

# Criando Ambiente Virtual

Primeiramente, configurar rapidamente o ambiente de ciência de dados e usar diferentes bibliotecas python que podem acelerar o processo de aprendizagem.

O Machine Learning parece ser fascinante para muitos iniciantes, mas muitas vezes eles se perdem no pool de informações disponíveis em diferentes recursos. Isso é verdade que temos muitos algoritmos e passos diferentes para aprender, mas começando com uma base forte não só dá confiança, mas também a motivação para aprender e explorar mais. Aqui, passaremos pelas etapas sobre como configurar seu ambiente e começar a aprender com a ajuda de um conjunto de dados conhecido, ou seja, conjunto de dados **IRIS**, que é um problema de **classificação multiclasse** no Machine Learning. Também passaremos por algumas **bibliotecas de python úteis que podem acelerar o processo de aprendizagem que pode ajudá-lo mesmo se você for um cientista de dados.**

## Configuração do Meio Ambiente

Usaremos a distribuição da Anaconda para configurar o ambiente de ciência de dados. Baixe a versão mais recente de Anaconda [daqui](#) e abra o **prompt anaconda** e execute o seguinte comando:


```
jupyter notebook
```

O comando acima iniciará o servidor Jupyter e carregará o diretório do notebook no seu navegador.

## Criar um ambiente virtual

Managing environments - conda 4.9.2.post14+2fcfec1a documentation

With conda, you can create, export, list, remove, and update environments that have different versions of Python and/or packages installed in them. Switching or moving between environments is called activating the environment. You can also

 <https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html>

Espero que esteja ciente dos ambientes virtuais, se não puder ler sobre eles [aqui](#). Embora a Anaconda venha com um **ambiente básico** que está tendo a maioria das bibliotecas já instaladas, **recomenda-se usar ambientes virtuais**, pois eles nos ajudam a gerenciar diferentes ambientes que podem ter pacotes diferentes e se algo der errado com um ambiente não afetará outros. Aqui estão os comandos que você pode usar para criar, ativar e instalar pacotes no ambiente virtual.

```
# create new environment
conda create --name your_env_name

# activate the environment
conda activate your_env_name

# installing package
conda install -c conda-forge package_name
```

## Link ambiente virtual com o Notebook

Por padrão, o **novo ambiente não apareceria no notebook Jupyter**. Precisariamos executar seguintes comandos para vincular o ambiente com o cliente Jupyter:

```
# install pip if it is not installed in the environment
conda install -c conda-forge pip

# install ipykernel
pip install --user ipykernel

# link with notebook
python -m ipykernel install --user --name=your_env_name
```



olhada no meu notebook, sintá-se livre para baixá-lo e importá-lo em seu ambiente e brincar com ele-

```
conda install -c conda-forge jupyter_contrib_nbextensions
```

## Jupyter Contrib Nbextensions

Muitas vezes precisamos compartilhar nosso caderno com diferentes stakeholders ou talvez precise apresentá-los, esta biblioteca nos fornece muitas extensões diferentes. Eu não vou passar pelas extensões aqui, mas eu recomendo usar isso. Meus favoritos são...

1. Títulos dobráveis.
2. Tabela de Conteúdo.
3. Tempo de execução.

Podemos instalá-lo usando:

```
conda install -c conda-forge jupyter_contrib_nbextensions
```

## Pandas- Python Data Analysis Library

Este é o coração da ciência de dados com python e fornece muitas capacidades diferentes como

- Estruturas de dados para trabalhar com os dados.
- Operações que você pode executar nos dados.
- Carregar e salvar dados em diferentes formatos.

e muitos mais. Muitas outras bibliotecas que usamos para aprendizado de máquina com python têm pandas como dependência. Instale-o usando...

```
conda install -c conda-forge pandas
```

O comando acima instalará outras bibliotecas como o NumPy que os pandas usam sob o capô.

## Sklearn (Scikit-Learn)

Usaremos esta biblioteca para baixar conjuntos de dados de teste e aplicar diferentes algoritmos de aprendizado de máquina. Instale usando o seguinte comando.

```
conda install -c conda-forge scikit-learn
```

Em problemas **de classificação** de aprendizado de máquina, o problema pode ser entendido como para recursos  $X$  (variáveis de entrada) prever  $y$  (valor-alvo). Sklearn fornece poucos conjuntos de dados de teste que podemos usar para jogar, levaremos o conjunto de dados IRIS para este exercício, mas se você quiser brincar com os outros, então você pode se referir a [isso](#).

