **hypertension**: Se o paciente tem hipertensão ou não
- **ohter\_disease**: Se o paciente tem outras doenças ou não
- **cardiovascular**: Se o paciente tem doenças cardíacas ou não
- **obesity**: Se o paciente tem obesidade ou não
- **renal\_chronic**: Se o paciente tem problemas renais ou não
- **tobacco**: Se o paciente é fumante ou não
- **contact\_other\_covid**: Se o paciente teve contato com outras pessoas diagnosticadas com covid
- **icu**: Se o paciente precisou ser internado na UTI
- **covid\_res**: Se o resultado do teste foi Positivo ou Negativo

Para ajudar no desenvolvimento do projeto, vamos separar o projeto em algumas seções, conforme descritas a seguir:

- **Preparação dos Dados e Verificação de Consistência**: Neste tópico deve ser feita a verificação da consistência dos dados e caso necessário efetuar eventuais modificações na base de dados. Alguns dos procedimentos que podemos fazer aqui são: Remoção e/ou tratamento de valores faltantes, remoção de duplicatas, ajustes dos tipos de variáveis, análise de *outliers* entre outras;

- **Análise Exploratória dos Dados:** Para fazermos a modelagem, precisamos conhecer muito bem os dados que estamos trabalhando. Por isso, nesta parte do projeto vocês desenvolveram análises e gráficos a respeito dos dados que estão utilizando. Tente tirar ao máximo informações sobre as variáveis em si e suas relações com as demais;
- **Modelagem dos Dados:** Nesta parte, vamos modelar um classificador para os resultados dos exames de COVID (`covid_res`). Vocês deveram **treinar pelo menos 3 modelos** (podendo testar mais que 3 também) e de acordo com alguma métrica de avaliação (escolhida por vocês), decidir qual será o melhor modelo a ser utilizado!;
- **Otimização do Modelo:** A partir do modelo escolhido no tópico anterior, vamos tentar aprimorar e garantir um melhor desempenho no modelo, seja fazendo validação cruzada, otimização de parâmetros com `GridSearchCV` ou `RandomizedSearchCV` e até mesmo testar diferentes *thresholds* (ao invés de utilizar a função `predict` do modelo, utilize a função `_predict_proba` do modelo e a partir das probabilidades determinar qual vai ser o limiar onde será considerado um caso positivo ou negativo);
- **Conclusões sobre o Projeto:** Para finalizar, descreva as suas conclusões sobre o desenvolvimento do modelo e os resultados obtidos.'

## Critérios de Avaliação

Um processo de análise e modelagem de dados depende de diversos fatores, desde quais sementes aleatórias foram definidas até mesmo o desempenho computacional da máquina utilizada. Dessa forma, **não** será cobrado que resultados os alunos obteram, mas sim o desenvolvimento do projeto, olhando os seguintes tópicos:

- Desenvolvimento mínimo de cada um dos itens acima;
- Padronização de Gráficos, Visualizações e códigos;
- *Clean Code* e códigos comentados;
- Explicação de todos os processos feitos e decisões tomadas

## Informações Gerais sobre o Projeto

- O projeto poderá ser desenvolvido **individualmente** ou em **grupos com até 4 pessoas**, caso façam em grupos enviar a relação de alunos do grupo para o professor;
- Data de Divulgação do Projeto: 29/11/2021;
- Monitoria do Projeto 1: 10/12/2021;
- Data de Entrega: 17/12/2021;

- Entrega: Através do *Class*, será criado um Projeto para a entrega dentro do módulo "Machine Learning", Aula "Árvore de Decisão" na aba Exercícios;

**Padrão de Entrega** Para a identificação dos alunos que entregaram o projeto, peço para que sigam o padrão de nome para o arquivo enviado conforme as orientações abaixo:

- Para quem realizar **individualmente**: NúmeroDaTurma\_Projeto1\_NomeDoALuno.ipynb;
- Para quem realizar em **grupos**: NúmeroDaTurma\_Projeto1 GrupoX.ipynb;

**Bem Importante:** Deixem devidamente identificado o número da turma, número do grupo e o nome do aluno ou alunos dentro do Notebook também.

## Preparação dos Dados e Verificação de Consistência

In [1]: `import pandas as pd`

In [2]: `# Carregamento do dataset  
df = pd.read_csv('COVID.csv')  
display(df.head(5))`

	Unnamed: 0																			
	sex	patient_type	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	co			
0	0	0	1	NaN	0.0	27	NaN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1	0	1	NaN	0.0	24	NaN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2	1	0	0.0	0.0	54	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0
3	3	0	0	0.0	1.0	30	NaN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	4	1	0	0.0	0.0	60	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.0

### Comentários:

Análise de tipos de dados e visualização inicial de faltantes

In [3]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 499692 entries, 0 to 499691
Data columns (total 20 columns):
 #   Column            Non-Null Count  Dtype  
--- 
 0   Unnamed: 0         499692 non-null   int64  
 1   sex               499692 non-null   int64  
 2   patient_type      499692 non-null   int64  
 3   intubed           107424 non-null   float64 
 4   pneumonia          499681 non-null   float64 
 5   age               499692 non-null   int64  
 6   pregnancy          245258 non-null   float64 
 7   diabetes           498051 non-null   float64 
 8   copd               498246 non-null   float64 
 9   asthma              498250 non-null   float64 
 10  inmsupr            498030 non-null   float64 
 11  hypertension        498203 non-null   float64 
 12  other_disease      497499 non-null   float64 
 13  cardiovascular     498183 non-null   float64 
 14  obesity             498222 non-null   float64 
 15  renal_chronic       498216 non-null   float64 
 16  tobacco             498113 non-null   float64 
 17  contact_other_covid 346017 non-null   float64 
 18  covid_res           499692 non-null   int64  
 19  icu                107423 non-null   float64 
dtypes: float64(15), int64(5)
memory usage: 76.2 MB
```

```
In [4]: # Análise de duplicidades com todas as colunas
q1 = len(df)
q2 = len(df.drop_duplicates())
print('Total de registros:', q1)
print('Descartando registros duplicados:', q2)
if q1 == q2:
    print('Nenhuma duplicidade.')
```

```
Total de registros: 499692
Descartando registros duplicados: 499692
Nenhuma duplicidade.
```

```
In [5]: # Análise de duplicidades desconsiderando o ID do paciente
df_tmp = df.drop('Unnamed: 0', axis=1)
q2 = len(df_tmp.drop_duplicates())
print('Total de registros:', q1)
print('Descartando registros duplicados:', q2)
if q1 == q2:
    print('Nenhuma duplicidade.')
df_tmp = None
```

```
Total de registros: 499692
Descartando registros duplicados: 74428
```

## Comentários:

A primeira coluna, denominada 'Unnamed: 0' é a coluna ID de paciente. Ela não será utilizada nos modelos.

Considerando ela, não temos duplicidades, porém, sem ela temos um número expressivo do modelo.

O que farei então, é buscar identificar o modelo ideal com as repetições, e ao final, repetir o processo de treino, validação e teste sem as repetições, para buscar avaliar que impacto há na remoção.

Uma variável de referência para a variável-alvo é definida para um código mais claro e fácil de manter.

```
In [6]: df['Unnamed: 0'].nunique()
```

```
Out[6]: 499692
```

```
In [7]: df = df.drop('Unnamed: 0', axis=1)  
df.head(3)
```

```
Out[7]:
```

	sex	patient_type	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_
0	0		1	NaN	0.0	27	NaN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0		1	NaN	0.0	24	NaN	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	1		0	0.0	0.0	54	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0

```
In [8]: # Feature alvo do projeto  
target = 'covid_res'
```

## Comentários:

Análise dos registros com dados faltantes

```
In [9]: len(df) - df[df.notna()].count()
```

```
Out[9]:
```

sex	0
patient_type	0
intubed	392268
pneumonia	11
age	0
pregnancy	254434
diabetes	1641
copd	1446
asthma	1442
inmsupr	1662
hypertension	1489
other_disease	2193
cardiovascular	1509
obesity	1470
renal_chronic	1476
tobacco	1579
contact_other_covid	153675
covid_res	0
icu	392269

dtype: int64

## Removendo registros sem dados para pneumonia

```
In [10]: df = df.drop(df.loc[df['pneumonia'].isna()].index)
```

### Comentários:

Antes de eliminar faltantes, temos que considerar as variáveis **sex** e **pregnancy**: Não devemos ter "homens grávidos" ou mulheres sem o estado de gravidez especificado. A quantidade de faltantes pode estar relacionada com o gênero das pessoas, logo é necessário tratar isso especificamente.

## Tratando *pregnancy*

```
In [11]: # Vamos criar um Dataset contendo apenas as colunas de interesse para agilizar
df_sex = df[['sex', 'pregnancy']].copy()
df_sex.head(3)
```

```
Out[11]:
```

	sex	pregnancy
0	0	NaN
1	0	NaN
2	1	0.0

```
In [12]: # Agrupando por sexo e gravidez, e adicionando uma coluna para contabilizar por grupo. Os valores nulos não podem
# ser desprezados.
```

```
df_sex['qty'] = 1  
df_sex.groupby(by=['sex', 'pregnancy'], dropna=False).count()
```

Out[12]:

qty

sex	pregnancy	qty
0	NaN	253092
1	0.0	241639
	1.0	3614
	NaN	1336

## Comentários:

**Sex=0** indica gênero masculino. Para todos o valor de **pregnancy** é **NaN**.

**Sex=1** indica gênero feminino. É seguro concluir que 0 indica "não grávida" e 1 indica "grávida".

Temos 1.336 mulheres sem indicação de gravidez, o que representa 0,27% da base.

## Decisões:

a) Atribuir **2** como valor de **pregnancy** para os homens, distinguindo dos valores possíveis para mulheres.

b) Eliminar todas as mulheres para as quais não foi informado o estado de gravidez, dada a pequena quantidade de registros.

In [13]:

```
df.loc[df['sex'] == 0, 'pregnancy'] = 2  
df = df.drop(df.loc[df['pregnancy'].isna()].index)  
  
# Conferindo  
df_sex = df[['sex', 'pregnancy']].copy()  
df_sex['qty'] = 1  
df_sex.groupby(by=['sex', 'pregnancy'], dropna=False).count()
```

Out[13]:

qty

sex	pregnancy	qty
0	2.0	253092
1	0.0	241639
	1.0	3614

## Avaliação de *Intubed* e *ICU*

**Intubed** está sem valor para vários registros.

Devemos avaliar considerando apenas os pacientes que ficaram no hospital (**patient\_type = 0**).

```
In [14]: df_sex = None
```

```
In [15]: df_tmp = df[['patient_type', 'intubed']].copy()
df_tmp['qty'] = 1
df_tmp.groupby(by=['patient_type', 'intubed'], dropna=False).count()
```

Out[15]:

		qty
patient_type	intubed	
0	0.0	98210
	1.0	9078
	NaN	121
1	NaN	390936

## Comentários:

Como podemos notar, **intubed** não está preenchido para os casos em que o paciente voltou para casa, e temos 121 registros de pacientes hospitalizados onde essa informação não foi fornecida.

### Decisões:

- Preencher com 2 (não entubado porque foi para casa) para todos que retornaram para casa.
- Eliminar os registros de pacientes hospitalizados sem informação sobre o entubamento, pois representam 0,024% apenas.

```
In [16]: df.loc[df['patient_type'] == 1, 'intubed'] = 2
df = df.drop(df.loc[df['intubed'].isna()].index)
# Conferindo
df_tmp = df[['patient_type', 'intubed']].copy()
df_tmp['qty'] = 1
df_tmp.groupby(by=['patient_type', 'intubed'], dropna=False).count()
```

Out[16]:

		qty
patient_type	intubed	
0	0.0	98210
	1.0	9078
1	2.0	390936

```
In [17]: # Revendo a posição de dados faltantes:
len(df) - df[df.notna()].count()
```

```
Out[17]: sex          0  
patient_type      0  
intubed          0  
pneumonia        0  
age              0  
pregnancy        0  
diabetes         1541  
copd             1345  
asthma           1340  
inmsupr          1556  
hypertension     1388  
other_disease    2089  
cardiovascular   1407  
obesity          1371  
renal_chronic    1377  
tobacco          1480  
contact_other_covid 153397  
covid_res         0  
icu              390937  
dtype: int64
```

Avaliando **ICU**: </spam>

```
In [18]: df_tmp = df[['patient_type', 'icu']].copy()  
df_tmp['qty'] = 1  
df_tmp.groupby(by=['patient_type', 'icu'], dropna=False).count()
```

```
Out[18]:      qty  
patient_type  icu  
0            0.0  98293  
1            1.0  8994  
NaN          NaN    1  
1  NaN       390936
```

## Comentários:

Dentre os pacientes que ficaram no hospital (**patient\_type=0**) somente 1 não têm informação sobre UTI.

Para todos que foram para casa, essa informação não foi preenchida no dataset.

### Decisões:

- Eliminar o paciente internado para o qual não foi informado o status para UTI.
- Preencher com 2 para todos os pacientes que voltaram para casa (não entubado por foi para casa).

