

 [dipucriodigital / ciencia-de-dados-e-analytics](#) Público[!\[\]\(666e09182d4cd268646ea700ea60dcdf_img.jpg\) Código](#) [!\[\]\(1ef1ef0bf9af6c6996401964cf280f2d_img.jpg\) Problemas](#) [!\[\]\(e9a80c8557f9285916925bd4ac40fff5_img.jpg\) Requisições pull](#) [!\[\]\(88e2edecff3400e68a80dd08c57d2f9c_img.jpg\) Ações](#) [!\[\]\(b612b1233231807474dc279cea9675f1_img.jpg\) Projetos](#) [!\[\]\(b896133b8558947b57dae58856c2941f_img.jpg\) Segurança](#) [!\[\]\(072e40399fb05077a449247099f420e9_img.jpg\) Perc](#)Beta Experimente a nova visualização de código principal ▾

...

[ciencia-de-dados-e-analytics / mvp-analise-de-dados-e-boas-praticas /](#)
[MVP_CD_Diabetes.ipynb](#)

tatianaesc Criado usando o Colaboratory

 1 colaborador

2,34 MB

...

[Open in Colab](#)

MVP de Análise de Dados e Boas Práticas

Profs. Tatiana Escovedo e Hugo Villamizar

Exemplo de um MVP

O que não está detalhado ou pode ser melhorado neste notebook para ficar como esperamos para o MVP:

- Blocos de texto que expliquem textualmente cada etapa e cada decisão do seu código, contando uma história completa e compreensível, do início ao fim;
- Boas práticas de codificação;
- Após cada gráfico, escrever 1 parágrafo resumindo os principais achados, analisando os resultados e levantando eventuais pontos de atenção.

1. Definição do Problema

O dataset usado neste projeto será o **Pima Indians Diabetes**, proveniente originalmente do Instituto Nacional de Diabetes e Doenças Digestivas e Renais. Seu objetivo é prever se um paciente tem ou não diabetes, com base em certas medidas de diagnóstico médico. Este dataset é um subconjunto do dataset original e aqui, todos os pacientes são mulheres com pelo menos 21 anos de idade e de herança indígena Pima. O dataset apresenta em diversos atributos relacionados a dados médicos e uma variável de classe binária (0 ou 1). As variáveis preditoras incluem o número de gestações que a paciente teve, seu IMC, nível de insulina, idade e assim por diante. Para mais detalhes sobre este dataset, consulte: <https://www.kaggle.com/uciml/pima-indians-diabetes-database>

Informações sobre os atributos:

1. **preg** - Número de vezes grávida
2. **plas** - Concentração de glicose no plasma
3. **pres** - Pressão sanguínea diastólica (mm Hg)
4. **skin** - Espessura da dobra da pele do tríceps (mm)
5. **test** - Insulina sérica de 2 horas (mu U/ml)
6. **mass** - Índice de massa corporal (peso em kg/(altura em m)²)
7. **pedi** - Função de linhagem de diabetes (uma função que pontua a probabilidade de diabetes com base no histórico familiar)
8. **age** - Idade (anos)
9. **class** - Variável de classe (0 ou 1)

In []:

```
# Imports
import pandas as pd
import numpy as np
import matplotlib
```

```
import matplotlib.pyplot as plt
import seaborn as sns
import missingno as ms # para tratamento de missings
from matplotlib import cm
from pandas import set_option
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In []:

```
# configuração para não exibir os warnings
import warnings
warnings.filterwarnings("ignore")
```

2. Carga de Dados

Iremos usar o pacote Pandas (Python Data Analysis Library) para carregar de um arquivo .csv sem cabeçalho disponível online.

Com o dataset carregado, iremos explorá-lo um pouco.

In []:

```
# Carrega arquivo csv usando Pandas usando uma URL

# Informa a URL de importação do dataset
url = "https://raw.githubusercontent.com/tatianaesc/datascience/main/diabetes

# Informa o cabeçalho das colunas
colunas = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'cl

# Lê o arquivo utilizando as colunas informadas
dataset = pd.read_csv(url, names=colunas, skiprows=1, delimiter=',')
```

In []:

```
dataset.head()
```

Out[]:

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0

	preg	plas	pres	skin	test	mass	pedi	age	class
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

3. Análise de Dados

3.1. Estatísticas Descritivas

Vamos iniciar examinando as dimensões do dataset, suas informações e alguns exemplos de linhas.