Scikit-learn 0.23.1 adicionou um recurso pelo qual podemos retornar o conjunto de dados do teste diretamente no dataframe  $X$  e  $y$ . Certifique-se de que está executando a versão 0.23.1.

```
from sklearn.datasets import load_iris
```

```
In [2]: X, y = load_iris(as_frame=True, return_X_y=True)
```

```
In [3]: X.head(2)
```

```
Out[3]:
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

```
In [4]: y.head(2)
```

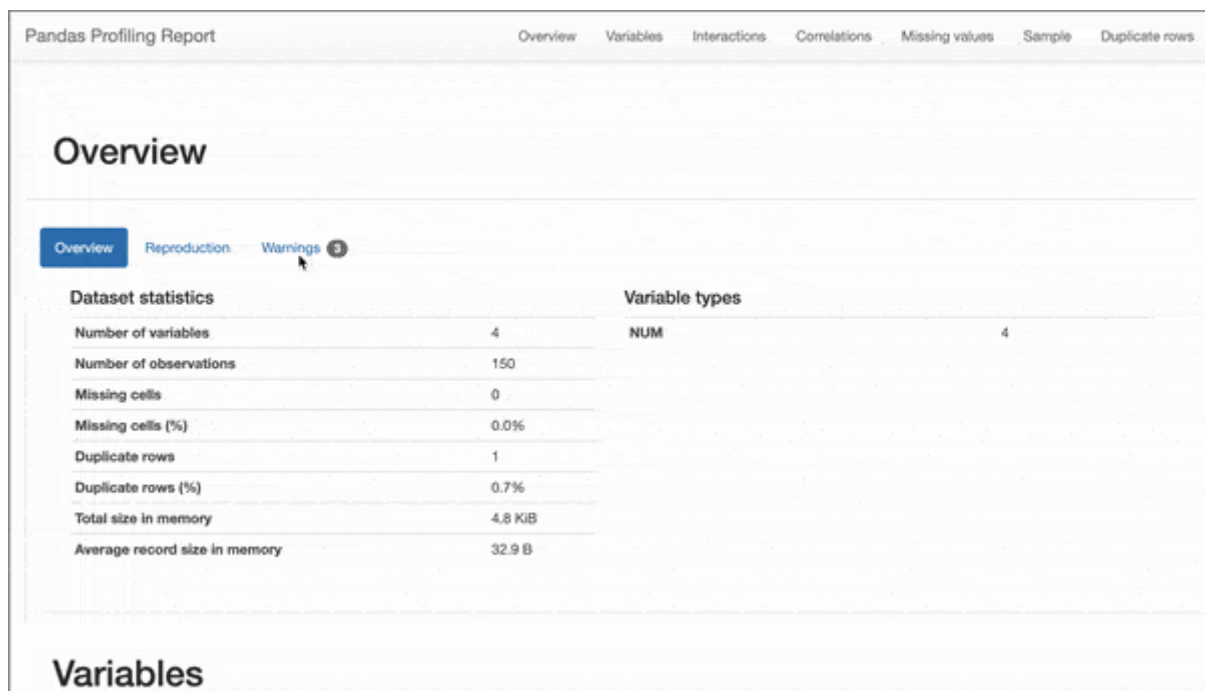
```
Out[4]: 0    0  
        1    0  
        Name: target, dtype: int64
```

Agora vamos passar pelas outras bibliotecas e vamos usar Sklearn para modelar mais tarde

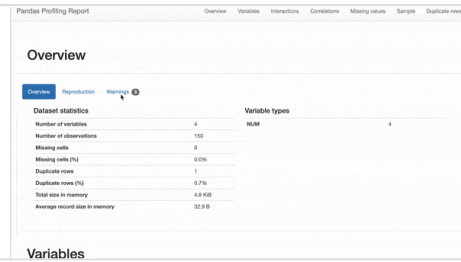
## pandas-profiling

Ele fornece um rico relatório de perfil para os dados que fornece um monte de informações de valores perdidos a correlações. Você precisa **instalá-lo usando pip** como conda-package baixa versão antiga dele.

```
pip install --user pandas-profiling
```



[https://miro.medium.com/max/750/1\\*knV7VkJboi-Vsg0OgQGe8A.gif](https://miro.medium.com/max/750/1*knV7VkJboi-Vsg0OgQGe8A.gif)



Dataset statistics		Variable types
Number of variables	4	NUM 4
Number of observations	152	
Missing cells	9	
Missing cells (%)	0.0%	
Duplicate rows	1	
Duplicate rows (%)	0.7%	
Total size in memory	4.2 KiB	
Average record size in memory	32.9 B	

Este relatório fornece muitos detalhes, dos quais poucos são...

1. Visão geral de diferentes variáveis no conjunto de dados.
2. Correlação entre variáveis.
3. Interações entre variáveis.
4. Detalhes sobre cada variável.

Os seguintes comandos podem ser usados para gerar e salvar o relatório de perfil:

```
# import ProfileReport
from pandas_profiling import ProfileReport

# generate the profile report and save it to a variable
profile_report = ProfileReport(your_dataframe)

# save to html format
profile_report.to_file('pandas_profiling_report.html')

# load as widget in the notebook, you might need to install pywidgets
profile_report.to_widgets()
```

## Plotly

Embora o perfil de pandas forneça muitas informações úteis, ainda precisamos visualizar diferentes informações, como por exemplo, precisamos descobrir como a variável alvo é distribuída entre múltiplas variáveis de entrada. Existem muitas bibliotecas para visualização, Matplotlib e Seaborn são as famosas que você teria ouvido falar. A principal coisa em que plotly se destaca são os **enredos**

**interativos**, ou seja, você pode interagir com as parcelas geradas. Instale-o usando o seguinte comando:

```
conda install -c conda-forge plotly
```

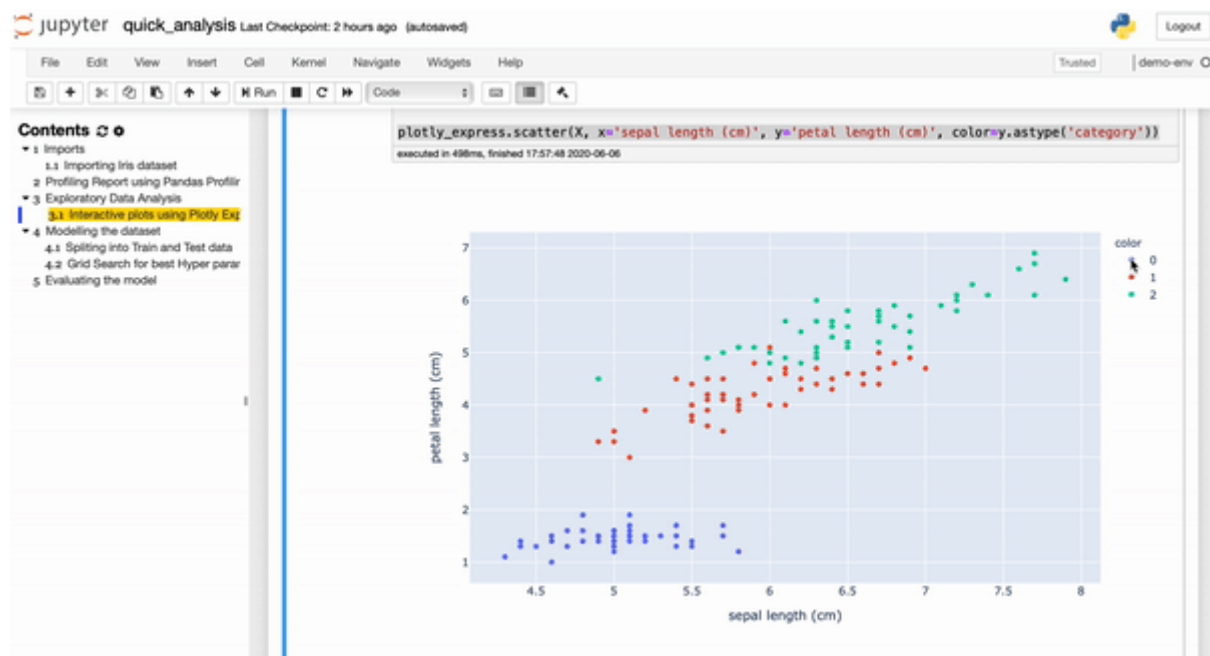
Abaixo, traçamos um gráfico de dispersão entre o comprimento do sepala com comprimento de pétala e usamos 'cor' para mostrar como a variável alvo está relacionada.

```
# import plotly express
import plotly.express as plotly_express

# plot between sepal length and petal length with target
plotly_express.scatter(X, x='sepal length (cm)', y='petal length (cm)', \
                      color=y.astype('category'))

# plot between sepal width and petal width with target
plotly_express.scatter(X, x='sepal width (cm)', y='petal width (cm)', \
                      color=y.astype('category'))
```

Podemos filtrar diferentes alvos, vide abaixo:





Esta biblioteca fornece um monte de funcionalidade adicional, talvez possamos cobrir isso em uma história diferente.

## Conjunto de dados de treinamento e teste

A ideia de gerar modelos é prever valores que não são conhecidos. Se aprendermos o modelo em todo o conjunto de dados, então não poderemos avaliar como ele se sai nos dados invisíveis. Para isso, dividimos o conjunto de dados em conjunto de dados de treinamento e teste. Um conjunto de dados de treinamento é usado para treinar o modelo e o conjunto de testes é usado para avaliar o modelo. Sklearn fornece uma função 'train\_test\_split' que divide o conjunto de dados em conjuntos de dados de trem e teste. O código a seguir pode ser usado para dividir os conjuntos de dados.