```
In [19]: df.loc[df['patient_type'] == 1, 'icu'] = 2  
df = df.drop(df.loc[df['icu'].isna()].index)
```

```
# Conferindo
df_tmp = df[['patient_type', 'icu']].copy()
df_tmp['qty'] = 1
df_tmp.groupby(by=['patient_type', 'icu'], dropna=False).count()
```

Out[19]:

**qty**

patient_type	icu	qty
0	0.0	98293
	1.0	8994
1	2.0	390936

In [20]: `len(df) - df_tmp['qty'].sum()`

Out[20]: 0

In [21]: # Revendo a posição de dados faltantes:

```
(len(df) - df[df.notna()].count()) / len(df) * 100      # Como percentuais
```

Out[21]:

sex	0.000000
patient_type	0.000000
intubed	0.000000
pneumonia	0.000000
age	0.000000
pregnancy	0.000000
diabetes	0.309299
copd	0.269959
asthma	0.268956
inmsupr	0.312109
hypertension	0.278590
other_disease	0.419089
cardiovascular	0.282404
obesity	0.275178
renal_chronic	0.276382
tobacco	0.297056
contact_other_covid	30.788623
covid_res	0.000000
icu	0.000000

`dtype: float64`

## Comentários:

Agora, a única variável com expressiva quantidade de dados não preenchidos é **contact\_other\_covid**.

Uma possibilidade é a falta de valor indicar pessoa sem contato. Ou pode realmente ser uma falta de informação.

Se obtivermos mais de um valor preenchido para a variável, a falta de valor é falta de informação.

Nesse cenário, vamos utilizar a variável **covid\_res** para mais análises.

## Avaliando contact\_other\_covid

```
In [22]: df['contact_other_covid'].value_counts()
```

```
Out[22]: 1.0    196257  
0.0    148570  
Name: contact_other_covid, dtype: int64
```

**Conclusão:** Temos valores 0 e 1. A falta de informação é um problema. Vamos consultar covid\_res

```
In [23]: df_tmp = df[[target, 'contact_other_covid']].copy()  
df_tmp['qty'] = 1  
df_tmp.groupby(by=[target, 'contact_other_covid'], dropna=False).sum()
```

```
Out[23]:      qty  
  
covid_res  contact_other_covid  
-----  
0           0.0    83419  
             1.0   122375  
             NaN    72545  
1           0.0    65151  
             1.0   73882  
             NaN    80851
```

## Comentários:

Observamos que praticamente metade dos casos está no grupo de pacientes com teste positivo para covid e metade no grupo com teste negativo.

Ainda assim, nada podemos afirmar sobre essas pessoas terem ou não tido contato com pessoas infectadas. A quantidade de registros é relevante, logo, a **decisão** é adotar uma estratégia para anular a ausência buscando não influenciar os modelos. Para tanto, vamos preencher esses registros com 2.

```
In [24]: df['contact_other_covid'].fillna(2, inplace=True)  
df['contact_other_covid'].value_counts()
```

```
Out[24]: 1.0    196257  
2.0    153396  
0.0    148570  
Name: contact_other_covid, dtype: int64
```

```
In [25]: df_tmp = None
```

```
In [26]: # Revendo a posição de dados faltantes:  
(len(df) - df[df.notna().count()]) / len(df) * 100
```

```
Out[26]: sex          0.000000  
patient_type  0.000000  
intubed       0.000000  
pneumonia     0.000000  
age           0.000000  
pregnancy    0.000000  
diabetes      0.309299  
copd          0.269959  
asthma        0.268956  
inmsupr       0.312109  
hypertension   0.278590  
other_disease 0.419089  
cardiovascular 0.282404  
obesity        0.275178  
renal_chronic  0.276382  
tobacco        0.297056  
contact_other_covid 0.000000  
covid_res      0.000000  
icu            0.000000  
dtype: float64
```

## Tratamento para os faltantes remanescentes

*Os registros remanescentes com dados faltantes representam muito pouco (< 0,5%), logo a **decisão** é eliminar todos.*

```
In [27]: df.dropna(how='any', inplace=True)  
df.info()
```

```

<class 'pandas.core.frame.DataFrame'>
Int64Index: 494948 entries, 0 to 499691
Data columns (total 19 columns):
 #   Column           Non-Null Count Dtype  
 --- 
 0   sex              494948 non-null  int64  
 1   patient_type     494948 non-null  int64  
 2   intubed          494948 non-null  float64 
 3   pneumonia         494948 non-null  float64 
 4   age               494948 non-null  int64  
 5   pregnancy         494948 non-null  float64 
 6   diabetes          494948 non-null  float64 
 7   copd              494948 non-null  float64 
 8   asthma             494948 non-null  float64 
 9   inmsupr           494948 non-null  float64 
 10  hypertension       494948 non-null  float64 
 11  other_disease     494948 non-null  float64 
 12  cardiovascular    494948 non-null  float64 
 13  obesity            494948 non-null  float64 
 14  renal_chronic      494948 non-null  float64 
 15  tobacco             494948 non-null  float64 
 16  contact_other_covid 494948 non-null  float64 
 17  covid_res          494948 non-null  int64  
 18  icu                494948 non-null  float64 

dtypes: float64(15), int64(4)
memory usage: 75.5 MB

```

## Avaliação dos tipos de dados das *features*

```
In [28]: display(df.sample(10))
```

**Conclusão:** Ideal converter todas as variáveis para **int**.

In [29]:

```
for cname in df.columns:  
    df[cname] = pd.to_numeric(df[cname], downcast='integer')  
display(df.info())  
df.head(3)
```

```
<class 'pandas.core.frame.DataFrame'>  
Int64Index: 494948 entries, 0 to 499691  
Data columns (total 19 columns):  
 #   Column           Non-Null Count  Dtype     
---  --  
 0   sex              494948 non-null   int8     
 1   patient_type     494948 non-null   int8     
 2   intubed          494948 non-null   int8     
 3   pneumonia         494948 non-null   int8     
 4   age               494948 non-null   int8     
 5   pregnancy         494948 non-null   int8     
 6   diabetes          494948 non-null   int8     
 7   copd              494948 non-null   int8     
 8   asthma             494948 non-null   int8     
 9   inmsupr           494948 non-null   int8     
 10  hypertension       494948 non-null   int8     
 11  other_disease     494948 non-null   int8     
 12  cardiovascular    494948 non-null   int8     
 13  obesity            494948 non-null   int8     
 14  renal_chronic     494948 non-null   int8     
 15  tobacco            494948 non-null   int8     
 16  contact_other_covid 494948 non-null   int8     
 17  covid_res          494948 non-null   int8     
 18  icu                494948 non-null   int8     
dtypes: int8(19)  
memory usage: 12.7 MB  
None
```

Out[29]:

	sex	patient_type	intubed	pneumonia	age	pregnancy	diabetes	copd	asthma	inmsupr	hypertension	other_disease	cardiovascular	obesity	renal_chronic	tobacco	contact_other_covid
0	0	1	2	0	27	2	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	0	24	2	0	0	0	0	0	0	0	0	0	0	0
2	1	0	0	0	54	0	0	0	0	0	0	0	0	1	0	0	0

Comentário:

O dataset diminuiu de 75.5MB para 12.7MB.

# Análise Exploratória dos Dados

## Características de posição e dispersão das variáveis

In [30]: `df.describe().T`

Out[30]:

	count	mean	std	min	25%	50%	75%	max
<b>sex</b>	494948.0	0.492807	0.499949	0.0	0.0	0.0	1.0	1.0
<b>patient_type</b>	494948.0	0.786178	0.410003	0.0	1.0	1.0	1.0	1.0
<b>intubed</b>	494948.0	1.590060	0.796320	0.0	2.0	2.0	2.0	2.0
<b>pneumonia</b>	494948.0	0.156283	0.363124	0.0	0.0	0.0	0.0	1.0
<b>age</b>	494948.0	42.512997	16.623871	0.0	31.0	41.0	53.0	120.0
<b>pregnancy</b>	494948.0	1.021657	0.996123	0.0	0.0	2.0	2.0	2.0
<b>diabetes</b>	494948.0	0.124722	0.330404	0.0	0.0	0.0	0.0	1.0
<b>copd</b>	494948.0	0.016545	0.127560	0.0	0.0	0.0	0.0	1.0
<b>asthma</b>	494948.0	0.032529	0.177400	0.0	0.0	0.0	0.0	1.0
<b>inmsupr</b>	494948.0	0.016123	0.125948	0.0	0.0	0.0	0.0	1.0
<b>hypertension</b>	494948.0	0.162864	0.369242	0.0	0.0	0.0	0.0	1.0
<b>other_disease</b>	494948.0	0.030848	0.172905	0.0	0.0	0.0	0.0	1.0
<b>cardiovascular</b>	494948.0	0.022792	0.149241	0.0	0.0	0.0	0.0	1.0
<b>obesity</b>	494948.0	0.163991	0.370268	0.0	0.0	0.0	0.0	1.0
<b>renal_chronic</b>	494948.0	0.020020	0.140070	0.0	0.0	0.0	0.0	1.0
<b>tobacco</b>	494948.0	0.086163	0.280604	0.0	0.0	0.0	0.0	1.0
<b>contact_other_covid</b>	494948.0	1.009773	0.778286	0.0	0.0	1.0	2.0	2.0
<b>covid_res</b>	494948.0	0.440836	0.496488	0.0	0.0	0.0	1.0	1.0
<b>icu</b>	494948.0	1.590044	0.796342	0.0	2.0	2.0	2.0	2.0

In [31]:

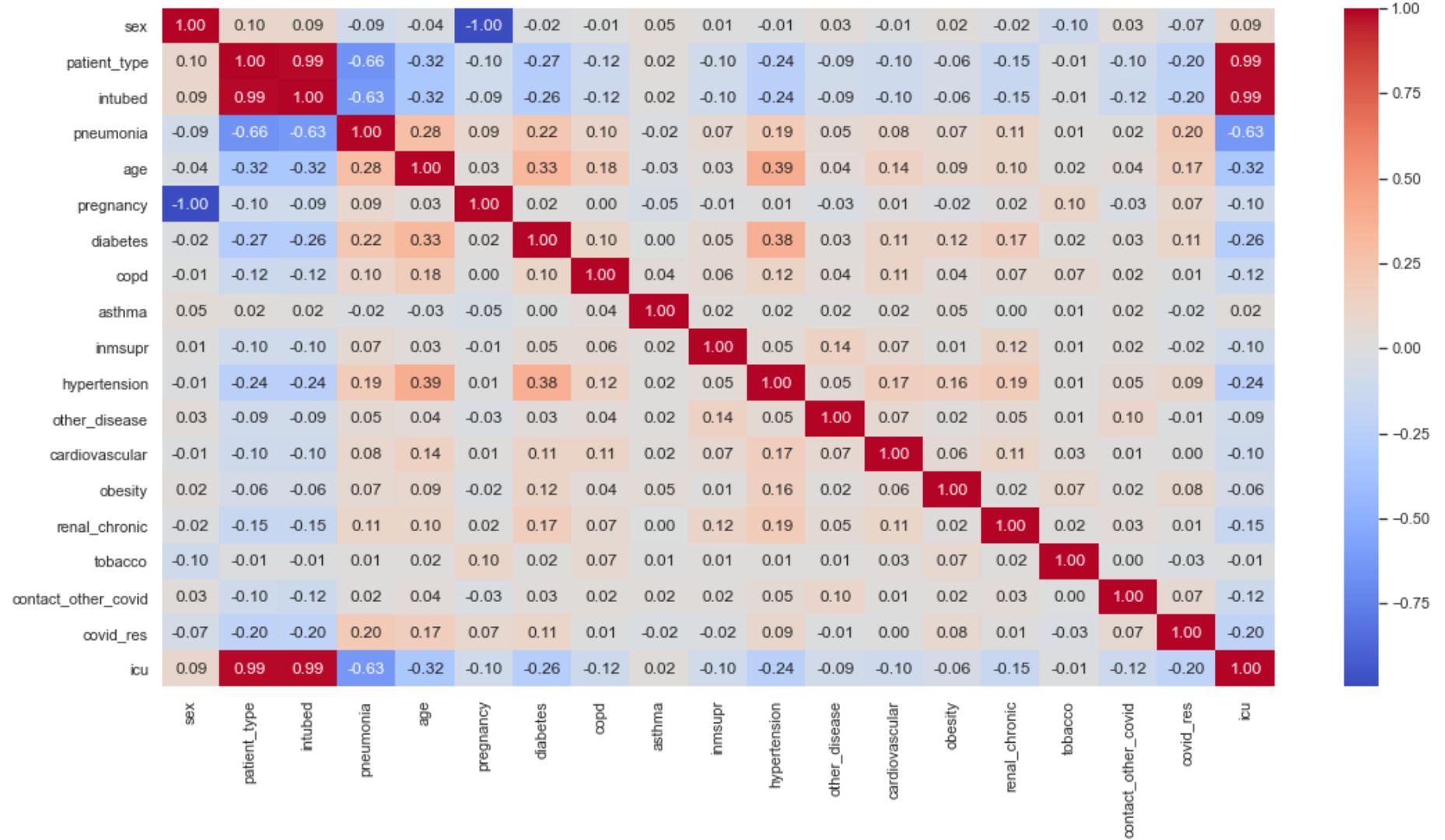
```
total      = len(df)
positivos = len(df.loc[df[target]==1])
negativos = len(df.loc[df[target]==0])
perc_pos  = positivos / total * 100
perc_neg  = negativos / total * 100
print(f'Positivos: {positivos} ({perc_pos:.2f}%), Negativos: {negativos} ({perc_neg:.2f}%)')
```

Positivos: 218191 (44.08%), Negativos: 276757 (55.92%)

```
In [32]: import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
sns.set_theme(style='whitegrid')
```

## Análise de correlação das variáveis