```
In [ ]: # Mostra as dimensões do dataset
print(dataset.shape)
```

(768, 9)

```
In [ ]: # Mostra as informações do dataset
print(dataset.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype  
--- 
 0   preg      768 non-null    int64  
 1   plas      768 non-null    int64  
 2   pres      768 non-null    int64  
 3   skin      768 non-null    int64  
 4   test      768 non-null    int64  
 5   mass      768 non-null    float64 
 6   pedi      768 non-null    float64 
 7   age       768 non-null    int64  
 8   class     768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

```
In [ ]: # Mostra as 10 primeiras Linhas do dataset
dataset.head(10)
```

	preg	plas	pres	skin	test	mass	pedi	age	class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
5	5	116	74	0	0	25.6	0.201	30	0
6	3	78	50	32	88	31.0	0.248	26	1
7	10	115	0	0	0	35.3	0.134	29	0

```
8    2   197   70   45   543   30.5   0.158   53    1
```

```
9    8   125   96    0    0    0.0   0.232   54    1
```

In []:

```
# Mostra as 10 últimas linhas do dataset
dataset.tail(10)
```

Out[]:

	preg	plas	pres	skin	test	mass	pedi	age	class
758	1	106	76	0	0	37.5	0.197	26	0
759	6	190	92	0	0	35.5	0.278	66	1
760	2	88	58	26	16	28.4	0.766	22	0
761	9	170	74	31	0	44.0	0.403	43	1
762	9	89	62	0	0	22.5	0.142	33	0
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

É sempre importante verificar o tipo do atributos do dataset, pois pode ser necessário realizar conversões. Já fizemos anteriormente com o comando info, mas vamos ver uma outra forma de verificar a natureza de cada atributo e então exibir um resumo estatístico do dataset.

In []:

```
# Verifica o tipo de dataset de cada atributo
dataset.dtypes
```

Out[]:

preg	int64
plas	int64
pres	int64
skin	int64
test	int64
mass	float64
pedi	float64
age	int64
class	int64
dtype:	object

In []:

```
# Faz um resumo estatístico do dataset (média, desvio padrão, mínimo, máximo)
dataset.describe()
```

Out[]:

	preg	plas	pres	skin	test	mass	
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	0.471
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078

		ciencia-de-dados-e-analytics/MVP_CD_Diabetes.ipynb at main · dipuciodigital/ciencia-de-dados-e-analytics						
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.24	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.37	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.62	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.42	

Vamos agora verificar se o dataset tem as classes balanceadas para que possamos tratar o desbalanceamento posteriormente, se necessário. Veremos que as classes 0 (não ocorrência de diabetes) e 1 (ocorrência de diabetes) estão desbalanceadas. Vamos guardar esta informação, pois possivelmente precisaremos realizar algum tipo de tratamento nas próximas etapas.

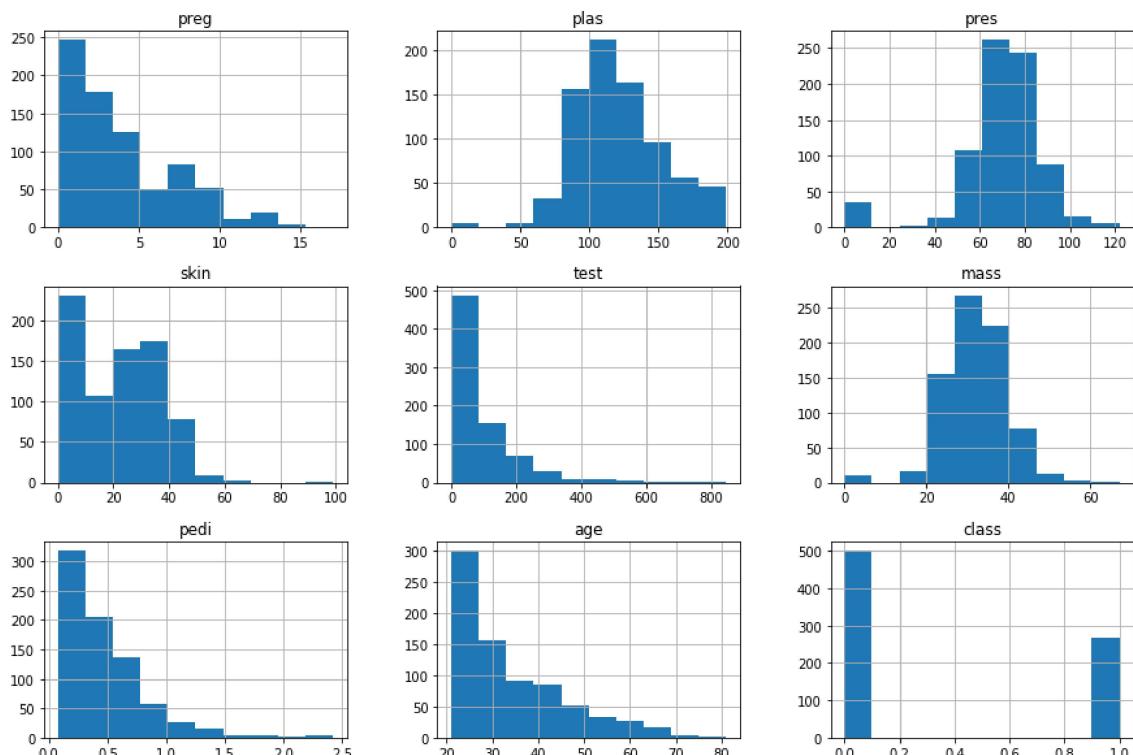
```
In [ ]: # distribuição das classes
print(dataset.groupby('class').size())
```

```
class
0    500
1    268
dtype: int64
```