```
from sklearn.model_selection import train_test_split
```

### Splitting into Train and Test data

```
In [10]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=0)
```

```
In [11]: print('train shape is ', X_train.shape, y_train.shape)
          print('test shape is ', X_test.shape, y_test.shape)
```

```
train shape is  (120, 4) (120,)
test shape is   (30, 4) (30,)
```

## Sintonizando hiperparâmetros

Uma das tarefas importantes no aprendizado de máquina é sintonizar hiperparâmetros, esses parâmetros são os diferentes atributos do algoritmo que controlam o processo de aprendizagem. Diferentes valores são adequados para diferentes problemas de aprendizagem e é importante descobrir os melhores parâmetros. O Sklearn fornece principalmente duas maneiras de 'GridSearchCV' e 'RandomizedSearchCV' para encontrar os melhores parâmetros. Para grandes conjuntos de treinamento, talvez precisemos usar randomizedSearchCV, pois levará muito tempo para aprender todos os parâmetros. No conjunto de dados IRIS, temos apenas 150 linhas e, portanto, usamos 'GridSearchCV'.

Para esta história, vamos treinar o modelo de Região Logística que é adequado para problemas de classificação e tem diferentes hiperparâmetros como 'solver', 'C', 'penalty' e 'l1-ratio'. Nem todos os solucionadores suportam todos os parâmetros e, portanto, criamos dicionários diferentes para todos os solucionadores diferentes.

```
# import LogisticRegression
from sklearn.linear_model import LogisticRegression

# import GridSearchCV
from sklearn.model_selection import GridSearchCV

# create a LogisticRegression instance
logistics_regression = LogisticRegression(random_state=0, max_iter=10000)

# define parameters grid
parameters = [
    {
        'penalty': ['l1'],
        'solver': [ 'liblinear', 'saga'],
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    },
    {
        'penalty': ['l2'],
        'solver': ['newton-cg', 'lbfgs', 'sag', 'saga'],
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
    },
    {
        'penalty': ['elasticnet'],
        'solver': ['saga'],
        'C': [0.001, 0.01, 0.1, 1, 10, 100, 1000],
        'l1_ratio': [.1, .5, .7, .9, .95, .99, 1]
    }
]

# Define GridSearchCV
grid_search = GridSearchCV(logistics_regression, param_grid = parameters)

# fit GridSearchCV
model = grid_search.fit(X_train, y_train)

# find the best estimator
print (model.best_estimator_)

# predict y values
y_pred = model.predict(X_test)
```

O código acima procuraria diferentes combinações de parâmetros e encontraria o melhor que melhor generaliza o problema.

## Avaliando o modelo

Como mencionamos, precisamos avaliar o modelo no conjunto de dados do teste, muitas métricas diferentes estão disponíveis. A comum é a precisão para problemas de classificação. Aqui mostraremos a precisão, `classification_report` e a matriz de confusão que Sklearn fornece.

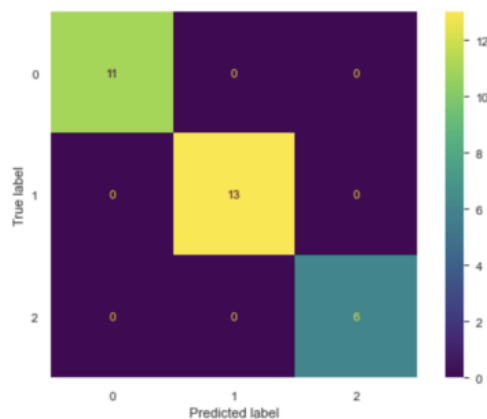
### Evaluating the model

```
In [18]: print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	11
1	1.00	1.00	1.00	13
2	1.00	1.00	1.00	6
accuracy			1.00	30
macro avg	1.00	1.00	1.00	30
weighted avg	1.00	1.00	1.00	30

```
In [19]: plot_confusion_matrix(model, X_test, y_test)
```

```
Out[19]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1400231c0>
```



```
In [20]: model.score(X_test, y_test)
```

```
Out[20]: 1.0
```

O conjunto de dados IRIS é classificado como um conjunto de dados fácil, o que significa que os dados já são adequados para fins de aprendizado de máquina e, portanto, conseguimos obter uma pontuação perfeita, ou seja, pontuação de precisão de 1.0 com o nosso modelo. Isso significa que nosso modelo previu todas as amostras no conjunto de dados do teste corretamente. Vai variar com os diferentes problemas que você está tentando resolver.