```
In [33]: plt.figure(figsize=(18,9))
sns.heatmap(df.corr(), cmap='coolwarm', annot=True, fmt='.2f')
plt.show()
```



## Comentários:

Algumas correlações óbvias estão confirmadas. "Gravidez" é totalmente relacionada com o gênero, e "UTI" é totalmente relacionada com "Tipo de paciente" (afinal, quem foi para casa não pode ter ido para a UTI).

É coerente também, a alta correlação entre UTI (**icu**) e entubação (**intubed**).

Nota-se também uma alta correlação entre **pneumonia** e **patient\_type**, ou seja, pacientes diagnosticados com pneumonia são internados, de forma geral. Constatamos que isoladamente, nenhuma das variáveis explicativas indica se um paciente tem ou não covid.

## Análise univariada

**ALERTA:** Execução leva vários minutos</spam>

In [34]:

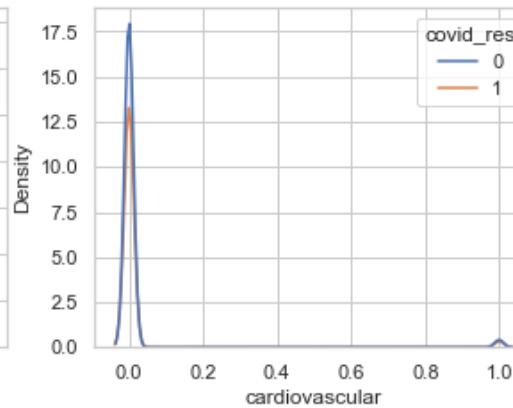
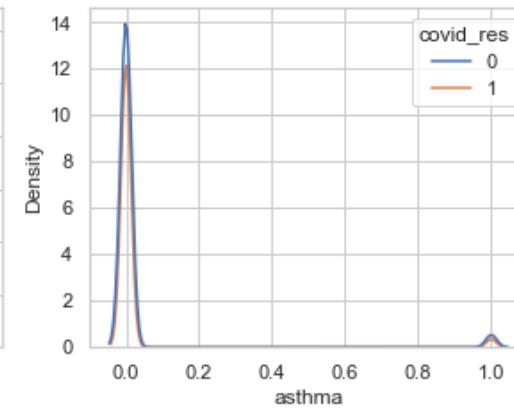
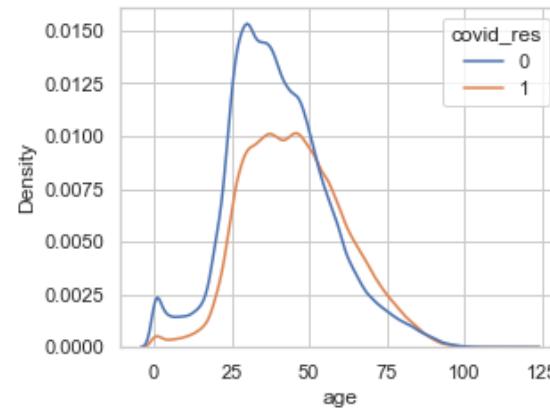
```
import warnings  
warnings.filterwarnings("ignore")
```

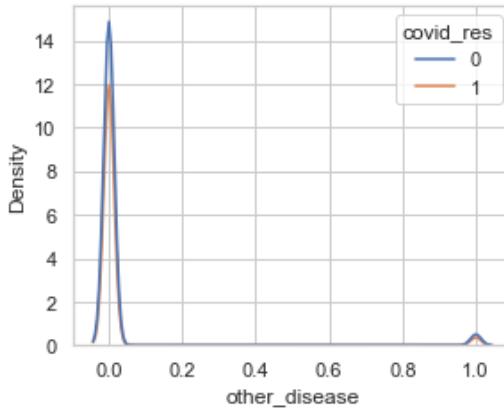
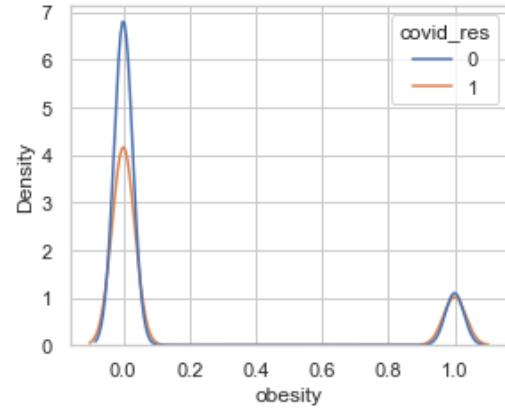
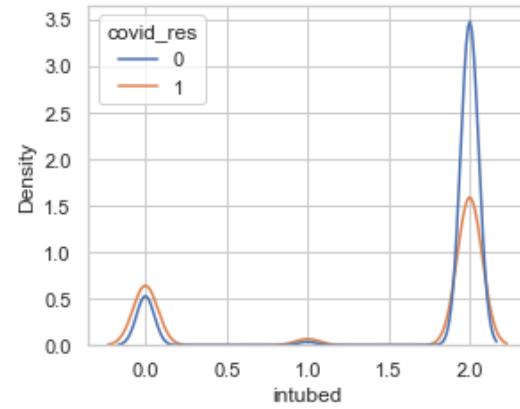
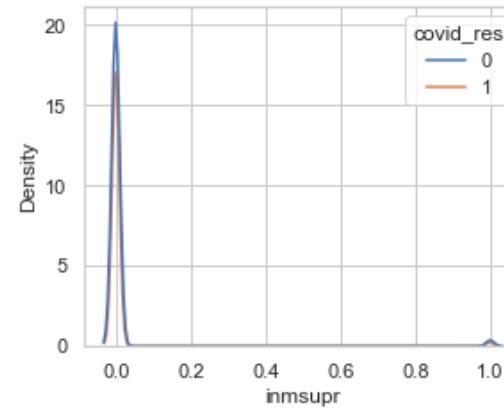
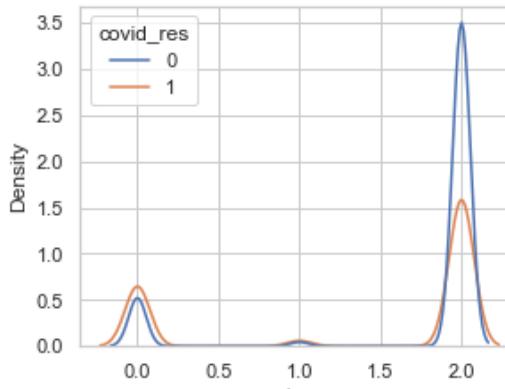
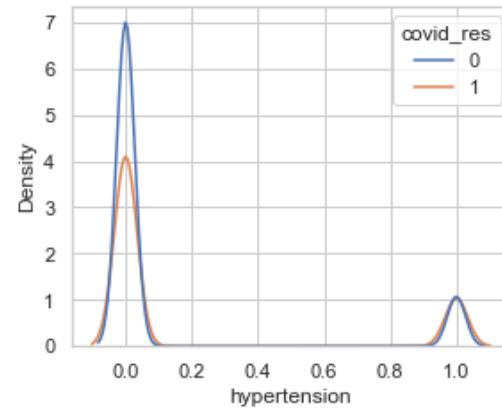
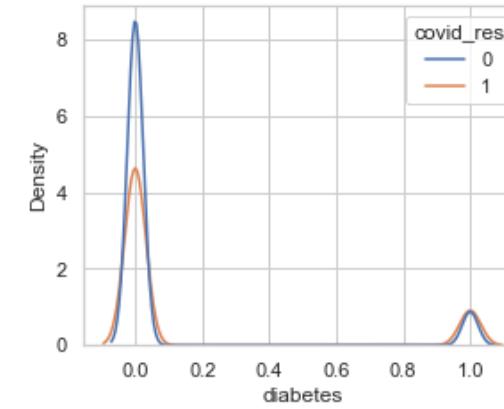
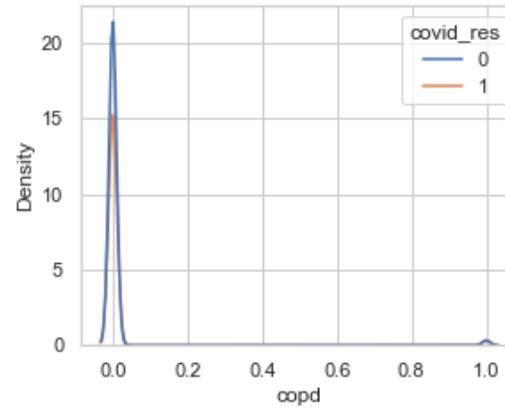
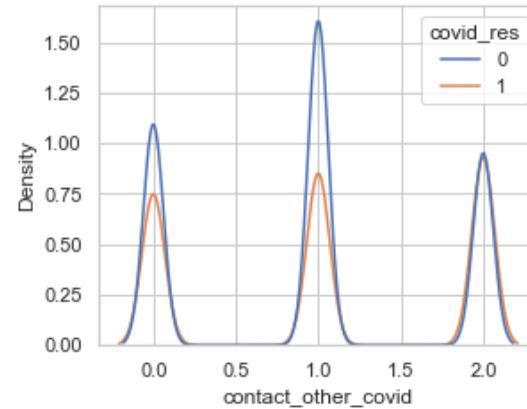
In [35]:

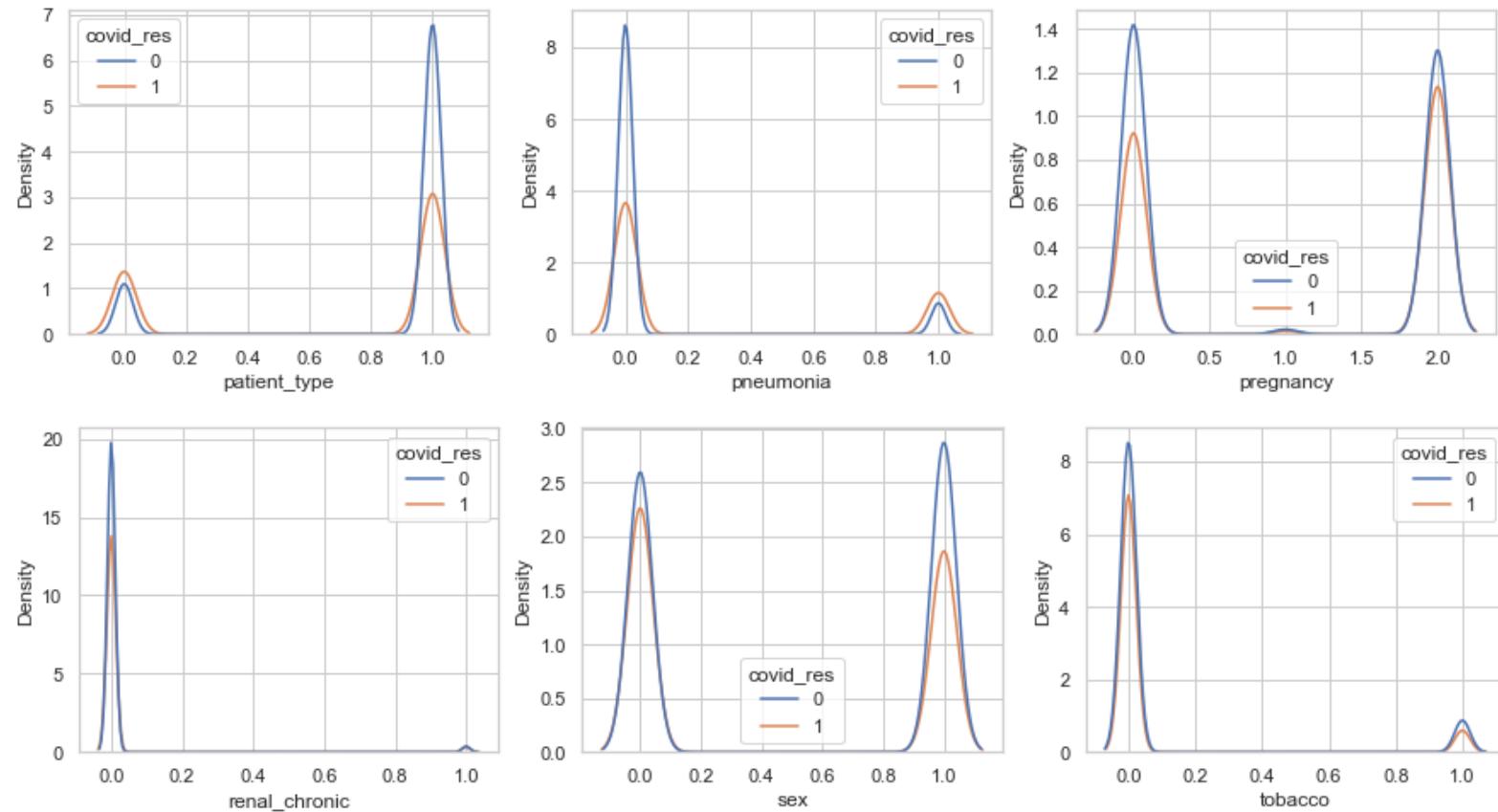
```
# Lista das variáveis explicativas (classificando alfabeticamente para facilitar a análise dos gráficos)  
vars = df.columns.tolist()  
vars.remove(target)  
vars.sort()
```

In [36]:

```
# Laço para plotar os gráficos em 3 colunas  
lin, col = 1, 1  
for var in vars:  
    if col == 1:  
        # No primeiro gráfico da linha prepara o eixo e dimensões  
        fig, axes = plt.subplots(1,3, figsize=(15,3.5))  
  