3.2. Visualizações Unimodais

Vamos criar agora um histograma para cada atributo do dataset. Veremos que os atributos age, pedi e test seguem uma distribuição exponencial, e que as colunas mass e pres seguem uma distribuição aproximadamente normal.

```
In [ ]: # Histograma
dataset.hist(figsize = (15,10))
plt.show()
```



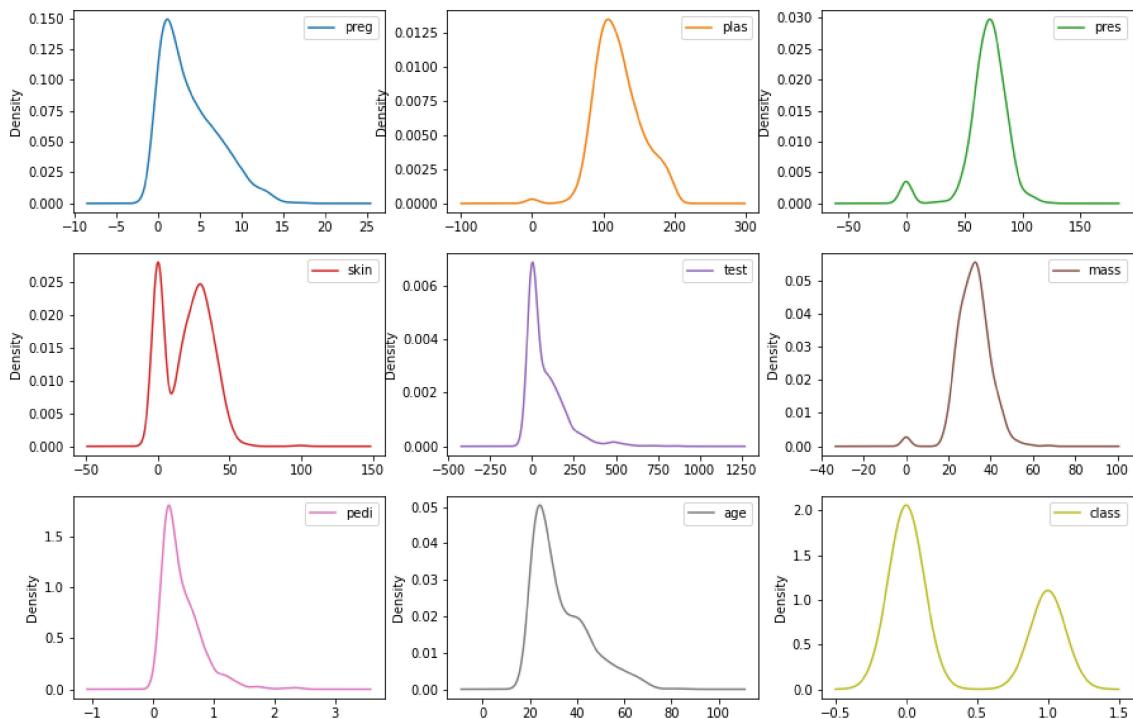
O Gráfico de Densidade, ou Density Plot, é bem parecido com o histograma, mas com uma visualização um pouco diferente. Com ele, pode ser mais fácil identificar a

distribuição do atributos do dataset. Assim como fizemos com o histograma, vamos criar um density plot para cada atributo do dataset.

Veremos que muitos dos atributos têm uma distribuição distorcida. Uma transformação como a Box-Cox, que pode aproximar a distribuição de uma Normal, pode ser útil neste caso.

In []:

```
# Density Plot
dataset.plot(kind = 'density', subplots = True, layout = (3,3), sharex = False
plt.show()
```



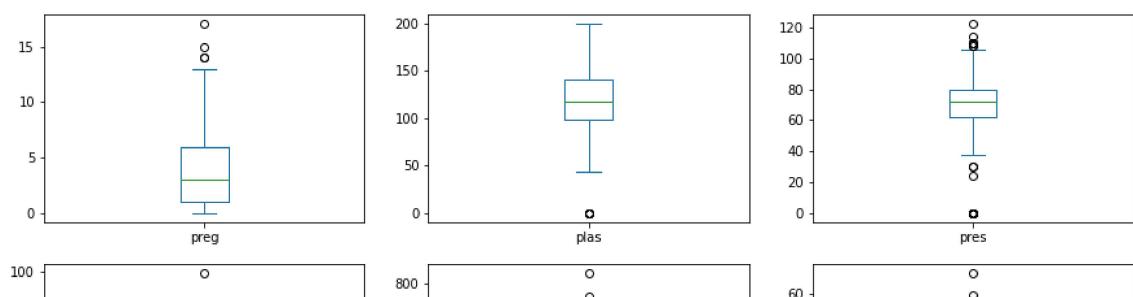
Vamos agora trabalhar com boxplots. No **boxplot**, a linha no centro (vermelha) representa o valor da mediana (segundo quartil ou p50). A linha abaixo é o 1o quartil (p25) e a linha acima o terceiro quartil (p75). O boxplot ajuda a ter uma ideia da dispersão dos dataset e os possíveis outliers.