        # Dados da área de plotagem a utilizar  
        ax = axes[col-1]  
  
        # Plotagem do gráfico  
        sns.kdeplot(data=df, x=var, hue='covid_res', ax=ax)  
  
        # Se todas as colunas foram preenchidas, apresenta os 3 gráficos e inicia uma nova linha  
        col += 1  
    if col == 4:  
        plt.show()  
        col = 1
```







## Comentários:

Pelos gráficos podemos observar alguns fatos:

- Os pacientes da amostragem concentram-se nas idades de 25 a 50 anos.
- O contato com pessoa infectada é relevante para uma pessoa ser diagnosticada com covid.
- Nessa amostragem, infecções são ligeiramente mais frequentes em homens do que em mulheres.
- UTI e entubação parecem ser bons indicadores de contaminação com covid.
- Pacientes diagnosticados com covid e outro quadro clínico são bem poucos na amostragem. Isso é esperado pois a população sadiá é bem maior do que aquela com algum quadro específico, tais como hipertensão, doença renal, etc. Mesmo raciocílio se aplica para fumantes e não-fumantes (tobacco). </span>

## Análise bivariada

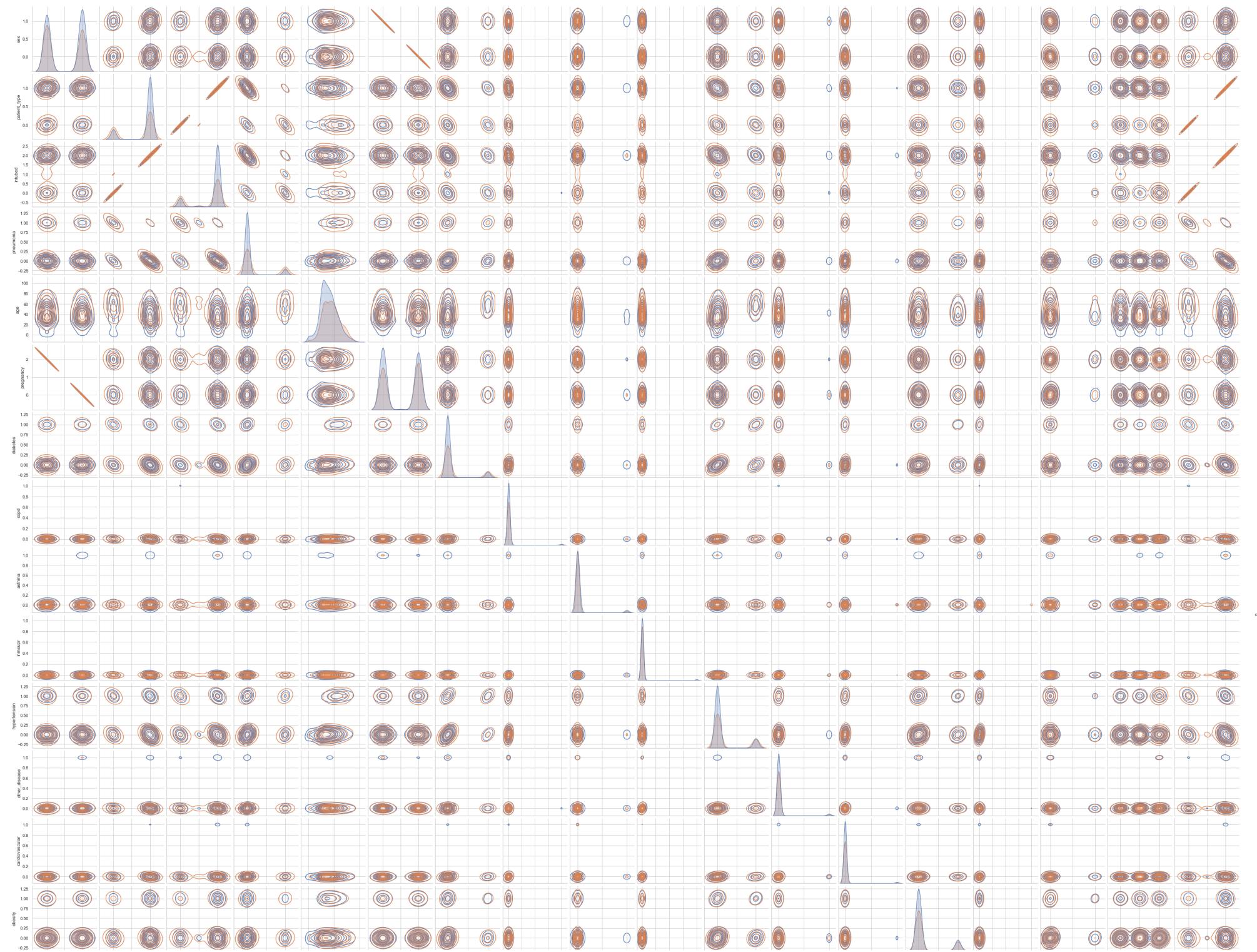
Uma pequena fração dos dados será utilizada para o tempo de plotagem não tornar-se impeditivo.

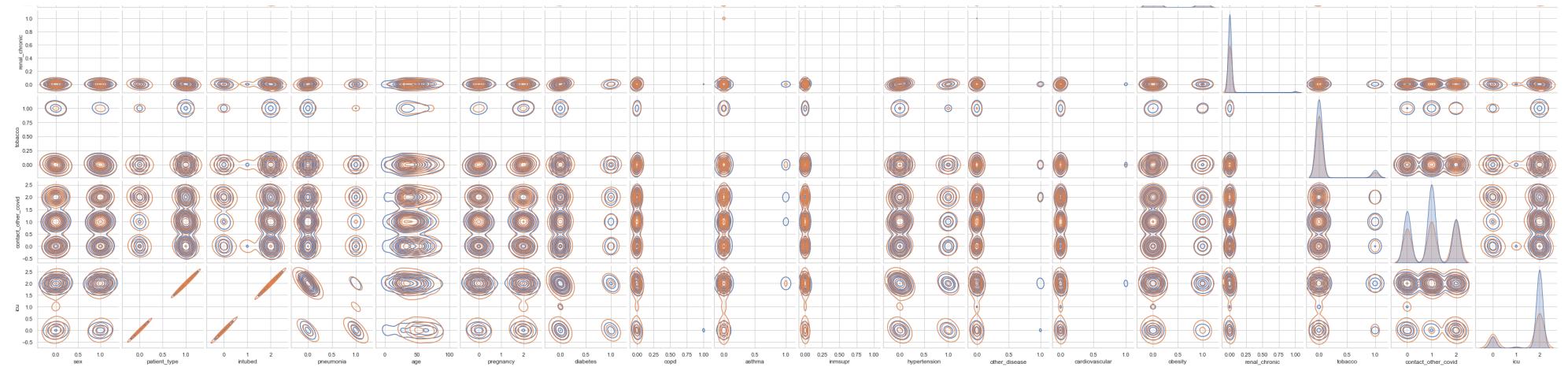
**ALERTA:** Execução leva vários minutos</spam>

```
In [37]: amostra = df.sample(frac=0.01, random_state=12)
```

```
In [38]: sns.pairplot(data=amostra, kind='kde', hue='covid_res')
```

```
Out[38]: <seaborn.axisgrid.PairGrid at 0x1c027f1b310>
```





## Comentários:

*Não temos nenhuma feature dominante (que diretamente nos permita deduzir a variável-alvo).  
Por todos os gráficos nota-se uma maior presença de negativos do que positivos na amostragem.  
Deve-se ter um cuidado relacionado a esse aspecto no cálculo de probabilidades dos modelos.*

## Modelagem dos Dados #1 - Regressão Logística

```
In [39]: from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm
```

O primeiro passo é separar as bases de treino, teste e validação

```
In [40]: from sklearn.model_selection import train_test_split

X = df.drop(target, axis=1)
y = df[target]

PERC_TST = 0.15
PERC_VAL = 0.10

XX, X_tst, yy, y_tst = train_test_split(X, y, test_size=PERC_TST, random_state=12)

X_trn, X_val, y_trn, y_val = train_test_split(XX, yy, test_size=PERC_VAL, random_state=12)

XX = None
yy = None

print('Shapes:')
```

```
print(f'treino      = X:{X_trn.shape}, y:{y_trn.shape}')
print(f'validação = X:{X_val.shape}, y:{y_val.shape}')
print(f'teste       = X:{X_tst.shape}, y:{y_tst.shape}')
```

Shapes:

```
treino      = X:(378634, 18), y:(378634,)
validação = X:(42071, 18), y:(42071,)
teste       = X:(74243, 18), y:(74243,)
```

**Uma função para facilitar a avaliação dos modelos**

```
In [41]: from sklearn.metrics import accuracy_score, precision_score, recall_score

def scores(tit, y, y_pred):
    acc = accuracy_score(y, y_pred) * 100
    prec = precision_score(y, y_pred) * 100
    rec = recall_score(y, y_pred) * 100

    return f'{tit}:\nAcc: {acc:.2f}%, Precision: {prec:.2f}%, Recall: {rec:.2f}%'
```

**Uma função para facilitar a plotagem da matriz de confusão dos modelos**

```
In [42]: from mlxtend.plotting import plot_confusion_matrix
from sklearn.metrics import confusion_matrix
from matplotlib import pyplot as plt

def cm(y, y_pred):
    plt.figure(figsize=(3.5, 3.5))
    g = confusion_matrix(y, y_pred)
    plot_confusion_matrix(conf_mat=g, show_normed=True, cmap='Greens')
    plt.ylabel('real')
    plt.xlabel('predito')
    plt.show()
```

**Uma função para facilitar as previsões, apresentação de scores e matrizes de confusão**

```
In [43]: def pred_fats_cm(modelo, X_train=None, y_train=None, X_valid=None, y_valid=None, X_test=None, y_test=None, gra_cm=True):
    # Sem previsões ainda
    y_trn_mod = None
    y_val_mod = None
    y_tst_mod = None

    # X_train não é nulo, então calcula y_trn_mod (predict do modelo para X_trn)
    if X_train is not None:
        y_trn_mod = modelo.predict(X_train)
        print(scores('Treino', y_train, y_trn_mod))

    # X_valid não é nulo, então calcula y_val_mod (predict do modelo para X_val)
    if X_valid is not None:
        y_val_mod = modelo.predict(X_valid)
```

```

print( scores('Validação', y_valid, y_val_mod) )

# Se não há X_test, apresenta a matriz de confusão de validação
if X_test is None and gra_cm:
    cm(y_val, y_val_mod)

# X_test não é nulo, então calcula y_tst_mod (predict do modelo para X_tst)
if X_test is not None and gra_cm:
    y_tst_mod = modelo.predict(X_test)
    print( scores('Teste' , y_test, y_tst_mod) )
    cm(y_tst, y_tst_mod)

return y_trn_mod, y_val_mod, y_tst_mod

```

### Modelo de regressão logística</spam>

In [44]: `lr = LogisticRegression(n_jobs=-1)`

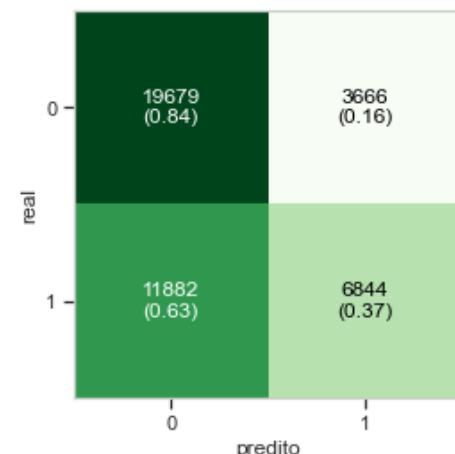
**ALERTA:** Execução leva alguns minutos

In [45]: `lr.fit(X_trn, y_trn)`

Out[45]: `LogisticRegression(n_jobs=-1)`

In [46]: `y_trn_pred_lr, y_val_pred_lr, _ = pred_fats_cm(lr, X_trn, y_trn, X_val, y_val)`

Treino:  
Acc: 63.39%, Precision: 64.83%, Recall: 36.86%  
Validação:  
Acc: 63.04%, Precision: 65.12%, Recall: 36.55%  
<Figure size 252x252 with 0 Axes>



# Modelagem dos Dados #2 - Árvore de Decisão

```
In [47]: from sklearn.tree import DecisionTreeClassifier  
from sklearn.tree import plot_tree
```

```
In [48]: dt = DecisionTreeClassifier()
```

**ALERTA: Execução leva alguns minutos**

```
In [49]: dt.fit(X_trn, y_trn)
```

```
Out[49]: DecisionTreeClassifier()
```

```
In [50]: y_trn_pred_dt, y_val_pred_dt, _ = pred_fats_cm(dt, X_trn, y_trn, X_val, y_val)
```

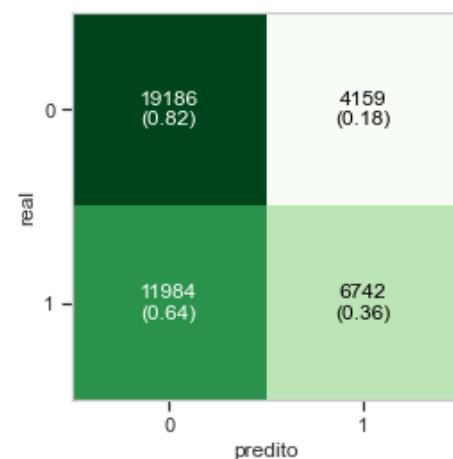
Treino:

Acc: 68.63%, Precision: 74.44%, Recall: 43.81%

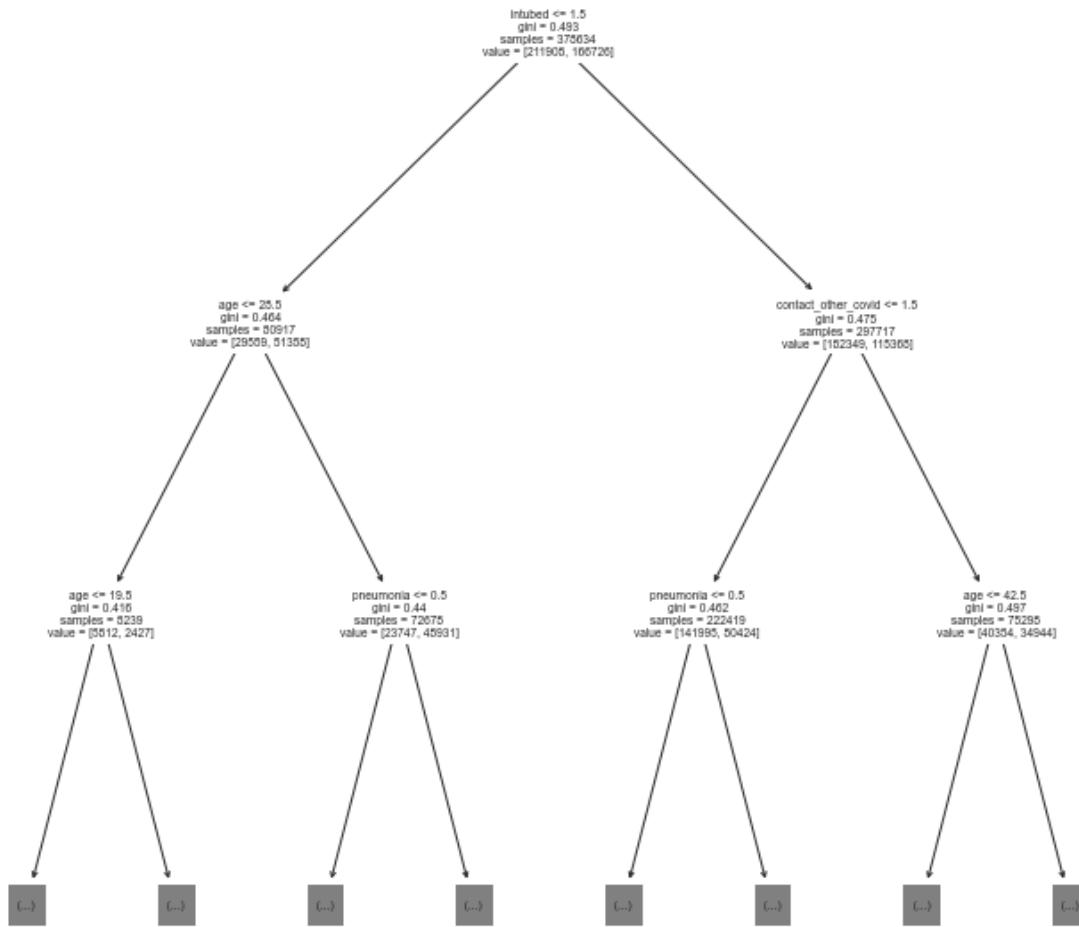
Validação:

Acc: 61.63%, Precision: 61.85%, Recall: 36.00%

<Figure size 252x252 with 0 Axes>



```
In [51]: plt.figure(figsize=(12,12))  
plot_tree(dt, feature_names=X_trn.columns, max_depth=2)  
plt.show()
```



## Modelagem dos Dados #3 - RandomForest

In [52]: `from sklearn.ensemble import RandomForestClassifier`

In [53]: `rf = RandomForestClassifier(random_state=12)`

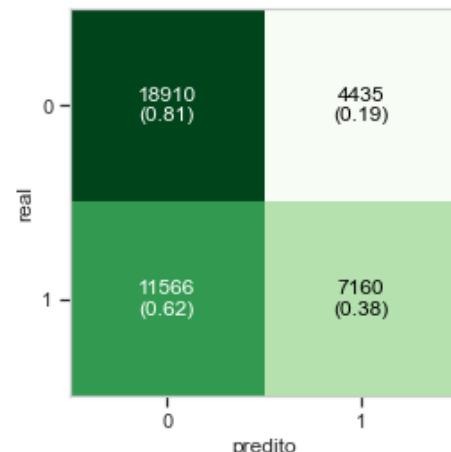
**ALERTA:** Execução leva vários minutos</spam>

```
In [54]: rf.fit(X_trn, y_trn)
```

```
Out[54]: RandomForestClassifier(random_state=12)
```

```
In [55]: y_trn_pred_rf, y_val_pred_rf, _ = pred_fats_cm(rf, X_trn, y_trn, X_val, y_val)
```

Treino:  
Acc: 68.63%, Precision: 72.98%, Recall: 45.66%  
Validação:  
Acc: 61.97%, Precision: 61.75%, Recall: 38.24%  
<Figure size 252x252 with 0 Axes>



## Modelagem dos Dados #4 - AdaBoost

```
In [56]: from sklearn.ensemble import AdaBoostClassifier
```

```
In [57]: ada = AdaBoostClassifier(n_estimators=20, random_state=12)
```

**ALERTA: Execução leva alguns minutos</spam>**

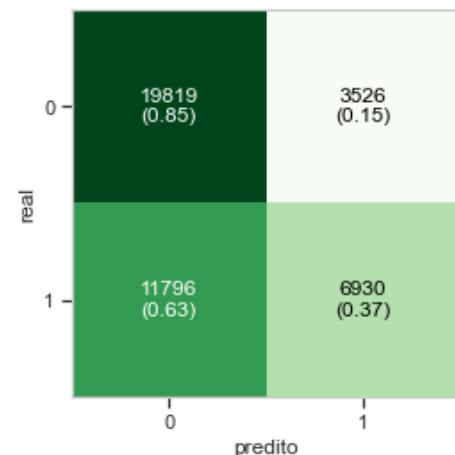
```
In [58]: ada.fit(X_trn, y_trn)
```

```
Out[58]: AdaBoostClassifier(n_estimators=20, random_state=12)
```

```
In [59]: y_trn_pred_ada, y_val_pred_ada, _ = pred_fats_cm(ada, X_trn, y_trn, X_val, y_val)
```

Treino:  
Acc: 63.73%, Precision: 65.54%, Recall: 37.18%  
Validação:  
Acc: 63.58%, Precision: 66.28%, Recall: 37.01%

<Figure size 252x252 with 0 Axes>



## Modelagem dos Dados #5 - LightGBM

In [60]: `from Lightgbm import LGBMClassifier`

In [61]: `lgbm = LGBMClassifier()`

**ALERTA: Execução leva alguns minutos**

In [62]: `lgbm.fit(X_trn, y_trn)`

Out[62]: `LGBMClassifier()`

In [63]: `y_trn_pred_lg, y_val_pred_lg, _ = pred_fats_cm(lgbm, X_trn, y_trn, X_val, y_val)`

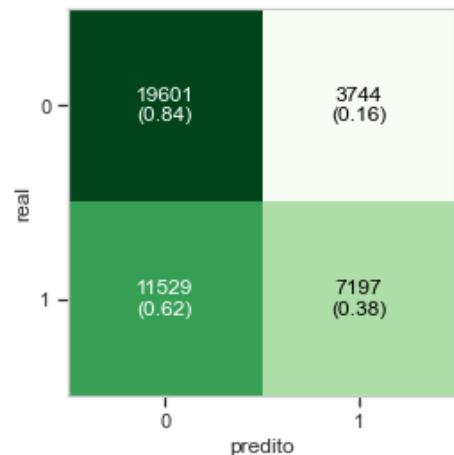
Treino:

Acc: 64.45%, Precision: 66.37%, Recall: 39.05%

Validação:

Acc: 63.70%, Precision: 65.78%, Recall: 38.43%

<Figure size 252x252 with 0 Axes>



## Comparação dos resultados dos modelos não otimizados

```
In [64]: print( scores('Regressão Logística - Treino' , y_trn, y_trn_pred_lr) )
print( scores('Regressão Logística - Validação', y_val, y_val_pred_lr) )
print()
print( scores('Árvore de Decisão - Treino' , y_trn, y_trn_pred_dt) )
print( scores('Árvore de Decisão - Validação', y_val, y_val_pred_dt) )
print()
print( scores('RandomForest - Treino' , y_trn, y_trn_pred_rf) )
print( scores('RandomForest - Validação', y_val, y_val_pred_rf) )
print()
print( scores('AdaBoost - Treino' , y_trn, y_trn_pred_ada) )
print( scores('AdaBoost - Validação', y_val, y_val_pred_ada) )
print()
print( scores('LightGBM - Treino' , y_trn, y_trn_pred_lg) )
print( scores('LightGBM - Validação', y_val, y_val_pred_lg) )
```

*Regressão Logística - Treino:*  
Acc: 63.39%, Precision: 64.83%, Recall: 36.86%  
*Regressão Logística - Validação:*  
Acc: 63.04%, Precision: 65.12%, Recall: 36.55%

*Árvore de Decisão - Treino:*  
Acc: 68.63%, Precision: 74.44%, Recall: 43.81%  
*Árvore de Decisão - Validação:*  
Acc: 61.63%, Precision: 61.85%, Recall: 36.00%

*RandomForest - Treino:*  
Acc: 68.63%, Precision: 72.98%, Recall: 45.66%  
*RandomForest - Validação:*  
Acc: 61.97%, Precision: 61.75%, Recall: 38.24%

*AdaBoost - Treino:*  
Acc: 63.73%, Precision: 65.54%, Recall: 37.18%  
*AdaBoost - Validação:*  
Acc: 63.58%, Precision: 66.28%, Recall: 37.01%

*LightGBM - Treino:*  
Acc: 64.45%, Precision: 66.37%, Recall: 39.05%  
*LightGBM - Validação:*  
Acc: 63.70%, Precision: 65.78%, Recall: 38.43%

## Conclusão:

*Dado o objeto do problema, escolho Recall como métrica mais relevante para a comparação dos modelos.*

*Isso não quer dizer que as demais variáveis serão desconsideradas, mas avaliadas com menor relevância.*

*Assim sendo:*

*LightGBM tem o maior Recall e é o modelo escolhido.*

```
In [65]: # Liberando memória
Lr , y_trn_pred_lr , y_val_pred_lr = None, None, None
dt , y_trn_pred_dt , y_val_pred_dt = None, None, None
rf , y_trn_pred_rf , y_val_pred_rf = None, None, None
ada, y_trn_pred_ada, y_val_pred_ada = None, None, None
```

## Otimização do Modelo

*Como primeiro passo, exploramos os hiperparâmetros do modelo que produzam um melhor resultado. Para isso, utilizaremos **GridSearch**.*

*Devido ao tempo de processamento, uma abordagem de refinamentos sucessivos é empregada. </spam>*

```
In [66]: from sklearn.model_selection import GridSearchCV
```

```
In [67]: # Vamos buscar otimizar Precision  
metricas_refit = 'precision,accuracy,recall'  
metraca_otim = 'recall'
```

**ALERTA: Execução leva vários minutos**

```
In [68]: # O primeiro ciclo é para definir boosting type, importance type, unbalance e max_depth  
lgbm = LGBMClassifier()  
parameters = {  
    'boosting_type' : ['gbdt', 'dart', 'goss', 'rf'],  
    'importance_type': ['split', 'gain'],  
    'is_unbalance' : [True, False],  
    'max_depth' : [3, 5, 10],  
    'random_state' : [12]  
}  
lgo1 = GridSearchCV(lgbm, parameters, cv=5, refit=metricas_refit, verbose=1, scoring=metraca_otim)  
lgo1.fit(X_trn, y_trn)
```

Fitting 5 folds for each of 48 candidates, totalling 240 fits

```
Out[68]: GridSearchCV(cv=5, estimator=LGBMClassifier(),  
param_grid={'boosting_type': ['gbdt', 'dart', 'goss', 'rf'],  
           'importance_type': ['split', 'gain'],  
           'is_unbalance': [True, False], 'max_depth': [3, 5, 10],  
           'random_state': [12]},  
refit='precision,accuracy,recall', scoring='recall', verbose=1)
```

```
In [69]: y_trn_pred, y_val_pred, _ = pred_fats_cm(lgo1.best_estimator_, X_trn, y_trn, X_val, y_val)
```

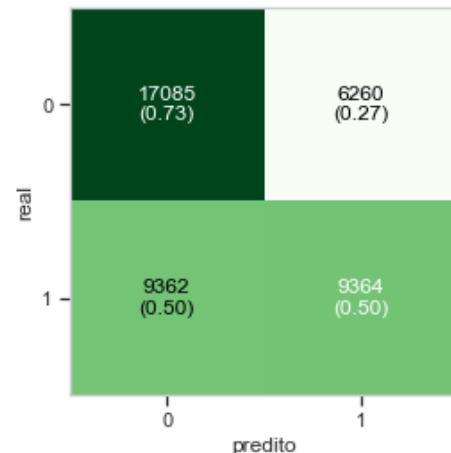
Treino:

Acc: 63.35%, Precision: 59.86%, Recall: 50.91%

Validação:

Acc: 62.87%, Precision: 59.93%, Recall: 50.01%

<Figure size 252x252 with 0 Axes>



```
In [70]: lgo1.best_params_
```

```
Out[70]: {'boosting_type': 'goss',
          'importance_type': 'split',
          'is_unbalance': True,
          'max_depth': 3,
          'random_state': 12}
```

```
In [71]: # Definidos:
BTTYPE = 'gbdt'
ITYPE = 'split'
UNBAL = True
MDEPT = 3
```

**ALERTA: Execução leva MUITOS minutos**