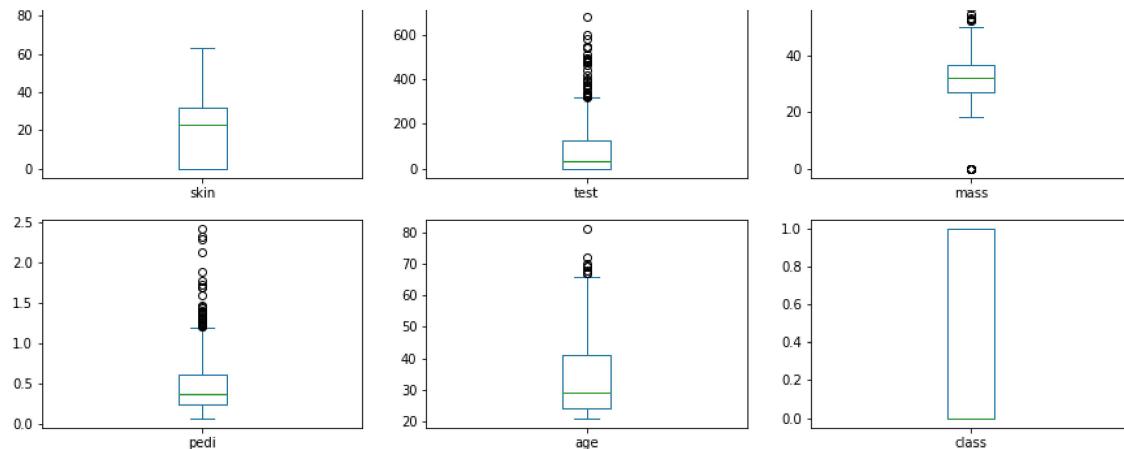
OBS: Se um ponto do dataset é muito distante da média (acima de 3 desvios padrão da média), pode ser considerado outlier.

Nos gráficos bloxplot, veremos que a dispersão dos atributos do dataset é bem diferente.

In []:

```
# Boxplot
dataset.plot(kind = 'box', subplots = True, layout = (3,3), sharex = False, s
plt.show()
```





3.3. Visualizações Multimodais

Ao visualizar as correlações entre os atributos através da matriz de correlação, perceberemos que parece haver alguma estrutura na ordem dos atributos. O azul ao redor da diagonal sugere que os atributos que estão próximos um do outro são geralmente mais correlacionados entre si. Os vermelhos também sugerem alguma correlação negativa moderada, a medida que os atributos

Vamos agora verificar a covariância entre as variáveis numéricas do dataset. A **covariância** representa como duas variáveis numéricas estão relacionadas. Existem várias formas de calcular a correlação entre duas variáveis, como por exemplo, o coeficiente de correlação de Pearson, que pode ser:

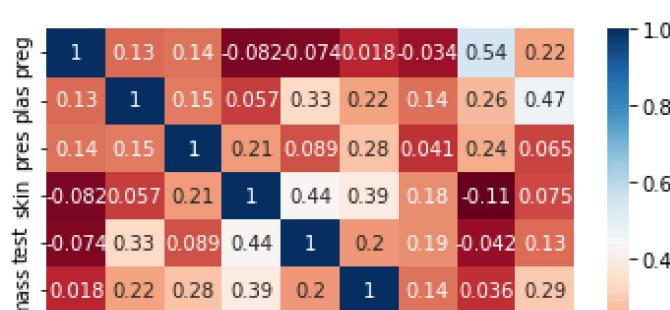
- Próximo de -1 : há uma correlação negativa entre as variáveis,
- Próximo de +1: há uma correlação positiva entre as variáveis.
- 0: não há correlação entre as variáveis.

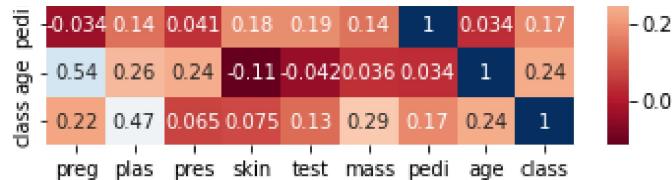
OBS: Esta informação é relevante porque alguns algoritmos como regressão linear e regressão logística podem apresentar problemas de performance se houver atributos altamente correlacionados. Vale a pena consultar a documentação do algoritmo para verificar se algum tipo de tratamento de dataset é necessário.