```
In [80]: # Ciclo 2: refinamento de hiperparametros
parameters = {
    # Definidos
    'boosting_type' : [BTTYPE],
    'importance_type' : [ITYPE],
    'is_unbalance' : [UNBAL],
    'max_depth' : [MDEPT],
    'objective' : ['binary'],
    'random_state' : [12],
    # Refinar
    'learning_rate' : [0.1, 0.2, 0.3],
    'min_child_samples' : [1, 5, 10],
    'n_estimators' : [50, 100, 150, 200]
}
Lgo2 = GridSearchCV(Lgbm, parameters, cv=5, refit=metricas_refit, verbose=1, scoring=metrica_otim)
Lgo2.fit(X_trn, y_trn)
```

Fitting 5 folds for each of 36 candidates, totalling 180 fits

```
Out[80]: GridSearchCV(cv=5, estimator=LGBMClassifier(),
                      param_grid={'boosting_type': ['gbdt'],
                                  'importance_type': ['split'], 'is_unbalance': [True],
                                  'learning_rate': [0.1, 0.2, 0.3], 'max_depth': [3],
                                  'min_child_samples': [1, 5, 10],
                                  'n_estimators': [50, 100, 150, 200],
                                  'objective': ['binary'], 'random_state': [12]},
                      refit='precision', accuracy, recall', scoring='recall', verbose=1)
```

```
In [81]: y_trn_pred, y_val_pred, _ = pred_fats_cm(Lgo2.best_estimator_, X_trn, y_trn, X_val, y_val)
```

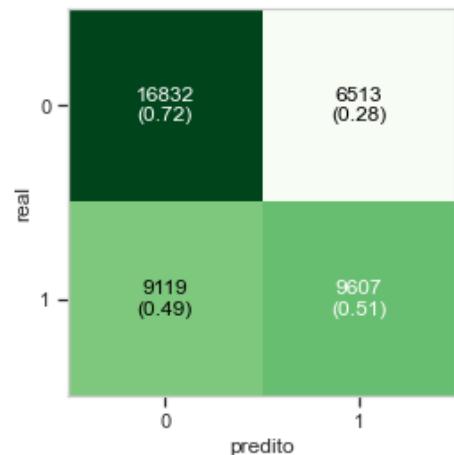
Treino:

Acc: 63.39%, Precision: 59.68%, Recall: 51.98%

Validação:

Acc: 62.84%, Precision: 59.60%, Recall: 51.30%

<Figure size 252x252 with 0 Axes>



In [82]: `lgo2.best_params_`

Out[82]:

```
{'boosting_type': 'gbdt',
 'importance_type': 'split',
 'is_unbalance': True,
 'Learning_rate': 0.3,
 'max_depth': 3,
 'min_child_samples': 5,
 'n_estimators': 150,
 'objective': 'binary',
 'random_state': 12}
```

In [85]:

```
# Definido:
LRATE = 0.3
MCSAM = 5
NESTM = 150
```

## Conclusão:

Essa é a configuração com melhor performance, e que utilizaremos a partir daqui:

In [86]:

```
print(f'BTYPE = boosting_type      = {BTYPE}')
print(f'ITYPE = importance_type     = {ITYPE}')
print(f'UNBAL = unbalance            = {UNBAL}')
print(f'LRATE = Learning_rate        = {LRATE}')
print(f'MDEPT = max_depth            = {MDEPT}')
print(f'MCSAM = min_child_samples    = {MCSAM}')
print(f'NESTM = n_estimators          = {NESTM}')
```

```
BTYPE = boosting_type      = gbdt
ITYPE = importance_type   = split
UNBAL = unbalance          = True
LRATE = Learning_rate      = 0.3
MDEPT = max_depth           = 3
MCSAM = min_child_samples   = 5
NESTM = n_estimators        = 150
```

## Comentário:

Ao ajustar os parâmetros, interferimos nos quatro resultados possíveis (negativos, falsos-negativos, positivos e falsos-positivos). Ao melhorar a identificação de positivos, pioramos a identificação de negativos.

Considerando o propósito de identificar pacientes com covid, é melhor termos mais falsos-positivos do que falsos-negativos, e foi nesse sentido que os parâmetros foram testados até aqui.

## Otimização com seleção de features

### Baseado no modelo

```
In [87]: lgbm = LGBMClassifier(boosting_type=BTYPE, importance_type=ITYPE, unbalance=UNBAL, Learning_rate=LRATE,
                           max_depth=MDEPT, min_child_samples=MCSAM, n_estimators=NESTM)
lgbm.fit(X_trn, y_trn)
```

```
Out[87]: LGBMClassifier(Learning_rate=0.3, max_depth=3, min_child_samples=5,
                        n_estimators=150, unbalance=True)
```

```
In [88]: fi = pd.DataFrame(X_trn.columns, lgbm.feature_importances_)
fi.reset_index(inplace=True)
fi = fi.rename(mapper={'index': 'importancia', 0: 'feature'}, axis=1)
fi.sort_values(by='importancia', ascending=False, inplace=True)
fi
```

Out[88]:

	<i>importancia</i>	<i>feature</i>
4	355	<i>age</i>
16	171	<i>contact_other_covid</i>
2	45	<i>intubed</i>
0	43	<i>sex</i>
14	39	<i>renal_chronic</i>
6	38	<i>diabetes</i>
9	37	<i>inmsupr</i>
3	36	<i>pneumonia</i>
10	35	<i>hypertension</i>
13	35	<i>obesity</i>
1	31	<i>patient_type</i>
17	30	<i>icu</i>
11	28	<i>other_disease</i>
7	26	<i>copd</i>
12	24	<i>cardiovascular</i>
15	24	<i>tobacco</i>
5	17	<i>pregnancy</i>
8	16	<i>asthma</i>

## Forward/Backward selection

In [89]:

```
from sklearn.feature_selection import SequentialFeatureSelector
```

In [90]:

```
fs = SequentialFeatureSelector(Lgbm, n_features_to_select=0.7, cv=2, scoring=metrica_otim, n_jobs=-1)
```

**ALERTA: Execução leva MUITOS minutos !!!**

In [91]:

```
fs.fit(X_trn, y_trn)
```

```
Out[91]: SequentialFeatureSelector(cv=2,
                                    estimator=LGBMClassifier(learning_rate=0.3,
                                                             max_depth=3,
                                                             min_child_samples=5,
                                                             n_estimators=150,
                                                             unbalance=True),
                                    n_features_to_select=0.7, n_jobs=-1,
                                    scoring='recall')
```

```
In [92]: sel_bfs = fs.get_support(True)
sel_bfs
```

```
Out[92]: array([ 4,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16], dtype=int64)
```

```
In [93]: # Variáveis selecionadas por backward/forward selection
features_bfs = np.array(X_trn.columns[sel_bfs])
features_bfs
```

```
Out[93]: array(['age', 'diabetes', 'copd', 'asthma', 'inmsupr', 'hypertension',
   'other_disease', 'cardiovascular', 'obesity', 'renal_chronic',
   'tobacco', 'contact_other_covid'], dtype=object)
```

```
In [94]: # Obtendo as variáveis do modelo na mesma quantidade das features selecionadas por backward/forward selection
features_fm = np.array(fi['feature'])[:len(features_bfs)]
features_fm
```

```
Out[94]: array(['age', 'contact_other_covid', 'intubed', 'sex', 'renal_chronic',
   'diabetes', 'inmsupr', 'pneumonia', 'hypertension', 'obesity',
   'patient_type', 'icu'], dtype=object)
```

## Comentários:

Nota-se uma diferença entre as features selecionadas do modelo e as features selecionadas por **Backward/Forward selection**. É necessário avaliar os resultados do modelo em cada cenário.

```
In [95]: # Melhores features pelo modelo
X_trn_fm = X_trn[features_fm]
X_val_fm = X_val[features_fm]

# Melhores features pelo B/F selection
X_trn_bfs = X_trn[features_bfs]
X_val_bfs = X_val[features_bfs]

# Conferindo os shapes
print(X_trn_fm.shape, y_trn.shape)
print(X_val_fm.shape, y_val.shape)

print(X_trn_bfs.shape, y_trn.shape)
print(X_val_bfs.shape, y_val.shape)
```

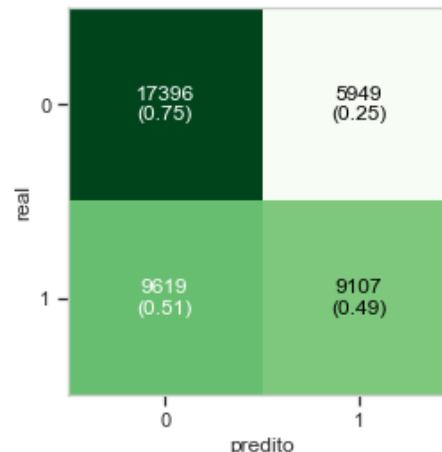
```
(378634, 12) (378634, )
(42071, 12) (42071, )
(378634, 12) (378634, )
(42071, 12) (42071, )
```

```
In [96]: # Métricas com as features do modelo
lgbm_fm = LGBMClassifier(boosting_type=BTYPE, importance_type=ITYPE, unbalance=UNBAL, Learning_rate=LRATE,
                         max_depth=MDEPTH, min_child_samples=MCSAM, n_estimators=NESTM)
lgbm_fm.fit(X_trn_fm, y_trn)
```

```
Out[96]: LGBMClassifier(Learning_rate=0.3, max_depth=3, min_child_samples=5,
                        n_estimators=150, unbalance=True)
```

```
In [97]: print("Com as Features do Modelo:")
y_trn_pred_fm, y_val_pred_fm, _ = pred_fats_cm(lgbm_fm, X_trn_fm, y_trn, X_val_fm, y_val)
```

```
Com as Features do Modelo:
Treino:
Acc: 63.43%, Precision: 60.39%, Recall: 49.29%
Validação:
Acc: 63.00%, Precision: 60.49%, Recall: 48.63%
<Figure size 252x252 with 0 Axes>
```



```
In [98]: # Métricas com as features do B/F selection
lgbm_bfs = LGBMClassifier(boosting_type=BTYPE, importance_type=ITYPE, unbalance=UNBAL, Learning_rate=LRATE,
                           max_depth=MDEPTH, min_child_samples=MCSAM, n_estimators=NESTM)
lgbm_bfs.fit(X_trn_bfs, y_trn)
```

```
Out[98]: LGBMClassifier(Learning_rate=0.3, max_depth=3, min_child_samples=5,
                        n_estimators=150, unbalance=True)
```

```
In [99]: print("Com as Features de B/F Selection:")
y_trn_pred_bfs, y_val_pred_bfs, _ = pred_fats_cm(lgbm_bfs, X_trn_bfs, y_trn, X_val_bfs, y_val)
```

Com as Features de B/F Selection:

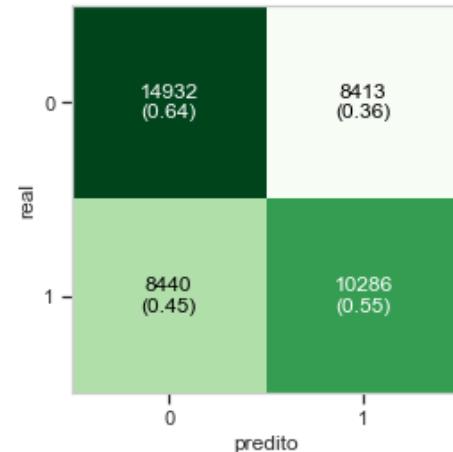
Treino:

Acc: 60.46%, Precision: 55.04%, Recall: 55.66%

Validação:

Acc: 59.94%, Precision: 55.01%, Recall: 54.93%

<Figure size 252x252 with 0 Axes>



Comentário:

As features indicadas pelo método de backward/forward selection produziram melhores resultados.

## Personalizando os cortes das variáveis explicativas

**Uma função para calcular probabilidade limitada a um fator de corte**

In [100...]

```
def prob(y_proba, corte):
    probs = y_proba.copy()
    mask = (probs >= corte)
    probs[mask] = 1
    probs[~mask] = 0
    return probs
```

Plot das métricas com as variáveis do modelo e variáveis de B/F Selection

**Features do modelo**

In [101...]

```
y_proba_trn_fm = Lgbm_fm.predict_proba(X_trn_fm)[:,1]
pd.DataFrame(y_proba_trn_fm).value_counts()
```

```
Out[101]: 0.361761    7921
0.411352    6619
0.474717    4486
0.493936    4478
0.362274    4282
...
0.586199      1
0.586220      1
0.586235      1
0.586284      1
0.613405      1
Length: 19013, dtype: int64
```

### Features de B/F selection

```
In [102...]: y_proba_trn_bfs = Lgbm_bfs.predict_proba(X_trn_bfs)[:,1]
pd.DataFrame(y_proba_trn_bfs).value_counts()
```

```
Out[102]: 0.414328    8075
0.405443    7693
0.378777    6390
0.387984    6245
0.509388    5372
...
0.489890      1
0.489891      1
0.489907      1
0.489954      1
0.844648      1
Length: 15451, dtype: int64
```

**ALERTA:** Execução leva alguns minutos</spam>

```
In [103...]: cortes      = []
acuracias_fm = []
precisoes_fm = []
recalls_fm   = []

acuracias_bfs = []
precisoes_bfs = []
recalls_bfs   = []

for corte in np.linspace(0, 1, 101):
    cortes.append(corte)

    y_prob = prob(y_proba_trn_fm, corte=corte)

    acc = accuracy_score(y_trn, y_prob)
    pre = precision_score(y_trn, y_prob)
    rec = recall_score(y_trn, y_prob)
```

```

acuracias_fm.append(acc)
precisoes_fm.append(pre)
recalls_fm.append(rec)

y_prob = prob(y_proba_trn_bfs, corte=corte)

acc = accuracy_score(y_trn, y_prob)
pre = precision_score(y_trn, y_prob)
rec = recall_score(y_trn, y_prob)

acuracias_bfs.append(acc)
precisoes_bfs.append(pre)
recalls_bfs.append(rec)

```

In [104...]

```

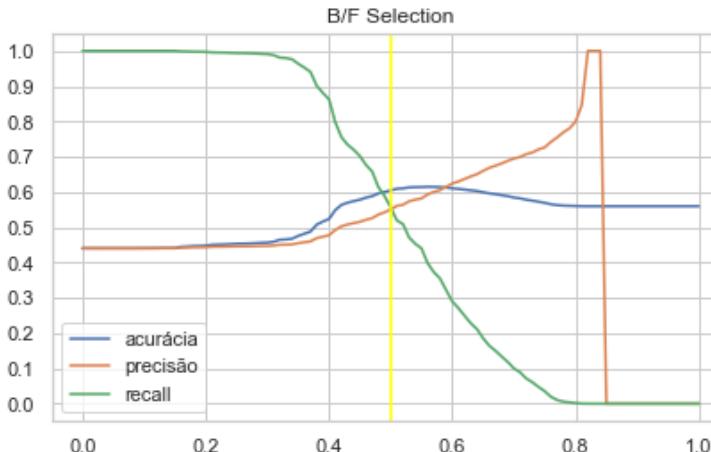
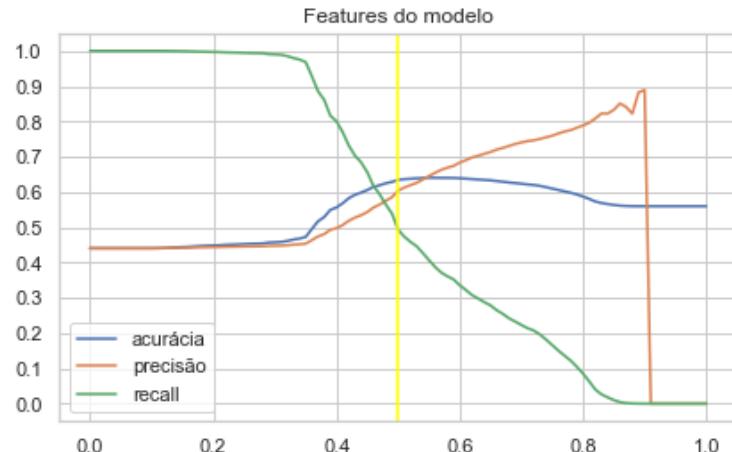
# Grid para dois gráficos
_, axes = plt.subplots(1,2, figsize=(15,4), sharey=True)

# Fixando as marcas do eixo Y
y_ticks = [x/10 for x in range(0, 11)]

# Gráfico das métricas de variáveis selecionadas pelo método do modelo (LightGBM)
plt.subplot(1,2,1)
plt.plot(cortes, acuracias_fm, label='acurácia')
plt.plot(cortes, precisoes_fm, label='precisão')
plt.plot(cortes, recalls_fm, label='recall' )
plt.axvline(0.50, color='yellow')
plt.title("Features do modelo")
plt.yticks(y_ticks)
plt.legend()

# Gráfico das métricas de variáveis selecionadas pelo método de B/W Selection
plt.subplot(1,2,2)
plt.plot(cortes, acuracias_bfs, label='acurácia')
plt.plot(cortes, precisoes_bfs, label='precisão')
plt.plot(cortes, recalls_bfs, label='recall' )
plt.axvline(0.50, color='yellow')
plt.title("B/F Selection")
plt.legend()
plt.yticks(y_ticks)
plt.show()

```



## Conclusão:

Os gráficos demonstram o comportamento das três métricas com todos os cortes, e em destaque (amarelo) o corte padrão.

Fica fácil compreender, que o corte ideal está antes do corte padrão. Assim, a seguir as métricas para os cortes anteriores próximos a 0.5.

```
In [120...]: #Métricas no ponto de maior acurácia
for fator in range(41, 48):
    corte = fator / 100

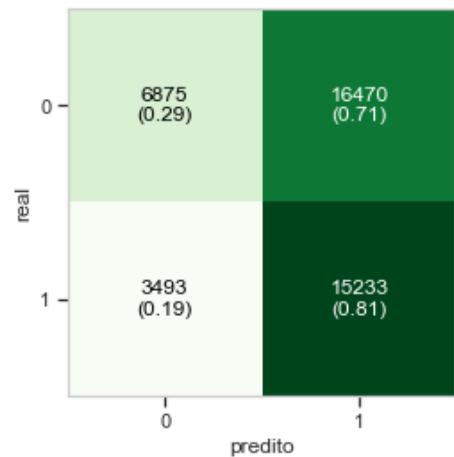
    y_proba_val = lgbm_bfs.predict_proba(X_val_bfs)[:,1]
    y_prob_val = prob(y_proba_val, corte=corte)

    print(scores(f'Corte {corte} - BF Sel.', y_val, y_prob_val))
    cm(y_val, y_prob_val)
```

Corte 0.41 - BF Sel.:

Acc: 52.55%, Precision: 48.05%, Recall: 81.35%

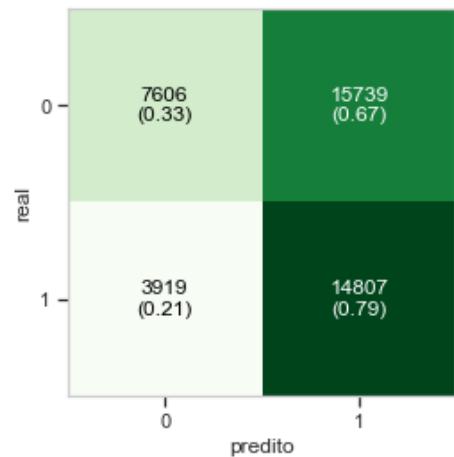
<Figure size 252x252 with 0 Axes>



Corte 0.42 - BF Sel.:

Acc: 53.27%, Precision: 48.47%, Recall: 79.07%

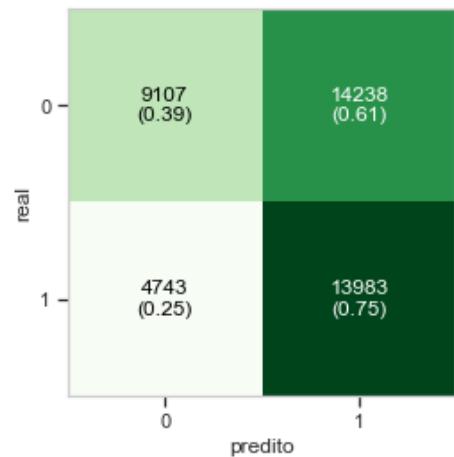
<Figure size 252x252 with 0 Axes>



Corte 0.43 - BF Sel.:

Acc: 54.88%, Precision: 49.55%, Recall: 74.67%

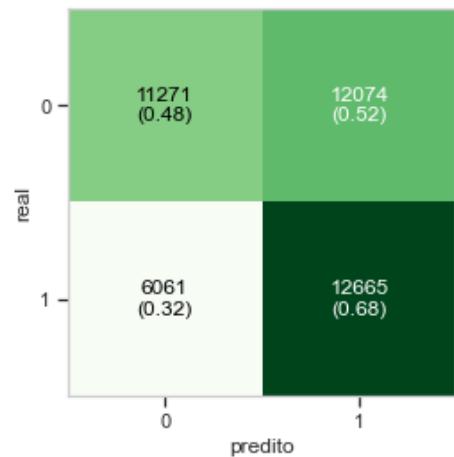
<Figure size 252x252 with 0 Axes>



Corte 0.44 - BF Sel.:

Acc: 56.89%, Precision: 51.19%, Recall: 67.63%

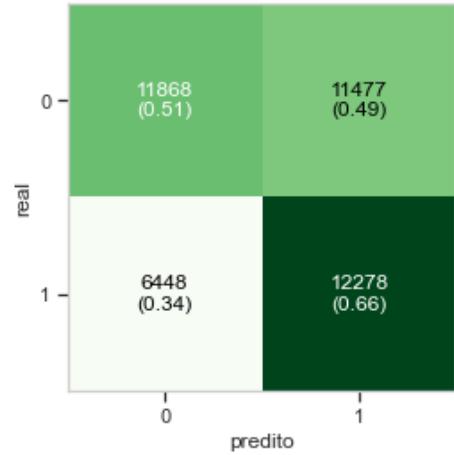
<Figure size 252x252 with 0 Axes>



Corte 0.45 - BF Sel.:

Acc: 57.39%, Precision: 51.69%, Recall: 65.57%

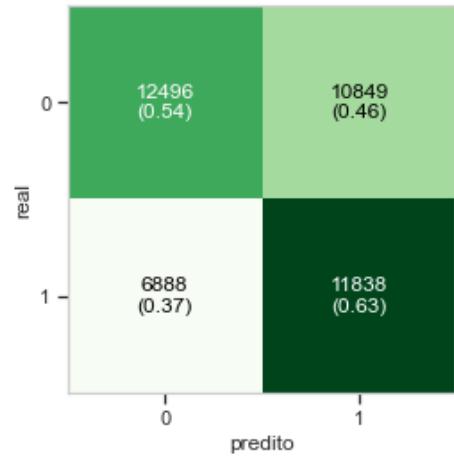
<Figure size 252x252 with 0 Axes>



Corte 0.46 - BF Sel.:

Acc: 57.84%, Precision: 52.18%, Recall: 63.22%

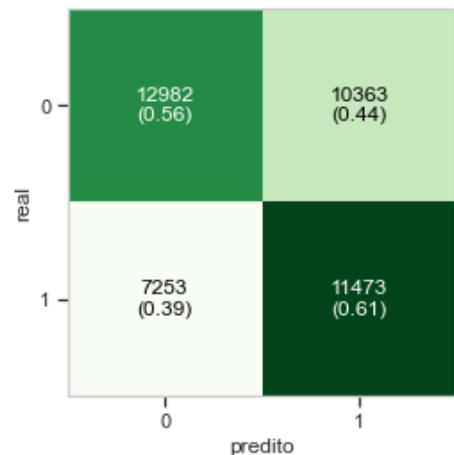
<Figure size 252x252 with 0 Axes>



Corte 0.47 - BF Sel.:

Acc: 58.13%, Precision: 52.54%, Recall: 61.27%

<Figure size 252x252 with 0 Axes>



## Comentário:

As matrizes de confusão definem que o melhor ponto de corte (opinião pessoal) é 0.44.

É o ponto onde temos o baixo número de falsos positivos, sem errar demais com falsos negativos.

Porém, numa decisão em grupo, outros pontos próximos poderiam ser considerados.

```
In [121...]
# Selecionando somente as melhores features do modelo para as três bases (treino, validação e teste)
lgbm_otim = lgbm_bfs

X_trn_otim = X_trn_bfs
X_val_otim = X_val_bfs
X_tst_otim = X_tst[features_bfs]

CORTE = 0.44
```

```
In [122...]
# Métricas no ponto de maior acurácia
# Como feito com a base de treino, calculamos as probabilidades para a base de validação
y_proba_trn = lgbm_otim.predict_proba(X_trn_otim)[:,1]
y_proba_val = lgbm_otim.predict_proba(X_val_otim)[:,1]

#Métricas no ponto de maior acurácia
y_prob_trn = prob(y_proba_trn, corte=CORTE)
y_prob_val = prob(y_proba_val, corte=CORTE)

print(scores('Treino', y_trn, y_prob_trn))
print(scores('Validação', y_val, y_prob_val))
cm(y_val, y_prob_val)
```

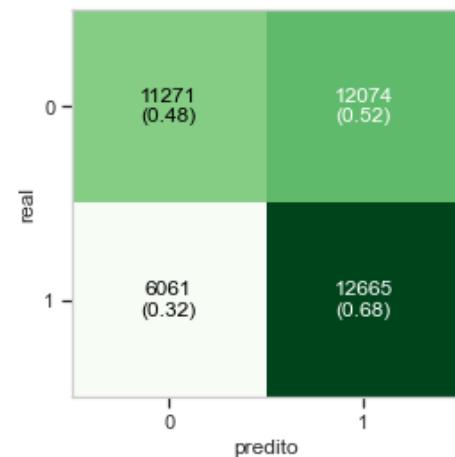
Treino:

Acc: 57.00%, Precision: 50.88%, Recall: 67.98%

Validação:

Acc: 56.89%, Precision: 51.19%, Recall: 67.63%

<Figure size 252x252 with 0 Axes>



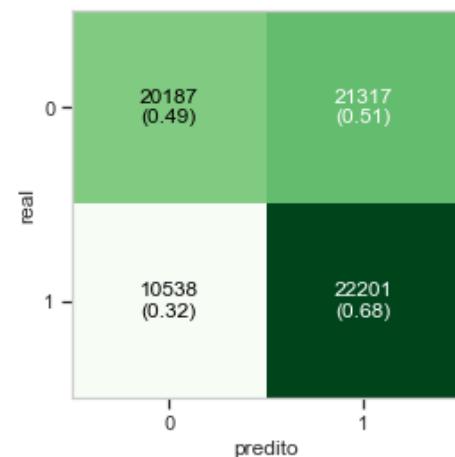
## Verificando com a base de teste

```
In [123...]: # Empregando o corte de probabilidade customizado  
y_proba_tst = lgbm_otim.predict_proba(X_tst_otim)[:,1]  
  
y_prob_tst = prob(y_proba_tst, corte=CORTE)  
  
print(scores('Teste', y_tst, y_prob_tst))  
  
cm(y_tst, y_prob_tst)
```

Teste:

Acc: 57.09%, Precision: 51.02%, Recall: 67.81%

<Figure size 252x252 with 0 Axes>



## Avaliando com remoção dos registros duplicados

In [124..]