Falamos anteriormente da importância da correlação entre os atributos, e agora iremos visualizar esta informação em formato gráfico. A **matriz de correlação** exibe graficamente a correlação entre os atributos numéricos do dataset.

O código a seguir exibe a matriz de correlação.

```
In [ ]: # Matriz de Correlação com Matplotlib Seaborn
sns.heatmap(dataset.corr(), annot=True, cmap='RdBu');
```

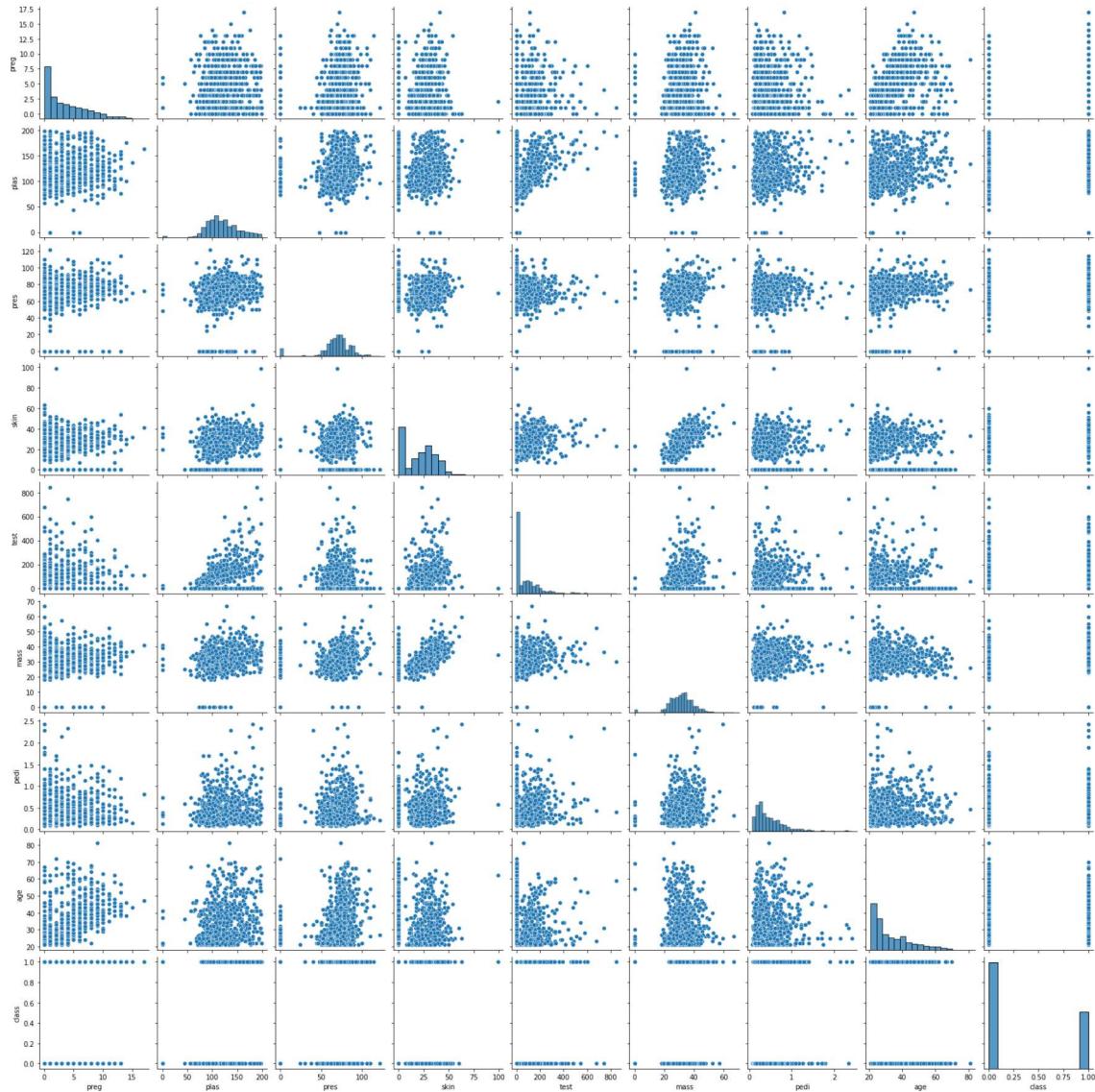




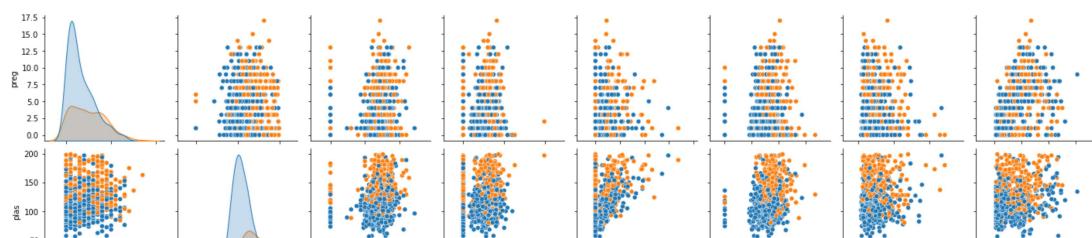
Por sua vez, o gráfico de dispersão (**scatter plot**) mostra o relacionamento entre duas variáveis. Vamos exibir um para cada par de atributos dos dataset, usando o Seaborn.

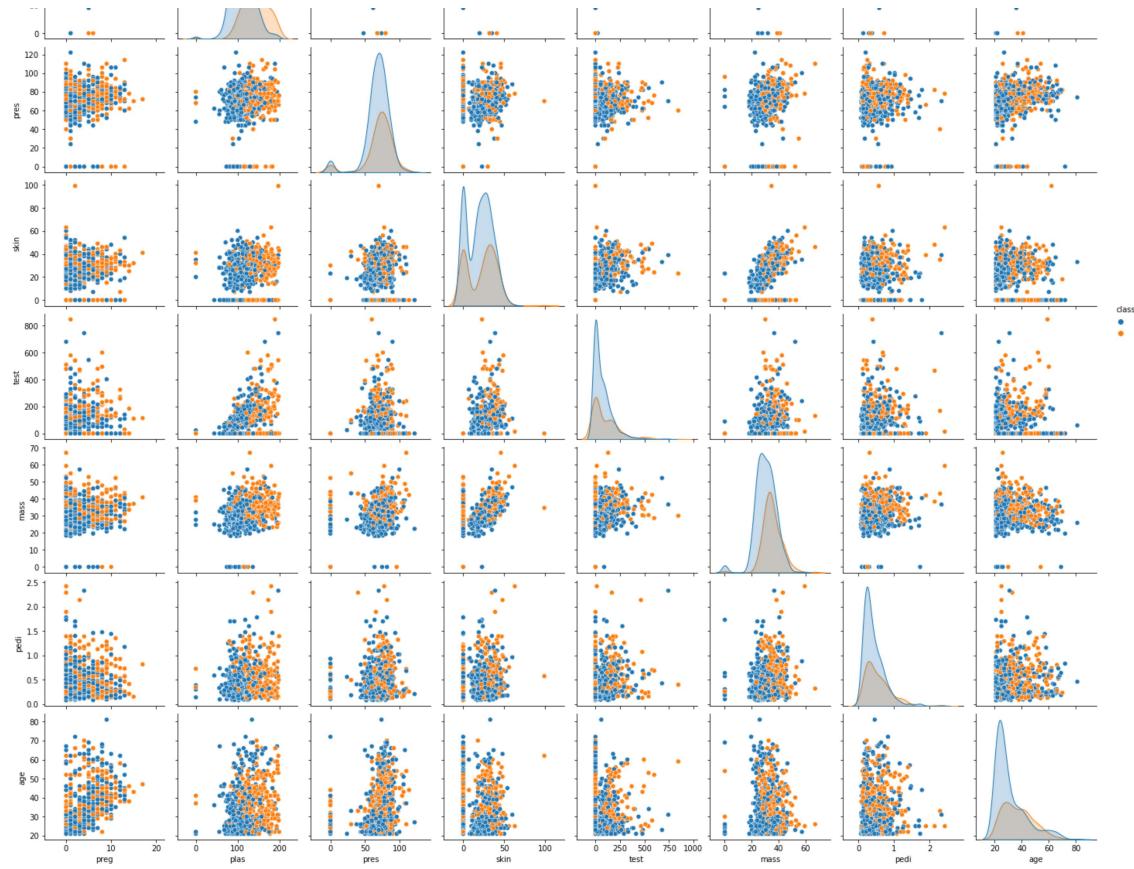
```
In [ ]: # Scatter Plot com Seaborn - Variação 1
sns.pairplot(dataset)
```

Out[]: <seaborn.axisgrid.PairGrid at 0x7f54e1724f10>



```
In [ ]: # Scatter Plot com Seaborn - Variação 2
sns.pairplot(dataset, hue = "class", height = 2.5);
```





4. Pré-Processamento de dados

Nesta etapa, poderíamos realizar diversas operações de preparação de dados, como por exemplo, tratamento de valores missings (faltantes), limpeza de dados, transformações como one-hot-encoding, seleção de características (feature selection), entre outras não mostradas neste notebook. Lembre-se de não criar uma versão padronizada/normalizada dos dados neste momento (apesar de serem operações de pré-processamento) para evitar o Data Leakage.

4.1. Tratamento de Missings e Limpeza

Sabemos que o dataset Diabetes não tem missings aparentes, mas valores "0" que parecem ser missings. Vamos então fazer este tratamento e criar uma nova visão do nosso dataset.

```
In [ ]: # verificando nulls no dataset
dataset.isnull().sum()
```

```
Out[ ]: preg      0
plas      0
pres      0
skin      0
test      0
mass      0
pedi      0
age       0
class     0
dtype: int64
```

```
In [ ]: # salvando um NOVO dataset para tratamento de missings (cuidado para não sobr
```

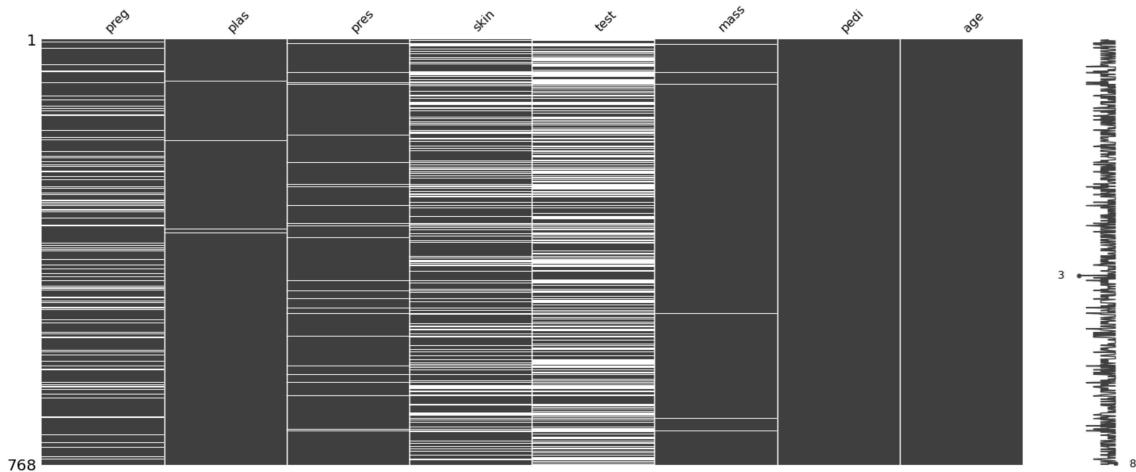
```
# recuperando os nomes das colunas
col = list(dataset.columns)

# o novo dataset irá conter todas as colunas com exceção da última (classe)
atributos = dataset[col[0:-1]]

# substituindo os zeros por NaN
atributos.replace(0, np.nan, inplace=True)

# exibindo visualização matricial da nulidade do dataset
ms.matrix(atributos)
```

Out[]: <Axes: >

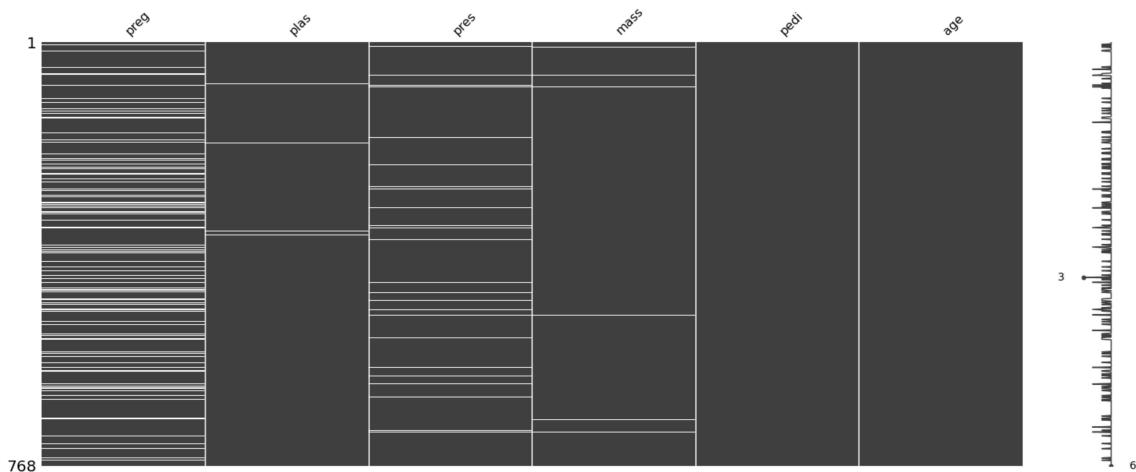


In []:

```
# removendo as colunas 'skin' e 'test'
atributos.drop(['skin', 'test'], axis=1, inplace=True)

# exibindo visualização matricial da nulidade do dataset
ms.matrix(atributos)
```

Out[]: <Axes: >



In []:

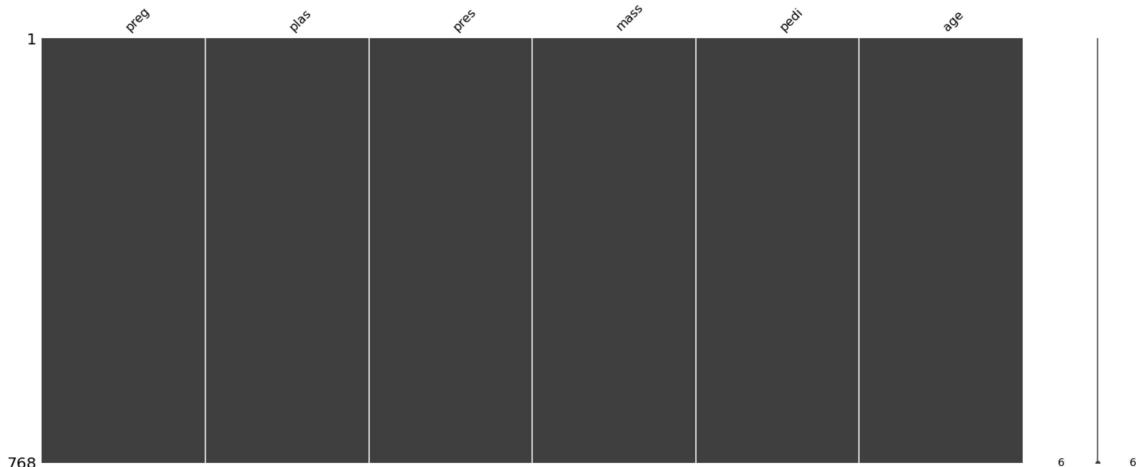
```
# substituindo os NaN de 'preg' por 0
atributos['preg'].fillna(0, inplace=True)

# substituindo os NaN de 'plas', 'pres' e 'mass' pela mediana da coluna
atributos['plas'].fillna(atributos['plas'].median(), inplace=True)
atributos['pres'].fillna(atributos['pres'].median(), inplace=True)
atributos['mass'].fillna(atributos['mass'].median(), inplace=True)

# exibindo visualização matricial da nulidade do dataset
```

```
ms.matrix(atributos)
```

```
Out[ ]: <Axes: >
```



```
In [ ]:
```

```
# Guardando o novo dataset para testes futuros
datasetSemMissings = atributos

# incluindo a coluna 'class' no novo dataset
datasetSemMissings['class'] = dataset['class']

# exibindo as primeiras Linhas
datasetSemMissings.head()
```

```
Out[ ]:
```

	preg	plas	pres	mass	pedi	age	class
0	6.0	148.0	72.0	33.6	0.627	50	1
1	1.0	85.0	66.0	26.6	0.351	31	0
2	8.0	183.0	64.0	23.3	0.672	32	1
3	1.0	89.0	66.0	28.1	0.167	21	0
4	0.0	137.0	40.0	43.1	2.288	33	1

4.2. Separação em conjunto de treino e conjunto de teste

É uma boa prática usar um conjunto de teste (na literatura também chamado de conjunto de validação), uma amostra dos dados que não será usada para a construção do modelo, mas somente no fim do projeto para confirmar a precisão do modelo final. É um teste que podemos usar para verificar o quanto boa foi a construção do modelo, e para nos dar uma ideia de como o modelo irá performar nas estimativas em dados não vistos. Usaremos 80% do conjunto de dados para modelagem e guardaremos 20% para teste, usando a estratégia train-test-split, já explicada anteriormente. Primeiramente, iremos sinalizar quais são as colunas de atributos (X - 0 a 7) e qual é a coluna das classes (Y - 8). Em seguida, especificaremos o tamanho do conjunto de teste desejado e uma semente (para garantir a reproduzibilidade dos resultados). Finalmente, faremos a separação dos conjuntos de treino e teste através do comando `train_test_split`, que retornará 4 estruturas de dados: os atributos e classes para o conjunto de teste e os atributos e classes para o conjunto de treino.

```
In [ ]:
```

```
+text ciço - a 7a
```

```
seed = 7
```

```
# Separação em conjuntos de treino e teste (dataset original)
array = dataset.values
X = array[:,0:8]
y = array[:,8]
#X_train, X_test, y_train, y_test = train_test_split(X, y,
#    test_size=test_size, shuffle=True, random_state=seed) # sem estratificação
X_train, X_test, y_train, y_test = train_test_split(X, y,
    test_size=test_size, shuffle=True, random_state=seed, stratify=y) # com e
```

In []:

```
# Separação em conjuntos de treino e teste (dataset sem missings - 2 colunas)
array = datasetSemMissings.values
X_sm = array[:,0:6]
y_sm = array[:,6]
#X_train_sm, X_test_sm, y_train_sm, y_test_sm = train_test_split(X_sm, y_sm,
#    test_size=test_size, shuffle=True, random_state=seed) # sem estratificação
X_train_sm, X_test_sm, y_train_sm, y_test_sm = train_test_split(X_sm, y_sm,
    test_size=test_size, shuffle=True, random_state=seed, stratify=y_sm) # co
```