```
# Eliminando as duplicidades
Xsd = df.drop(X.drop_duplicates().index)

# Separando a variável-alvo das features explicativas
ysd = Xsd[target]
Xsd = Xsd.drop(target, axis=1)

# Aplicando as features selecionadas
Xsd_otim = Xsd[features_bfs]
```

In [125..]

```
# Separando os dados de treino, validação e teste
XX, Xsd_tst, yy, ysd_tst = train_test_split(Xsd_otim, ysd, test_size=PERC_TST, random_state=12)

Xsd_trn, Xsd_val, ysd_trn, ysd_val = train_test_split(XX, yy, test_size=PERC_VAL, random_state=12)

XX = None
yy = None

print('Shapes:')
print(f'treino    = X:{Xsd_trn.shape}, y:{ysd_trn.shape}')
print(f'validação = X:{Xsd_val.shape}, y:{ysd_val.shape}')
print(f'teste     = X:{Xsd_tst.shape}, y:{ysd_tst.shape}')

Shapes:
treino    = X:(336745, 12), y:(336745,)
validação = X:(37417, 12), y:(37417,)
teste     = X:(66029, 12), y:(66029,)
```

In [126..]

```
# Fit do modelo para a nova base de treino
Lgbm_otim.fit(Xsd_trn, ysd_trn)
```

Out[126]:

```
LGBMClassifier(Learning_rate=0.3, max_depth=3, min_child_samples=5,
                n_estimators=150, unbalance=True)
```

In [127..]

```
y_proba_trn = lgbm_otim.predict_proba(Xsd_trn)[:,1]
```

In [128..]

```
acuracias_bfs = []
precisoes_bfs = []
recalls_bfs   = []
cortes_bfs    = []

for corte in np.linspace(0, 1, 101):
    y_prob = prob(y_proba_trn, corte=corte)

    acc = accuracy_score(ysd_trn, y_prob)
    pre = precision_score(ysd_trn, y_prob)
    rec = recall_score(ysd_trn, y_prob)
```

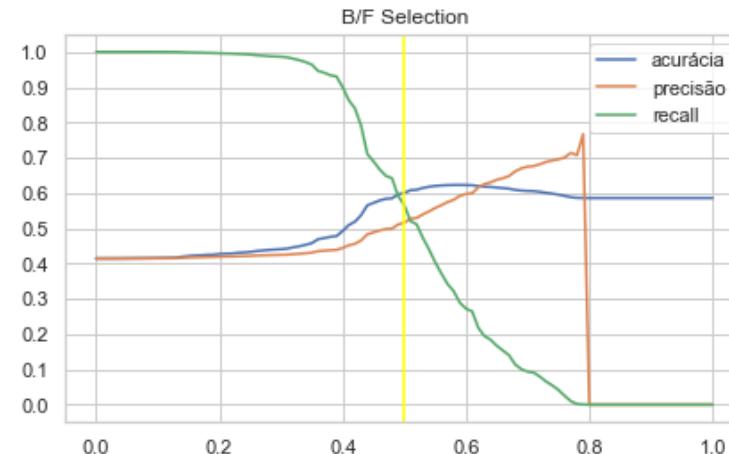
```
cortes_bfs.append(corte)
acuracias_bfs.append(acc)
precisoes_bfs.append(pre)
recalls_bfs.append(rec)
```

```
In [129...]: # Grid para dois gráficos
_, axes = plt.subplots(1,1, figsize=(15,4), sharey=True)
```

```
# Fixando as marcas do eixo Y
y_ticks = [x/10 for x in range(0, 11)]
```

```
# Gráfico das métricas de variáveis selecionadas pelo método de B/W Selection
plt.subplot(1,2,2)
```

```
plt.plot(cortes_bfs, acuracias_bfs, label='acurácia')
plt.plot(cortes_bfs, precisoes_bfs, label='precisão')
plt.plot(cortes_bfs, recalls_bfs, label='recall' )
plt.axvline(0.50, color='yellow')
plt.title("B/F Selection")
plt.legend()
plt.yticks(y_ticks)
plt.show()
```



```
In [130...]: # Relembrando quantos não-infectados e infectados há na amostra
ysd_trn.value_counts(normalize=True)
```

```
Out[130]: 0    0.585695
1    0.414305
Name: covid_res, dtype: float64
```

```
In [131...]: # Empregando o corte de probabilidade customizado
ysd_proba_trn = lgmb_otim.predict_proba(Xsd_trn)[:,1]
ysd_proba_val = lgmb_otim.predict_proba(Xsd_val)[:,1]

ysd_prob_trn = prob(ysd_proba_trn, corte=CORTE)
```

```

ysd_prob_val = prob(ysd_proba_val, corte=CORTE)

print(scores('Treino' , ysd_trn, ysd_prob_trn))
print(scores('Validação', ysd_val, ysd_prob_val))
cm(ysd_val, ysd_prob_val)

```

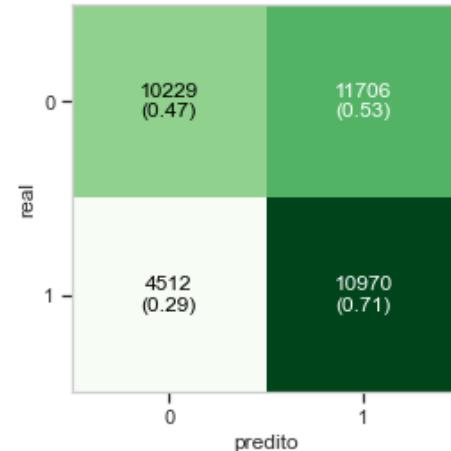
Treino:

Acc: 56.51%, Precision: 48.31%, Recall: 71.06%

Validação:

Acc: 56.66%, Precision: 48.38%, Recall: 70.86%

<Figure size 252x252 with 0 Axes>



## Verificando com os dados de Teste

```

In [132]: ysd_proba_tst = lgbm_otim.predict_proba(Xsd_tst)[:,1]

ysd_prob_tst = prob(ysd_proba_tst, corte=CORTE)

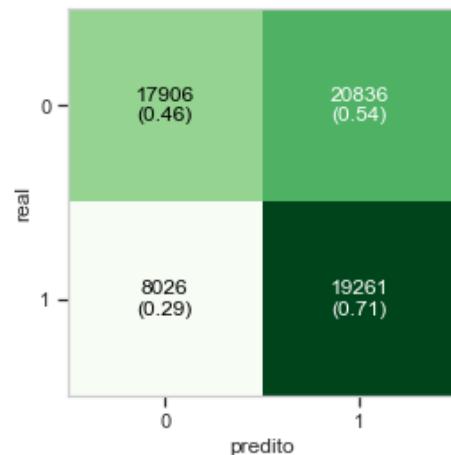
print(scores('Teste', ysd_tst, ysd_prob_tst))
cm(ysd_tst, ysd_prob_tst)

```

Teste:

Acc: 56.29%, Precision: 48.04%, Recall: 70.59%

<Figure size 252x252 with 0 Axes>



## Avaliação: Com duplicidades x Sem duplicidades

*Com duplicidades na base, alcançamos os seguintes resultados no modelo final otimizado:*

In [135...]

```
print(scores('Treino' , y_trn, y_prob_trn))
print(scores('Validação', y_val, y_prob_val))
print(scores('Teste'   , y_tst, y_prob_tst))
```

Treino:

Acc: 57.00%, Precision: 50.88%, Recall: 67.98%

Validação:

Acc: 56.89%, Precision: 51.19%, Recall: 67.63%

Teste:

Acc: 57.09%, Precision: 51.02%, Recall: 67.81%

Já com sem duplicidades na base, com o mesmo modelo alcançamos os seguintes resultados:

In [136...]

```
print(scores('Treino' , ysd_trn, ysd_prob_trn))
print(scores('Validação', ysd_val, ysd_prob_val))
print(scores('Teste'   , ysd_tst, ysd_prob_tst))
```

Treino:

Acc: 56.51%, Precision: 48.31%, Recall: 71.06%

Validação:

Acc: 56.66%, Precision: 48.38%, Recall: 70.86%

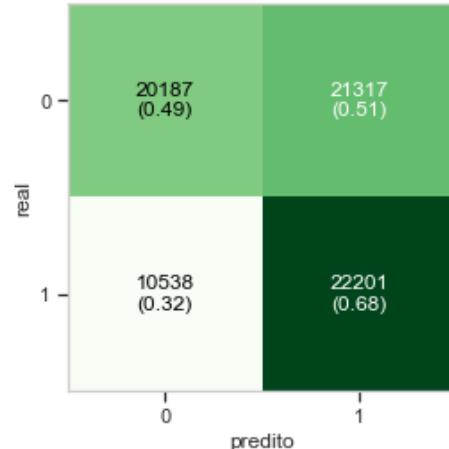
Teste:

Acc: 56.29%, Precision: 48.04%, Recall: 70.59%

In [137...]

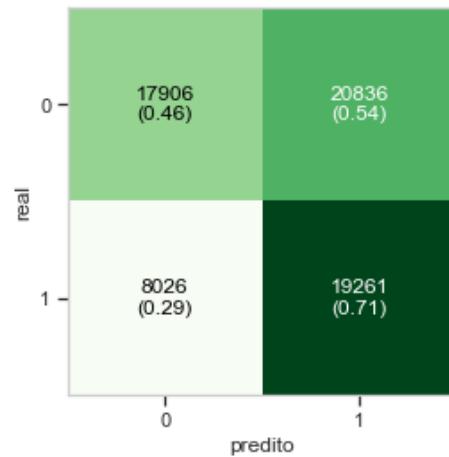
```
cm(y_tst, y_prob_tst)
```

<Figure size 252x252 with 0 Axes>



In [138]: `cm(ysd_tst, ysd_prob_tst)`

<Figure size 252x252 with 0 Axes>



## Conclusão:

*Considerando o objetivo de acertar mais diagnósticos positivos, o modelo sem duplicidades de registros teve um melhor desempenho, sendo portanto, a melhor opção.*

Resumo:

- Tratamento de dados com exclusão de registros duplicados.
- Utilização do modelo LightGBM com otimização para melhorar Recall.
- Seleção de 70% das features por backward/forward selection.
- Utilização de ponto de corte ideal 0.44, devido ao desbalanceamento entre positivos e negativos.

# Conclusões sobre o Projeto

## Preparação dos Dados e Verificação de Consistência

*É essencial interpretar as features corretamente. Conhecer o universo de valores possíveis e interdependências é essencial para realizar correções assertivas.*

*Esse projeto evidenciou o risco de erros, caso o cientista não reflita sobre o impacto que o valor de uma feature pode ter em outras. A simples exclusão de todos os registros com alguma informação faltante não deve ser feita.*

## Análise Exploratória dos Dados

*Uma visualização correta do "comportamento" de cada feature é muito importante.*

*Podemos ter insights sobre dados que podem comprometer o modelo, e variáveis muito semelhantes, que indicam que podemos trabalhar com menos features sem comprometer o modelo.*

*A experiência do cientista é essencial, para determinar quais informações e quais gráficos devem ser avaliados.*

*Confesso que tive dificuldade em selecionar gráficos para dados binários, e a análise do pairplot ainda é algo que devo aprimorar.*

*Me pareceu não fazer sentido uma análise de outliers com dados puramente binários, portanto, não a fiz.*

## Modelagem dos Dados

*É essencial que o cientista domine sobre tipos de variáveis (explicativas e alvo) e os modelos mais adequados para cada cenário.*

*A seleção do melhor modelo também depende da vivência e conhecimento do cientista. Aqui, puramente selecionei pelas métricas resultantes numa execução sem otimização, para selecionar um para a fase de otimização, como orienta o enunciado do projeto. Penso que num cenário real, algumas otimizações dos diferentes modelos precisariam ser realizadas para uma escolha mais embasada.*

## Otimização do Modelo

*Sem dúvida, uma fase crucial para qualquer bom modelo. O cientista precisa dominar os modelos e o impacto dos parâmetros.*

*Há muita experimentação envolvida, e eu concluí que:*

- a) *Não se deve testar com subconjuntos da base de treino. Amostras diferentes produzem resultados diferentes.*
- b) *Poder de processamento é essencial. Restrições de tempo devem ser consideradas.*
- c) *Muito cuidado deve ser tomado no processo de ajustes de parâmetros a testar. Preferencialmente, deve-se alterar apenas um parâmetro entre um teste e outro.*
- d) *Não se deve deixar de testar cada resultado. As ferramentas não evitam overfitting, e saídas "otimizadas" podem, com frequência, não resultarem em melhor performance do modelo.*
- e) *É fundamental decidir, já no início, o que é melhor para o modelo: errar menos com os falso-positivos ou errar menos com os falso-negativos.*

*Nota: além dos testes documentados nesse projeto, fiz inúmeros outros, ora priorizando acurácia, ora precisão; Todos os testes levam a resultados distintos, e sem critério, essa fase pode prolongar-se por muito tempo.*

## Conclusão

*Para um modelo que visa identificar pacientes positivos para covid, uma precisão acima de 50% com base apenas em sintomas, sem um exame objetivo para a doença, é bastante razoável. Certamente poderia ser utilizado para priorização de populações a tratar.*

*Como já citado aqui, definir as métricas ideais a otimizar, e a importância dos falso-negativos e falso positivos é essencial.*

*Bons modelos dependem essencialmente de bons cientistas e bons métodos. Técnica é fundamental para evitar, identificar e tratar erros e otimizar o tempo requerido para a construção.*  
*</span>*

In [ ]: