

# Data Science Academy - Python Fundamentos - Capítulo 8

**Download:** <http://github.com/dsacademybr>  
<http://github.com/dsacademybr>

## Exercícios Sobre Módulos Python Para Análise de Dados

### \*\* ATENÇÃO \*\*

Alguns dos exercícios podem requerer pesquisa adicional na documentação dos pacotes. Pesquise!

### Exercício 1

In [1]:

```
# Crie um array NumPy com 1000000 e uma lista com 1000000.
# Multiplique cada elemento do array e da lista por 2 e calcule o tempo de execução com cada
# Qual objeto oferece melhor performance, array NumPy ou lista?
import numpy as np
my_arr = np.arange(1000000)
my_list = list(range(1000000))

%time for _ in range(10): my_arr2 = my_arr * 2
%time for _ in range(10): my_list2 = [x * 2 for x in my_list]

CPU times: user 13.5 ms, sys: 4.26 ms, total: 17.8 ms
Wall time: 17.9 ms
CPU times: user 587 ms, sys: 169 ms, total: 756 ms
Wall time: 757 ms
```

### Exercício 2

In [2]:

```
# Exercício 2
# Crie um array de 10 elementos
# Altere os valores de todos os elementos dos índices 5 a 8 para 0
import numpy as np
arr = np.arange(10)
arr
```

Out[2]:

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

In [3]:

```
arr[5:8] = 0
arr
```

Out[3]:

```
array([0, 1, 2, 3, 4, 0, 0, 0, 8, 9])
```

## Exercício 3

In [4]:

```
# Crie um array de 3 dimensões e imprima a dimensão 1
import numpy as np
arr3d = np.array([[1, 2, 3], [4, 5, 6], [[7, 8, 9], [10, 11, 12]]])
arr3d
```

Out[4]:

```
array([[[ 1,  2,  3],
       [ 4,  5,  6]],
      [[ 7,  8,  9],
       [10, 11, 12]])
```

In [5]:

```
arr3d[0]
```

Out[5]:

```
array([[1, 2, 3],
       [4, 5, 6]])
```

## Exercício 4

In [6]:

```
# Crie um array de duas dimensões (matriz).
# Imprima os elementos da terceira Linha da matriz
# Imprima todos os elementos da primeira e segunda linhas e segunda e terceira colunas
arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
arr2d
```

Out[6]:

```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[7]:

```
# Imprima os elementos da terceira linha da matriz
arr2d[2]
```

Out[7]:

```
array([7, 8, 9])
```

In [8]:

```
# Imprima todos os elementos da primeira e segunda linhas e segunda e terceira colunas
arr2d[:2, 1:]
```

Out[8]:

```
array([[2, 3],
       [5, 6]])
```

## Exercício 5

In [9]:

```
# Calcule a transposta da matriz abaixo
arr = np.arange(15).reshape((3, 5))
arr
```

Out[9]:

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

In [10]:

```
arr.T
```

Out[10]:

```
array([[ 0,  5, 10],
       [ 1,  6, 11],
       [ 2,  7, 12],
       [ 3,  8, 13],
       [ 4,  9, 14]])
```

## Exercício 6

In [11]:

```
# Considere os 3 arrays abaixo
# Retorne o valor do array xarr se o valor for True no array cond. Caso contrário, retorne
xarr = np.array([1.1, 1.2, 1.3, 1.4, 1.5])
yarr = np.array([2.1, 2.2, 2.3, 2.4, 2.5])
cond = np.array([True, False, True, True, False])
```

[12]:

```
for x, y, c in zip(xarr, yarr, cond):
    print(x, y, c)
```

1.1 2.1 True  
 1.2 2.2 False  
 1.3 2.3 True  
 1.4 2.4 True  
 1.5 2.5 False

In [13]:

```
resultado = [(x if c else y) for x, y, c in zip(xarr, yarr, cond)]
resultado
```

Out[13]:

[1.1, 2.2, 1.3, 1.4, 2.5]

## Exercício 7

In [14]:

```
# Crie um array A com 10 elementos e salve o array em disco com a extensão npy
# Depois carregue o array do disco no array B
A = np.arange(10)
print(A)
np.save('array_a', A)
```

[0 1 2 3 4 5 6 7 8 9]

In [15]:

```
B = np.load('array_a.npy')
print(B)
```

[0 1 2 3 4 5 6 7 8 9]

## Exercício 8

In [16]:

```
# Considerando a série abaixo, imprima os valores únicos na série
import pandas as pd
obj = pd.Series(['c', 'a', 'd', 'a', 'a', 'b', 'b', 'c', 'c', 'a', 'b'])
```

[17]:

```
uniques = obj.unique()
uniques
```

Out[17]:

```
array(['c', 'a', 'd', 'b'], dtype=object)
```

## Exercício 9

In [18]:

```
# Considerando o trecho de código que conecta em uma url na internet, imprima o dataframe com as issues
import requests
url = 'https://api.github.com/repos/pandas-dev/pandas/issues'
resp = requests.get(url)
resp
```

Out[18]:

```
<Response [200]>
```

In [19]:

```
data = resp.json()
data[0]['title']
```

Out[19]:

```
'Slicing columns with mixed types <str>,<int> fails with ValueError'
```

[20]:

```
issues = pd.DataFrame(data, columns=['number', 'title', 'labels', 'state'])
issues
```

Out[20]:

	number	title	labels	state
0	20975	Slicing columns with mixed types <str>,<int> f...	[]	open
1	20974	BUG: Fix drop_duplicates failure when DataFram...	[{"id": 76811, "url": 'https://api.github.com/...']	open
2	20970	performance when "combining" categories	[{"id": 78527356, "url": 'https://api.github.c...']	open
3	20969	Getting a ... in my CSV when using to_csv()	[]	open
4	20968	Sharey keyword for boxplot	[]	open
5	20967	Column can be extended beyond DataFrame's length	[]	open
6	20966	BUG: Fix wrong khash method definition	[{"id": 76811, "url": 'https://api.github.com/...']	open
7	20965	BUG: Fix combine_first converts other columns ...	[{"id": 76811, "url": 'https://api.github.com/...']	open
8	20964	BUG: Index with integer data and datetime64[ns...	[]	open
9	20962	df.rank() has different behavior for python2 a...	[]	open
10	20960	DOC: add reshaping visuals to the docs (Reshap...	[{"id": 134699, "url": 'https://api.github.com/...']	open
11	20958	Using apply on a grouper works only if done af...	[{"id": 76811, "url": 'https://api.github.com/...']	open
12	20956	ENH: Return DatetimeIndex or TimedeltaIndex bi...	[{"id": 49597148, "url": 'https://api.github.c...']	open
13	20955	Building documentation fails with 'ImportError...	[]	open
14	20954	Correlation inconsistencies between Series and...	[]	open
15	20951	Addressing multiindex raises TypeError if indi...	[]	open
16	20948	Indexing DataFrame with DateOffset is nearly i...	[]	open
17	20947	Allow drop bins when using the cut function	[{"id": 76812, "url": 'https://api.github.com/...']	open
18	20945	Data is mismatched with labels after stack wit...	[]	open
19	20944	Drop rows based on condition	[]	open
20	20943	pd.read_sql does not handle queries that retur...	[]	open
21	20940	unclear error with read_sas	[]	open
22	20937	BUG: Series(DT1/TDI) loses the frequency infor...	[]	open

number		title	labels	state
23	20936	__new__() missing 1 required positional argument		open
24	20932	API: expose 'axis' keyword in pandas.core.algo...	[{"id": 35818298, "url": "https://api.github.c..."}]	open
25	20930	DOC: Index.get_loc Cannot Accept List-like Tols		open
26	20928	Added script to fetch wheels [ci skip]	[{"id": 131473665, "url": "https://api.github...."}]	open
27	20927	read_sas OverflowError: int too big to convert...		open
28	20926	BLD: Lint for invalid files in wheels / packages	[{"id": 129350, "url": "https://api.github.com..."}]	open
29	20925	Series.reset_index(level_name, drop=True) accept...	[{"id": 42670965, "url": "https://api.github.c..."}]	open

## Exercício 10

In [21]:

```
# Crie um banco de dados no SQLite, crie uma tabela, insira registros,
# consulte a tabela e retorne os dados em dataframe do Pandas
import sqlite3
import pandas as pd
query = """
CREATE TABLE TESTE
(Cidade VARCHAR(20),
Estado VARCHAR(20),
taxa REAL,
'Impostos' INTEGER
);"""
con = sqlite3.connect('dsa.db')
con.execute(query)
con.commit()
```

In [22]:

```
data = [('Natal', 'Rio Grande do Norte', 1.25, 6),
        ('Recife', 'Pernambuco', 2.6, 3),
        ('Londrina', 'Paraná', 1.7, 5)]
stmt = "INSERT INTO TESTE VALUES(?, ?, ?, ?)"
con.executemany(stmt, data)
con.commit()
```

[23]:

```
cursor = con.execute('select * from teste')
rows = cursor.fetchall()
rows
```

Out[23]:

```
[('Natal', 'Rio Grande do Norte', 1.25, 6),
 ('Recife', 'Pernambuco', 2.6, 3),
 ('Londrina', 'Paraná', 1.7, 5)]
```

In [24]:

```
cursor.description
pd.DataFrame(rows, columns=[x[0] for x in cursor.description])
```

Out[24]:

	Cidade	Estado	taxa	Impostos
0	Natal	Rio Grande do Norte	1.25	6
1	Recife	Pernambuco	2.60	3
2	Londrina	Paraná	1.70	5

**Fim**

Obrigado - Data Science Academy - [facebook.com/dsacademybr](http://facebook.com/dsacademybr)  
[\(http://facebook.com/dsacademybr\)](http://facebook.com/dsacademybr)

# Data Science Academy - Python Fundamentos - Capítulo 9

Download: <http://github.com/dsacademybr>  
[\(http://github.com/dsacademybr\)](http://github.com/dsacademybr)

## Análise Exploratória de Dados

Neste notebook usaremos uma pesquisa recente nos EUA sobre o mercado de trabalho para programadores de software. Nosso objetivo é fazer uma investigação inicial dos dados a fim de detectar problemas com os dados, necessidade de mais variáveis, falhas na organização e necessidades de transformação.

Pesquisa Salarial realizada pelo site <https://www.freecodecamp.com/> (<https://www.freecodecamp.com/>) com programadores de software nos EUA que frequentaram treinamentos Bootcamp.

In [1]:

```
# Importando os pacotes
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import colorsys
plt.style.use('seaborn-talk')
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline
```

In [2]:

```
# Carregando o dataset
df = pd.read_csv("Dados-Pesquisa.csv", sep = ',', low_memory=False)
```

[3]:

`print(df.head())` transposed troco linha por coluna

```

Age AttendedBootcamp BootcampFinish BootcampFullJobAfter \
0 28.0             0.0      NaN           NaN
1 22.0             0.0      NaN           NaN
2 19.0             0.0      NaN           NaN
3 26.0             0.0      NaN           NaN
4 20.0             0.0      NaN           NaN

BootcampLoanYesNo BootcampMonthsAgo BootcampName BootcampPostSalary \
0          NaN            NaN        NaN           NaN
1          NaN            NaN        NaN           NaN
2          NaN            NaN        NaN           NaN
3          NaN            NaN        NaN           NaN
4          NaN            NaN        NaN           NaN

BootcampRecommend ChildrenNumber ... ResourceSoloLearn \
0          NaN            NaN     ...           NaN
1          NaN            NaN     ...           NaN
2          NaN            NaN     ...           NaN
3          NaN            NaN     ...           NaN
4          NaN            NaN     ...           NaN

ResourceStackOverflow ResourceTreehouse ResourceUdacity ResourceUdemy \
\ 0          NaN            NaN       NaN           NaN
1          NaN            NaN       NaN           1.0
2          NaN            NaN       NaN           NaN
3          NaN            NaN       NaN           NaN
4          NaN            NaN       NaN           NaN

ResourceW3Schools ResourceYouTube \
0          NaN            NaN
1          NaN            NaN
2          NaN            NaN
3          NaN            NaN
4          NaN            NaN

                               SchoolDegree           SchoolMajor \
0   some college credit, no degree           NaN
1   some college credit, no degree           NaN
2 high school diploma or equivalent (GED)           NaN
3           bachelor's degree Cinematography And Film
4   some college credit, no degree           NaN

StudentDebtOwe
0    20000.0
1      NaN
2      NaN
3    7000.0
4      NaN

```

rows x 113 columns]

In [4]:

df

Out[4]:

	Age	AttendedBootcamp	BootcampFinish	BootcampFullJobAfter	BootcampLoanYesNo	BootcampMor
0	28.0	0.0	NaN	NaN	NaN	NaN
1	22.0	0.0	NaN	NaN	NaN	NaN
2	19.0	0.0	NaN	NaN	NaN	NaN
3	26.0	0.0	NaN	NaN	NaN	NaN
4	20.0	0.0	NaN	NaN	NaN	NaN

[S]:

```
print(df.describe())
```

	Age	AttendedBootcamp	BootcampFinish	BootcampFullJobAfter
count	13613.000000	15380.000000	933.000000	635.000000
mean	29.175421	0.061964	0.689175	0.584252
std	9.017716	0.241097	0.463080	0.493239
min	10.000000	0.000000	0.000000	0.000000
25%	23.000000	0.000000	0.000000	0.000000
50%	27.000000	0.000000	1.000000	1.000000
75%	33.000000	0.000000	1.000000	1.000000
max	86.000000	1.000000	1.000000	1.000000

	BootcampLoanYesNo	BootcampMonthsAgo	BootcampPostSalary	\
count	934.000000	631.000000	330.000000	
mean	0.332976	9.055468	63740.506061	
std	0.471531	12.968035	26347.200265	
min	0.000000	0.000000	6000.000000	
25%	0.000000	3.000000	50000.000000	
50%	0.000000	6.000000	60000.000000	
75%	1.000000	12.000000	77000.000000	
max	1.000000	220.000000	200000.000000	

	BootcampRecommend	ChildrenNumber	CodeEventBootcamp	...
count	937.000000	2554.000000	42.0	...
mean	0.785486	1.896241	1.0	...
std	0.410704	1.115975	0.0	...
min	0.000000	0.000000	1.0	...
25%	1.000000	1.000000	1.0	...
50%	1.000000	2.000000	1.0	...
75%	1.000000	2.000000	1.0	...
max	1.000000	18.000000	1.0	...

	ResourceReddit	ResourceSkillCrush	ResourceSoloLearn	\
count	29.0	36.0	30.0	
mean	1.0	1.0	1.0	
std	0.0	0.0	0.0	
min	1.0	1.0	1.0	
25%	1.0	1.0	1.0	
50%	1.0	1.0	1.0	

	1.0	1.0	1.0	
count	ResourceStackOverflow	ResourceTreehouse	ResourceUdacity \	
mean	191.0	422.0	3306.0	
std	1.0	1.0	1.0	
min	0.0	0.0	0.0	
25%	1.0	1.0	1.0	
50%	1.0	1.0	1.0	
75%	1.0	1.0	1.0	
max	1.0	1.0	1.0	
	ResourceUdemy	ResourceW3Schools	ResourceYouTube	StudentDebtOwe
count	4130.0	121.0	121.0	3514.000000
mean	1.0	1.0	1.0	34556.143711
std	0.0	0.0	0.0	54423.139781
min	1.0	1.0	1.0	0.000000
25%	1.0	1.0	1.0	10000.000000
50%	1.0	1.0	1.0	20000.000000
75%	1.0	1.0	1.0	40000.000000
max	1.0	1.0	1.0	1000000.000000

[8 rows x 85 columns]

In [6]:

```
# Lista todas as colunas
list(df)
```

Out[6]:

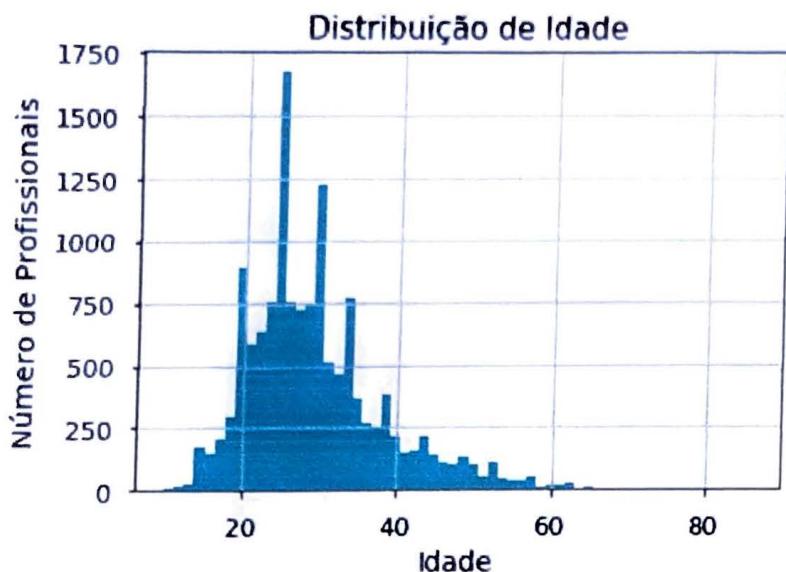
- 'Age',
- 'AttendedBootcamp',
- 'BootcampFinish',
- 'BootcampFullJobAfter',
- 'BootcampLoanYesNo',
- 'BootcampMonthsAgo',
- 'BootcampName',
- 'BootcampPostSalary',
- 'BootcampRecommend',
- 'ChildrenNumber',
- 'CityPopulation',
- 'CodeEventBootcamp',
- 'CodeEventCoffee',
- 'CodeEventConferences',
- 'CodeEventDjangoGirls',
- 'CodeEventGameJam',
- 'CodeEventGirlDev',
- 'CodeEventHackathons'.

## Distribuição de Idade

[7]:

```
# Qual a distribuição de idade dos participantes da pesquisa?
# A maioria dos profissionais que trabalham como programadores de
# software estão na faixa de idade entre 20 e 30 anos, sendo 25 anos
# a idade mais frequente.
```

```
# Gerando um histograma
df.Age.hist(bins = 60)
plt.xlabel("Idade")
plt.ylabel("Número de Profissionais")
plt.title("Distribuição de Idade")
plt.show()
```



## Distribuição de Sexo

[8]:

# Qual é a distribuição de sexo dos participantes da pesquisa?  
# A grande maioria dos programadores é do sexo masculino

# Definindo a quantidade

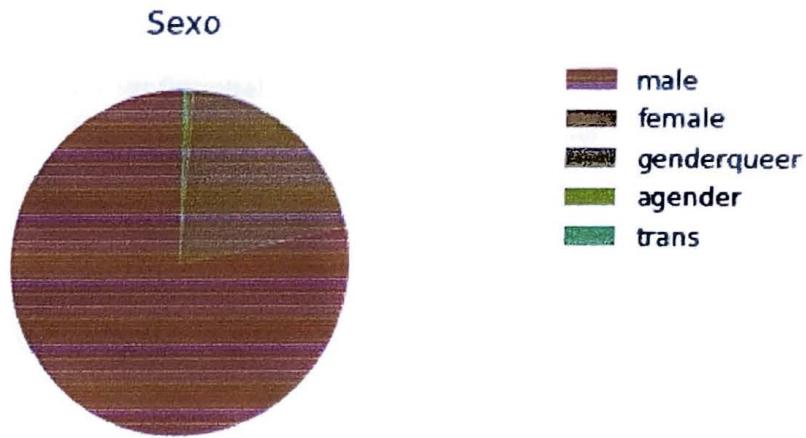
```
labels = df.Gender.value_counts().index
num = len(df.EmploymentField.value_counts().index)
```

# Criando a Lista de cores

```
listaHSV = [(x*1.0/num, 0.5, 0.5) for x in range(num)]
listaRGB = list(map(lambda x: colorsys.hsv_to_rgb(*x), listaHSV))
```

# Gráfico de Pizza

```
fatias, texto = plt.pie(df.Gender.value_counts(), colors = listaRGB, startangle = 90)
plt.axes().set_aspect('equal', 'datalim')
plt.legend(fatias, labels, bbox_to_anchor = (1.05,1))
plt.title("Sexo")
plt.show()
```



## Distribuição de Interesses

[9]:

```
# Quais são os principais interesses dos participantes da pesquisa?
# O principal interesse profissional dos programadores é o desenvolvimento web (Full-Stack,
# seguido pela área de Data Science.

# Definindo a quantidade
num = len(df.JobRoleInterest.value_counts().index)

# Criando a Lista de cores
listaHSV = [(x*1.0/num, 0.5, 0.5) for x in range(num)]
listaRGB = list(map(lambda x: colorsys.hsv_to_rgb(*x), listaHSV))
labels = df.JobRoleInterest.value_counts().index
colors = ['OliveDrab', 'Orange', 'OrangeRed', 'DarkCyan', 'Salmon', 'Sienna', 'Maroon', 'Li

# Gráfico de Pizza
fatias, texto = plt.pie(df.JobRoleInterest.value_counts(), colors = listaRGB, startangle =
plt.axes().set_aspect('equal', 'datalim')
plt.legend(fatias, labels, bbox_to_anchor = (1.25, 1))
plt.title("Interesse Profissional")
plt.show()
```

Interesse Profissional



## Distribuição de Empregabilidade

[10]:

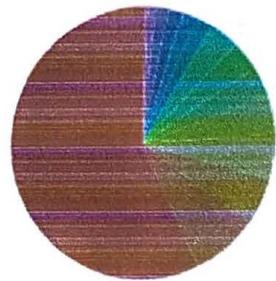
```
# Quais as áreas de negócio em que os participantes da pesquisa trabalham?
# A maioria dos programadores trabalha na área de desenvolvimento de
# softwares e TI, mas outras áreas como finanças e saúde também são
# significativas.

# Definindo a quantidade
num = len(df.EmploymentField.value_counts().index)

# Criando a lista de cores
listaHSV = [(x*1.0/num, 0.5, 0.5) for x in range(num)]
listaRGB = list(map(lambda x: colorsys.hsv_to_rgb(*x), listaHSV))
labels = df.EmploymentField.value_counts().index

# Gráfico de Pizza
fatias, texto = plt.pie(df.EmploymentField.value_counts(), colors = listaRGB, startangle =
plt.axes().set_aspect('equal', 'datalim')
plt.legend(fatias, labels, bbox_to_anchor = (1.3, 1))
plt.title("Área de trabalho Atual")
plt.show()
```

Área de trabalho Atual



- software development and IT
- education
- arts, entertainment, sports, or media
- office and administrative support
- sales
- food and beverage
- finance
- health care
- architecture or physical engineering
- transportation
- software development
- construction and extraction
- legal
- law enforcement and fire and rescue
- farming, fishing, and forestry

## Preferências de Trabalho por Idade

[11]:

```

# Quais são as preferências de trabalho por idade?
# Perceba que à medida que a idade aumenta, o interesse por trabalho
# freelance também aumenta, sendo o modelo preferido por profissionais
# acima de 60 anos. Profissionais mais jovens preferem trabalhar em
# Startups ou no seu próprio negócio. Profissionais entre 20 e 50 anos
# preferem trabalhar em empresas de tamanho médio.

# Agrupando os dados
df_ageranges = df.copy()
bins=[0, 20, 30, 40, 50, 60, 100]

df_ageranges['AgeRanges'] = pd.cut(df_ageranges['Age'],
                                     bins,
                                     labels=["< 20", "20-30", "30-40", "40-50", "50-60", "> 60"])

df2 = pd.crosstab(df_ageranges.AgeRanges,
                   df_ageranges.JobPref).apply(lambda r: r/r.sum(), axis=1)

# Definindo a quantidade
num = len(df_ageranges.AgeRanges.value_counts().index)

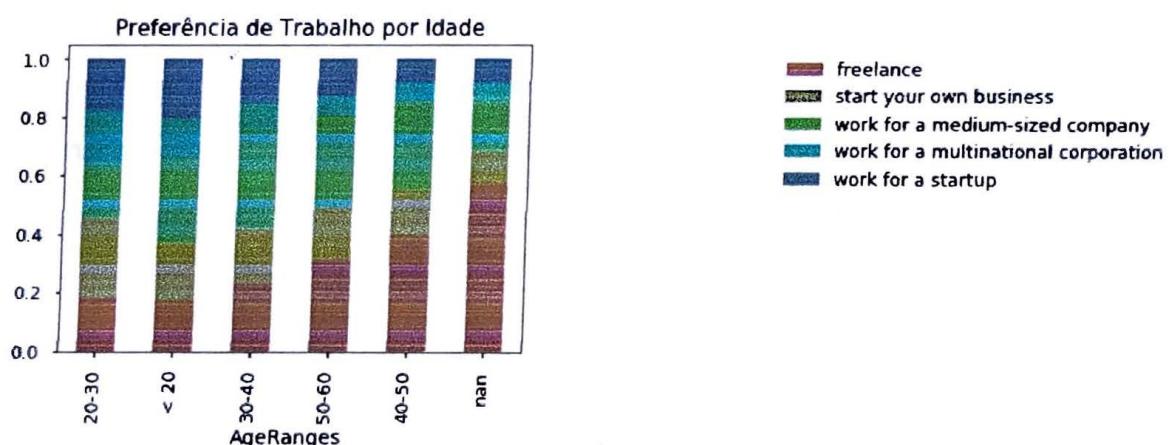
# Criando a lista de cores
listaHSV = [(x*1.0/num, 0.5, 0.5) for x in range(num)]
listaRGB = list(map(lambda x: colorsys.hsv_to_rgb(*x), listaHSV))

# Gráfico de Barras (Stacked)
ax1 = df2.plot(kind = "bar", stacked = True, color = listaRGB, title = "Preferência de Trabalho por Idade")
lines, labels = ax1.get_legend_handles_labels()
ax1.legend(lines, labels, bbox_to_anchor = (1.51, 1))

```

Out[11]:

&lt;matplotlib.legend.Legend at 0x1085026a0&gt;



[12]:

```
# Visualizando o help
help(pd.crosstab)
```

Help on function crosstab in module pandas.core.reshape.pivot:

```
crosstab(index, columns, values=None, rownames=None, colnames=None, aggfunc=
None, margins=False, margins_name='All', dropna=True, normalize=False)
Compute a simple cross-tabulation of two (or more) factors. By default
computes a frequency table of the factors unless an array of values and
an
aggregation function are passed
```

#### Parameters

-----

```
index : array-like, Series, or list of arrays/Series
    Values to group by in the rows
columns : array-like, Series, or list of arrays/Series
    Values to group by in the columns
values : array-like, optional
    Array of values to aggregate according to the factors.
    Requires `aggfunc` be specified.
```

```
aggfunc : function, optional
    If specified, requires `values` be specified as well
rownames : sequence, default None
```

```
If passed, must match number of row arrays passed
colnames : sequence, default None
    If passed, must match number of column arrays passed
```

```
margins : boolean, default False
    Add row/column margins (subtotals)
margins_name : string, default 'All'
```

```
Name of the row / column that will contain the totals
when margins is True.
```

.. versionadded:: 0.21.0

dropna : boolean, default True

Do not include columns whose entries are all NaN

normalize : boolean, {'all', 'index', 'columns'}, or {0,1}, default Fals

e

Normalize by dividing all values by the sum of values.

- If passed 'all' or 'True', will normalize over all values.
- If passed 'index' will normalize over each row.
- If passed 'columns' will normalize over each column.
- If margins is 'True', will also normalize margin values.

.. versionadded:: 0.18.1

#### Notes

-----

Any Series passed will have their name attributes used unless row or column names for the cross-tabulation are specified.

Any input passed containing Categorical data will have \*\*all\*\* of its categories included in the cross-tabulation, even if the actual data doe

not contain any instances of a particular category.

In the event that there aren't overlapping indexes an empty DataFrame will be returned.

### Examples

```
-----
>>> a = np.array(["foo", "foo", "foo", "foo", "bar", "bar",
...                 "bar", "bar", "foo", "foo", "foo"], dtype=object)
>>> b = np.array(["one", "one", "one", "two", "one", "one",
...                 "one", "two", "two", "two", "one"], dtype=object)
>>> c = np.array(["dull", "dull", "shiny", "dull", "dull", "shiny",
...                 "shiny", "dull", "shiny", "shiny", "shiny"],
...                 dtype=object)

>>> pd.crosstab(a, [b, c], rownames=['a'], colnames=['b', 'c'])
... # doctest: +NORMALIZE_WHITESPACE
b   one      two
c dull shiny dull shiny
a
bar    1      2    1      0
foo    2      2    1      2

>>> foo = pd.Categorical(['a', 'b'], categories=['a', 'b', 'c'])
>>> bar = pd.Categorical(['d', 'e'], categories=['d', 'e', 'f'])
>>> crosstab(foo, bar) # 'c' and 'f' are not represented in the data,
...                      # but they still will be counted in the output
... # doctest: +SKIP
col_0  d  e  f
row_0
a      1  0  0
b      0  1  0
c      0  0  0
```

### Returns

crosstab : DataFrame

## Reallocação por Idade

[13]:

```

# Qual o objetivo de relocação?
# A vontade de buscar um novo emprego diminui com a idade.
# Quase 80% das pessoas abaixo dos 30 anos estão preparadas para isso.

# Agrupando os dados
df3 = pd.crosstab(df_ageranges.AgeRanges,
                   df_ageranges.JobRelocateYesNo).apply(lambda r: r/r.sum(), axis = 1)

# Definindo a quantidade
num = len(df_ageranges.AgeRanges.value_counts().index)

# Criando a lista de cores
listaHSV = [(x*1.0/num, 0.5, 0.5) for x in range(num)]
listaRGB = list(map(lambda x: colorsys.hsv_to_rgb(*x), listaHSV))

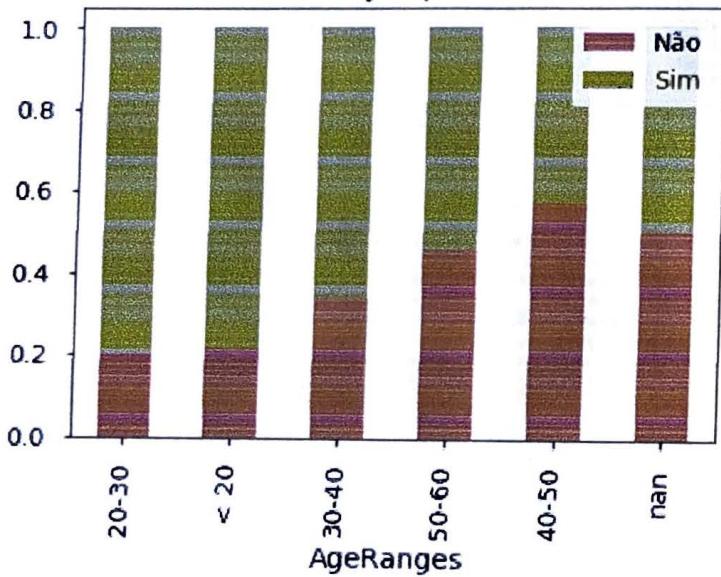
# Gráfico de Barras (Stacked)
ax1 = df3.plot(kind = "bar", stacked = True, color = listaRGB, title = "Relocação por Idade",
               lines, labels = ax1.get_legend_handles_labels())
ax1.legend(lines,[ "Não", "Sim"], loc = 'best')

```

Out[13]:

&lt;matplotlib.legend.Legend at 0x10f67cf28&gt;

Relocação por Idade



## Idade x Horas de Aprendizagem

[14]:

```

# Qual a relação entre idade e horas de aprendizagem?
# A idade dos profissionais não afeta a quantidade de tempo gasto com capacitação e treinamento

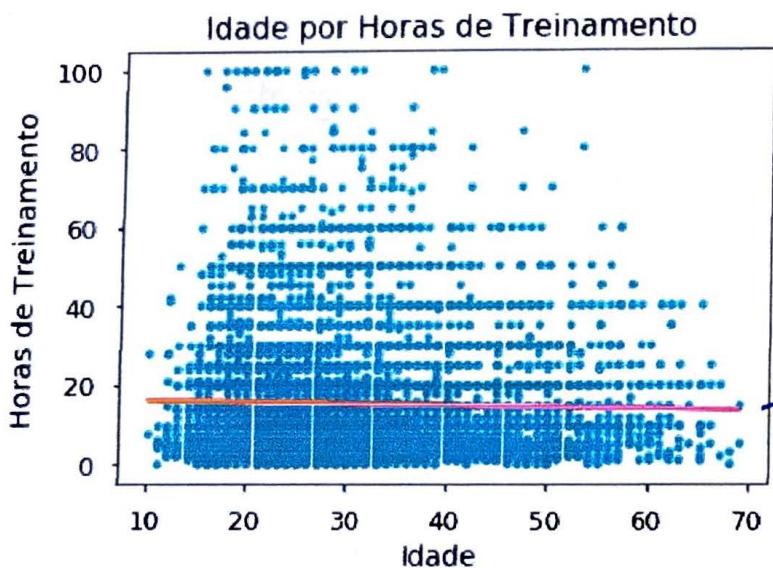
import warnings
warnings.filterwarnings('ignore')

# Criando subset dos dados
df9 = df.copy() → Cria uma cópia do data frame
df9 = df9.dropna(subset=["HoursLearning"]) → remove valores missing
df9 = df9[df['Age'].isin(range(0,70))]

# Definindo os valores de x e y
x = df9.Age
y = df9.HoursLearning

# Computando os valores e gerando o gráfico
m, b = np.polyfit(x, y, 1)
plt.plot(x, y, '.')
plt.plot(x, m*x + b, '-', color = "red")
plt.xlabel("Idade")
plt.ylabel("Horas de Treinamento")
plt.title("Idade por Horas de Treinamento")
plt.show()

```



Significa que os profissionais mais estão sempre estudando

## Investimento em Capacitação x Expectativa Salarial

[15]:

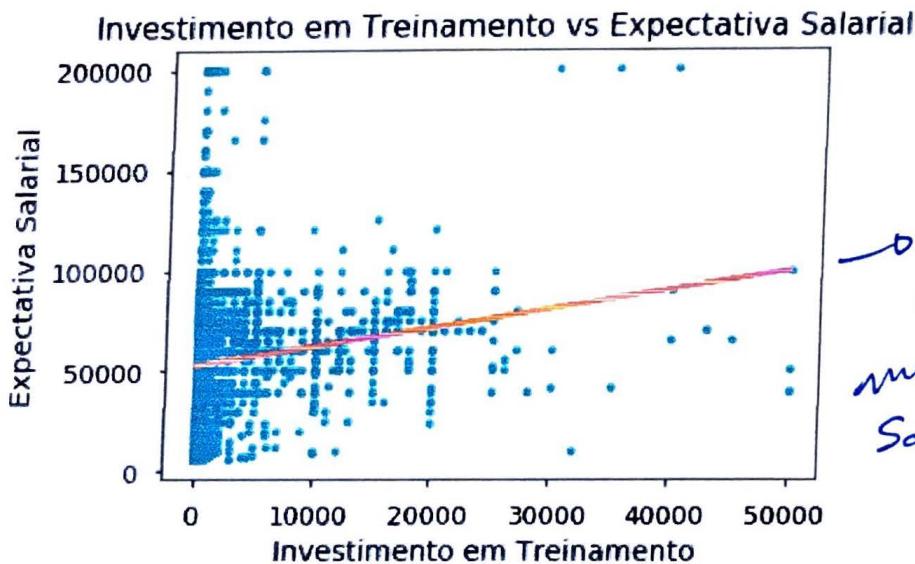
# Qual a relação entre investimento em capacitação e expectativa salarial?  
# Os profissionais que investem tempo e dinheiro em capacitação e  
# treinamento, em geral, conseguem salários mais altos, embora alguns  
# profissionais esperem altos salários, investindo 0 em treinamento.

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

# Criando subset dos dados
df5 = df.copy()
df5 = df5.dropna(subset=["ExpectedEarning"])
df5 = df5[df['MoneyForLearning'].isin(range(0, 60000))]

# Definindo os valores de x e y
x = df5.MoneyForLearning
y = df5.ExpectedEarning

# Computando os valores e gerando o gráfico
m, b = np.polyfit(x, y, 1)
plt.plot(x, y, '.')
plt.plot(x, m*x + b, '--', color = "red")
plt.xlabel("Investimento em Treinamento")
plt.ylabel("Expectativa Salarial")
plt.title("Investimento em Treinamento vs Expectativa Salarial")
plt.show()
```



Conheça a Formação Cientista de Dados, um programa completo, 100% online e 100% em português, com 340 horas, mais de 1.200 aulas em vídeos e 26 projetos, que vão ajudá-lo a se tornar um dos profissionais mais cobiçados do mercado de análise de dados. Clique no link abaixo, faça sua inscrição, comece hoje mesmo e aumente sua empregabilidade:

<https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados>  
[\(https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados\)](https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados)

# Fim

# Data Science Academy - Python Fundamentos - Capítulo 9

Download: <http://github.com/dsacademybr>  
<http://github.com/dsacademybr>)

## Exercício: Análise Exploratória de Dados com Python

Neste exercício, você vai realizar uma análise exploratória em um dos mais famosos datasets para Machine Learning, o dataset iris com informações sobre 3 tipos de plantas. Esse dataset é comumente usado em problemas de Machine Learning de classificação, quando nosso objetivo é prever a classe dos dados. No caso deste dataset, prever a categoria de uma planta a partir de medidas da planta (sepal e petal).

Em cada célula, você encontra a tarefa a ser realizada. Faça todo o exercício e depois compare com a solução proposta.

Dataset (já disponível com o Scikit-Learn): <https://archive.ics.uci.edu/ml/datasets/iris>  
<https://archive.ics.uci.edu/ml/datasets/iris>)

In [1]:

```
# Imports
import time
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.datasets import load_iris
%matplotlib inline

fontsize = 14
ticklabelsize = 14
```

[2]:

```
# Carregando o dataset
iris = load_iris()
df = pd.DataFrame(iris.data, columns=iris.feature_names)
print(len(df))
df.head()
```

150

Out[2]:

	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
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

# Extração e Transformação de Dados

In [3]:

```
# Imprima os valores numéricos da Variável target (o que queremos prever),  
# uma de 3 possíveis categorias de plantas: setosa, versicolor ou virginica  
iris.target_names
```

Out[3]:

```
array(['setosa', 'versicolor', 'virginica'], dtype='|<U10')
```

In [4]:

```
# Imprima os valores numéricos da Variável target (o que queremos prever),  
# uma de 3 possíveis categorias de plantas: 0, 1 ou 2  
iris.target
```

out[4]:

In [5]:

```
# Adicione ao dataset uma nova coluna com os nomes das espécies, pois é isso que vamos tentar
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
df.head()
```

Out[5]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [6]:

```
# Inclua no dataset uma coluna com os valores numéricos da variável target
df['target'] = iris.target
df.head()
```

Out[6]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	species	target
0	5.1	3.5	1.4	0.2	setosa	0
1	4.9	3.0	1.4	0.2	setosa	0
2	4.7	3.2	1.3	0.2	setosa	0
3	4.6	3.1	1.5	0.2	setosa	0
4	5.0	3.6	1.4	0.2	setosa	0

In [7]:

```
# Extraia as features (atributos) do dataset e imprima
features = df.columns[:4]
features
```

Out[7]:

```
Index(['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)',  
       'petal width (cm)'],
      dtype='object')
```

In [8]:

```
# Calcule a média de cada feature para as 3 classes
df.groupby('target').mean().T
```

Out[8]:

target	0	1	2
sepal length (cm)	5.006	5.936	6.588
sepal width (cm)	3.418	2.770	2.974
petal length (cm)	1.464	4.260	5.552
petal width (cm)	0.244	1.326	2.026

## Exploração de Dados

In [9]:

```
# Imprima uma Transposta do dataset (transforme linhas e colunas e colunas em linhas)
df.head(10).T
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9
sepal length (cm)	5.1	4.9	4.7	4.6	5	5.4	4.6	5	4.4	4.9
sepal width (cm)	3.5	3	3.2	3.1	3.6	3.9	3.4	3.4	2.9	3.1
petal length (cm)	1.4	1.4	1.3	1.5	1.4	1.7	1.4	1.5	1.4	1.5
petal width (cm)	0.2	0.2	0.2	0.2	0.2	0.4	0.3	0.2	0.2	0.1
species	setosa									
target	0	0	0	0	0	0	0	0	0	0

In [10]:

```
# Utilize a função Info do dataset para obter um resumo sobre o dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
sepal length (cm)    150 non-null float64
sepal width (cm)     150 non-null float64
petal length (cm)    150 non-null float64
petal width (cm)     150 non-null float64
species              150 non-null category
target               150 non-null int64
dtypes: category(1), float64(4), int64(1)
memory usage: 6.2 KB
```

In [11]:

```
# Faça um resumo estatístico do dataset
df.describe()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667	1.000000
std	0.828066	0.433594	1.764420	0.763161	0.819232
min	4.300000	2.000000	1.000000	0.100000	0.000000
25%	5.100000	2.800000	1.600000	0.300000	0.000000
50%	5.800000	3.000000	4.350000	1.300000	1.000000
75%	6.400000	3.300000	5.100000	1.800000	2.000000
max	7.900000	4.400000	6.900000	2.500000	2.000000

In [12]:

```
# Verifique se existem valores nulos no dataset
df.isnull().sum(axis=0)
```

Out[12]:

```
sepal length (cm)      0
sepal width (cm)      0
petal length (cm)      0
petal width (cm)      0
species                 0
target                  0
dtype: int64
```

In [13]:

```
# Faça uma contagem de valores de sepal length  
df['sepal length (cm)'].value_counts(dropna=False)
```

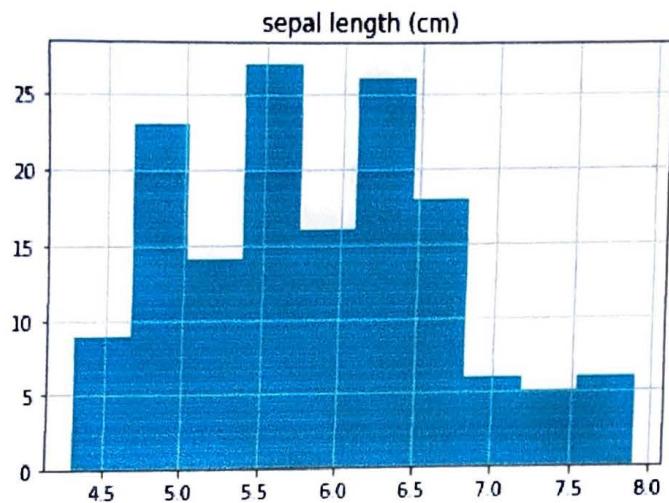
Out[13]:

```
5.0    10  
6.3     9  
5.1     9  
6.7     8  
5.7     8  
5.5     7  
5.8     7  
6.4     7  
6.0     6  
4.9     6  
6.1     6  
5.4     6  
5.6     6  
6.5     5  
4.8     5  
7.7     4  
6.9     4  
5.2     4  
6.2     4  
4.6     4  
7.2     3  
6.8     3  
4.4     3  
5.9     3  
6.6     2  
4.7     2  
7.6     1  
7.4     1  
4.3     1  
7.9     1  
7.3     1  
7.0     1  
4.5     1  
5.3     1  
7.1     1  
Name: sepal length (cm), dtype: int64
```

## Plot

In [14]:

```
# Crie um Histograma de sepal Length
exclude = ['Id', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)', 'target']
df.loc[:, df.columns.difference(exclude)].hist()
plt.figure(figsize=(15,10))
plt.show()
```



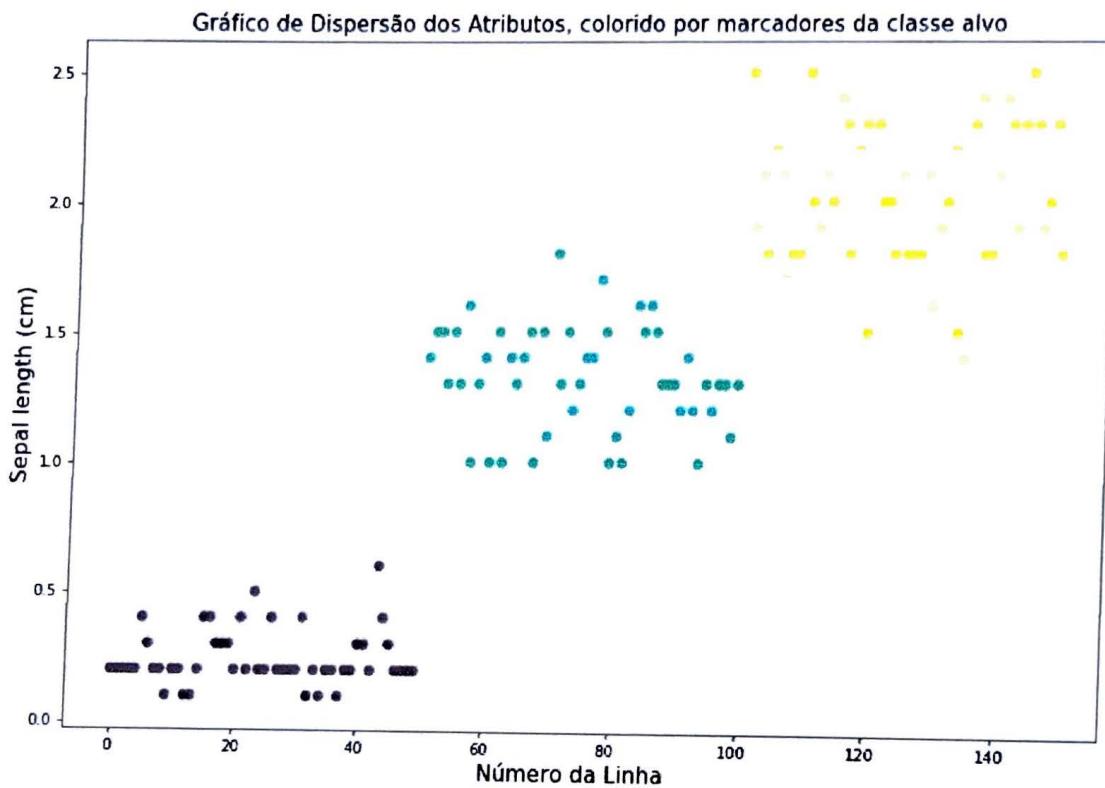
&lt;Figure size 1080x720 with 0 Axes&gt;

In [15]:

```
# Crie um Gráficos de Dispersão (scatter Plot) da variável sepal length versus número da Linha
# colorido por marcadores da variável target
plt.figure(figsize=(12, 8), dpi=80)
plt.scatter(range(len(df)), df['petal width (cm)'], c=df['target'])
plt.xlabel('Número da Linha', fontsize=fontsize)
plt.ylabel('Sepal length (cm)', fontsize=fontsize)
plt.title('Gráfico de Dispersão dos Atributos, colorido por marcadores da classe alvo', fontweight='bold')
# plt.title('Scatter plot of features, colored by target labels', fontsize=fontsize)
```

Out[15]:

Text(0.5,1,'Gráfico de Dispersão dos Atributos, colorido por marcadores da classe alvo')

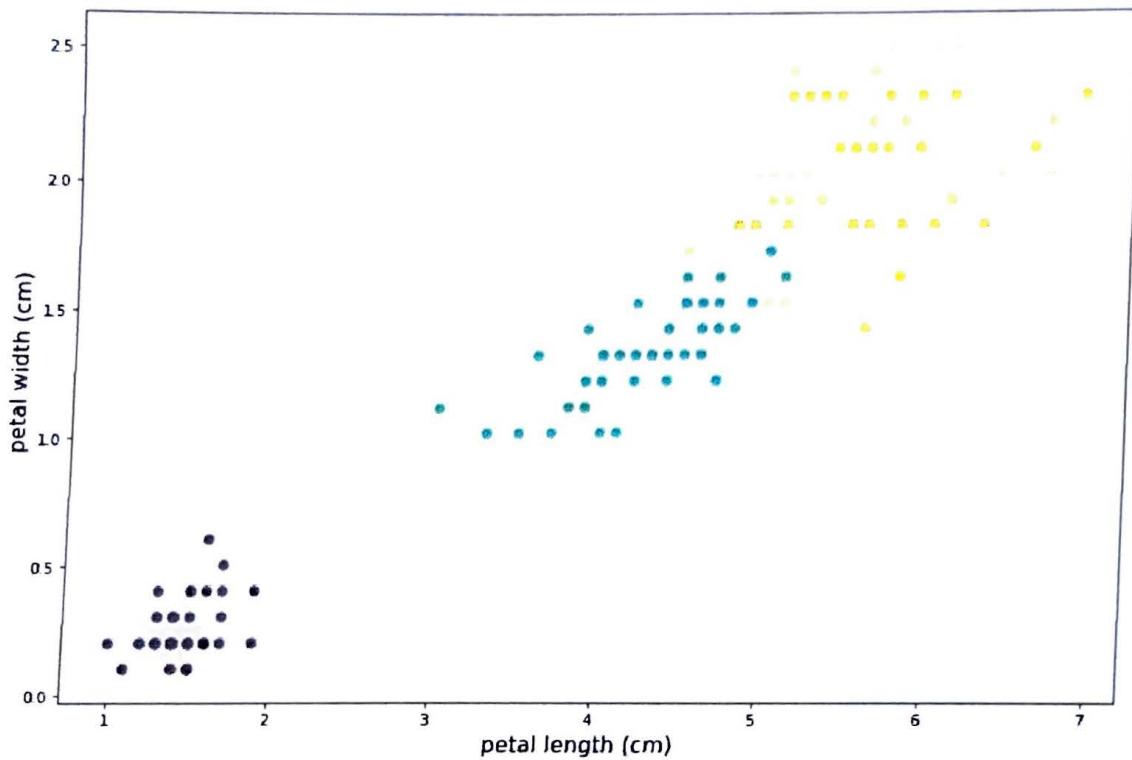


In [16]:

```
# Crie um Scatter Plot de 2 Features (atributos)
plt.figure(figsize=(12, 8), dpi=80)
plt.scatter(df['petal length (cm)'], df['petal width (cm)'], c=df['target'])
plt.xlabel('petal length (cm)', fontsize=fontsize)
plt.ylabel('petal width (cm)', fontsize=fontsize)
```

Out[16]:

Text(0,0.5,'petal width (cm)')

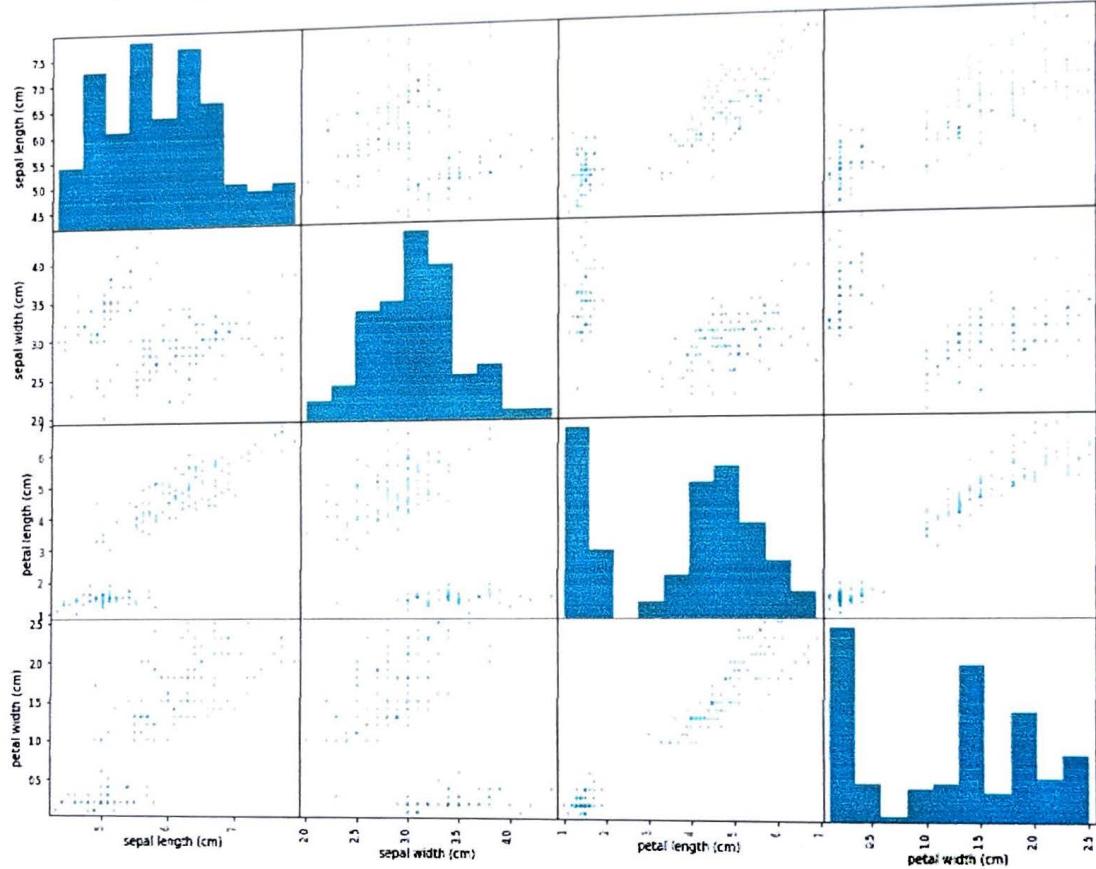


In [17]:

```
# Crie um Scatter Matrix das Features (atributos)
attributes = ['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
pd.plotting.scatter_matrix(df[attributes], figsize=(16, 12))
```

Out[17]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1097c2dd8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1097ec6a0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1098369b0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x109860cc0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x10988bfd0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x109892048>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a15a146a0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a15a3cd30>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x1a15a6c400>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a15a92a90>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a15ac6160>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a15aee7f0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x1a15b15e80>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a15b48550>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a15b70be0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a15ba12b0>]],
      dtype=object)
```

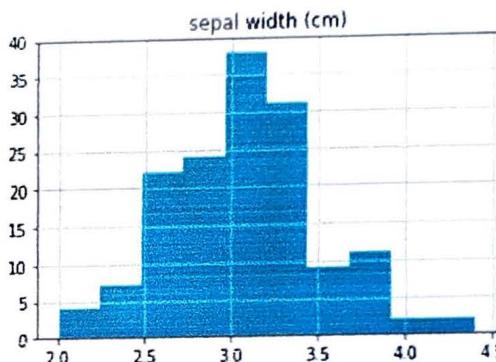
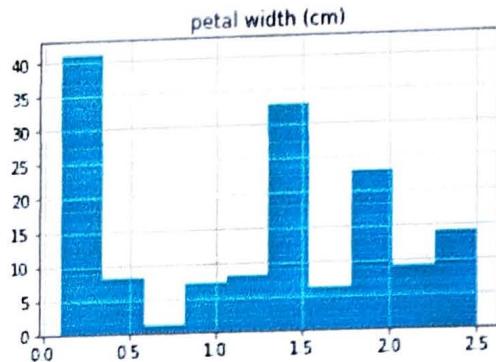
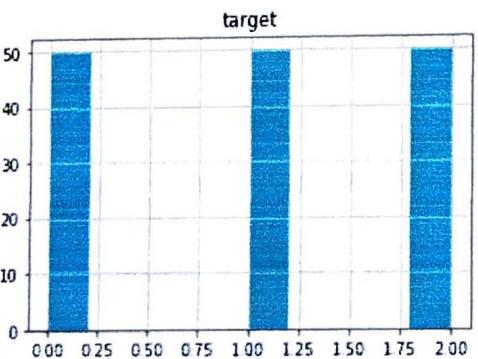
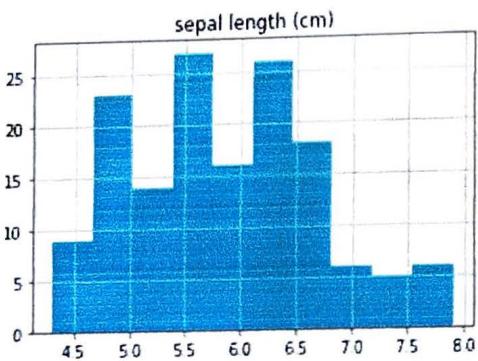
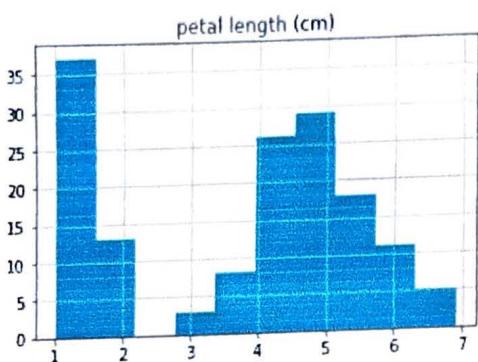


In [18]:

```
# Crie um Histograma de todas as features
df.hist(figsize=(12,12))
```

Out[18]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1a161d3550>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a161fafd0>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x1a164a9668>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a164cecf8>],
      [<matplotlib.axes._subplots.AxesSubplot object at 0x1a165013c8>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x1a16501400>]],
     dtype=object)
```



Conheça a Formação Cientista de Dados, um programa completo, 100% online e 100% em português, com 340 horas, mais de 1.200 aulas em vídeos e 26 projetos, que vão ajudá-lo a se tornar um dos profissionais mais cobiçados do mercado de análise de dados. Clique no link abaixo, faça sua inscrição, comece hoje mesmo e aumente sua empregabilidade:

<https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados>  
[\(https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados\)](https://www.datascienceacademy.com.br/pages/formacao-cientista-de-dados)

# Data Science Academy - Python Fundamentos - Capítulo 10

Download: <http://github.com/dsacademybr>  
<http://github.com/dsacademybr>)

## Lab 4 - Construindo um Modelo de Regressão Linear com TensorFlow

Use como referência o Deep Learning Book: <http://www.deeplearningbook.com.br/>  
<http://www.deeplearningbook.com.br/>)

In [1]:

```
# Imports
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

### Definindo os **hyperparâmetros** do modelo

In [2]:

```
# Hyperparâmetros do modelo
learning_rate = 0.01
training_epochs = 2000 → passados de treinamento, quantas Vezes serão
display_step = 200 treinado
```

### Definindo os datasets de treino e de teste

Considere **X** como o tamanho de uma casa e **y** o preço de uma casa

*X*

In [3]:

```
# Dataset de treino
train_X = np.asarray([3.3,4.4,5.5,6.71,6.93,4.168,9.779,6.182,7.59,2.167,7.042,10.791,5.313
train_y = np.asarray([1.7,2.76,2.09,3.19,1.694,1.573,3.366,2.596,2.53,1.221,2.827,3.465,1.6
n_samples = train_X.shape[0]

# Dataset de teste
test_X = np.asarray([6.83, 4.668, 8.9, 7.91, 5.7, 8.7, 3.1, 2.1])
test_y = np.asarray([1.84, 2.273, 3.2, 2.831, 2.92, 3.24, 1.35, 1.03])
```

## Placeholders e variáveis

In [4]:

```
# Placeholders para as variáveis preditoras (x) e para variável target (y)
x = tf.placeholder(tf.float32)
y = tf.placeholder(tf.float32)

# Pesos e bias do modelo
W = tf.Variable(np.random.randn(), name="weight")
b = tf.Variable(np.random.randn(), name="bias")
```

↳ reserva espaço no memória  
Sem o Shape você pode adicionar variáveis de qualquer tamanho depois

## Construindo o modelo

In [5]:

```
# Construindo o modelo Linear
# Fórmula do modelo Linear:  $y = W \cdot X + b$ 
linear_model = W*X + b

# Mean squared error (erro quadrado médio)
cost = tf.reduce_sum(tf.square(linear_model - y)) / (2*n_samples)

# Otimização com Gradient descent
optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
```

existem mais de 60 modelos de machine learning

taxa de erro      Gradient = dividido

↓      ↓  
previsão      valor real

↳ controla a forma como os pesos serão atualizados

## Executando o grafo computacional, treinando e testando o modelo

Jo temos o x e y

o modelo vai aprender o que é o W e o b.

$y =$

Linear model é o resultado do modelo

Será composto com o valor real x

In [6]:

```
# Definindo a inicialização das variáveis
init = tf.global_variables_initializer()

# Iniciando a sessão
with tf.Session() as sess:
    # Iniciando as variáveis
    sess.run(init)

    # Treinamento do modelo
    for epoch in range(training_epochs):
        # Otimização com Gradient Descent
        sess.run(optimizer, feed_dict={X: train_X, y: train_y})
        # Display de cada epoch
        if (epoch+1) % display_step == 0:
            c = sess.run(cost, feed_dict={X: train_X, y: train_y})
            print("Epoch:{0}: {1} Cost:{2:.10f} {3:.4f} {4:.4f}" .format(epoch+1,
                c, sess.run(W), sess.run(b)))
        # Imprimindo os parâmetros finais do modelo
        print("\nOtimização Concluída!")
        training_cost = sess.run(cost, feed_dict={X: train_X, y: train_y})
        print("Custo Final de Treinamento:", training_cost, " - W Final:", sess.run(W), " - b ")

# Visualizando o resultado
plt.plot(train_X, train_y, 'ro', label='Dados Originais')
plt.plot(train_X, sess.run(W) * train_X + sess.run(b), label='Linha de Regressão')
plt.legend()
plt.show()  → imprime todos os parâmetros do modelo
            → visualiza o resultado através do plt
```

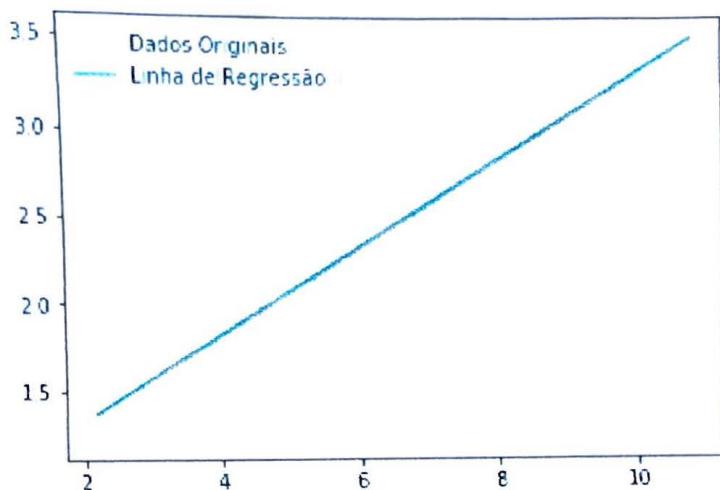
```
sess.close()
```

Epoch:	Cost:	W:	b:
200	0.08133	0.214	1.065
400	0.07964	0.2221	1.008
600	0.07859	0.2285	0.9628
800	0.07795	0.2335	0.9275
1000	0.07756	0.2374	0.8997
1200	0.07732	0.2405	0.878
1400	0.07717	0.2429	0.8609
1600	0.07708	0.2448	0.8475
1800	0.07702	0.2462	0.837
2000	0.07698	0.2474	0.8288

Otimização Concluída!

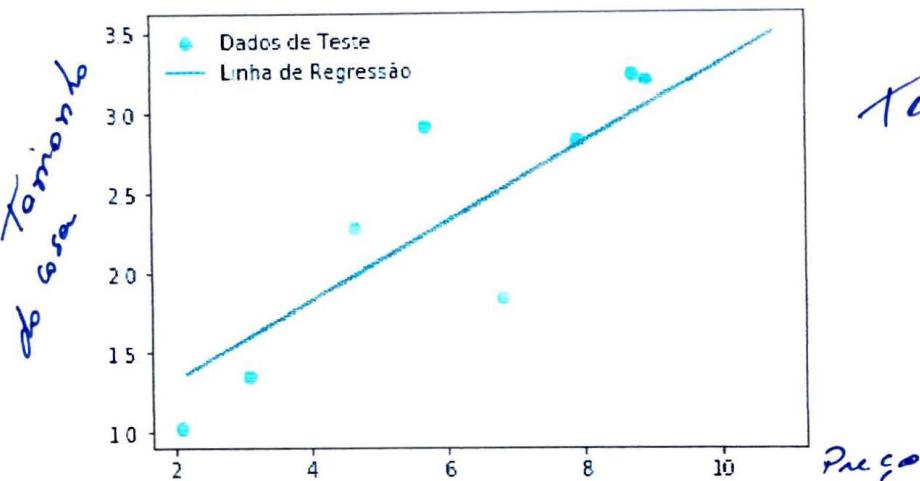
→ o Custo = taxa de erro vai diminuindo com os passos

Custo Final de Treinamento: 0.07698439 - W Final: 0.24740951 - b Final: 0.8287576



Treino

Custo Final em Teste: 0.07920746  
Diferença Média Quadrada Absoluta: 0.002223067



Teste

Preço

Conheça a Formação Inteligência Artificial, um programa completo, 100% online e 100% em português, com 402 horas em 9 cursos de nível intermediário/avançado, que vão ajudá-lo a se tornar um dos profissionais mais cobiçados do mercado de tecnologia. Clique no link abaixo, faça sua inscrição, comece hoje mesmo e aumente sua empregabilidade:

<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>  
(<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>)

**Fim**

**Obrigado - Data Science Academy - [facebook.com/dsacademybr](https://facebook.com/dsacademybr)**  
(<http://facebook.com/dsacademybr>)

# Data Science Academy - Python Fundamentos - Capítulo 10

Download: <http://github.com/dsacademybr>  
[\(http://github.com/dsacademybr\)](http://github.com/dsacademybr)

## Introdução ao TensorFlow → Fluxo de tensors

O Tensorflow é uma das bibliotecas mais amplamente utilizadas para implementar o aprendizado de máquina e outros algoritmos que envolvem grandes operações matemáticas. O Tensorflow foi desenvolvido pelo Google e é uma das bibliotecas de aprendizado de máquina mais populares no GitHub. O Google usa o Tensorflow para aprendizado de máquina em quase todos os aplicativos. Se você já usou o Google Photos ou o Google Voice Search, então já utilizou uma aplicação criada com a ajuda do TensorFlow. Vamos compreender os detalhes por trás do TensorFlow.

Matematicamente, um tensor é um vetor N-dimensional, significando que um tensor pode ser usado para representar conjuntos de dados N-dimensionais. Aqui está um exemplo:

In [1]:

```
from IPython.display import Image  
Image('imagens/tensor1.png')
```

importante

Out[1]:

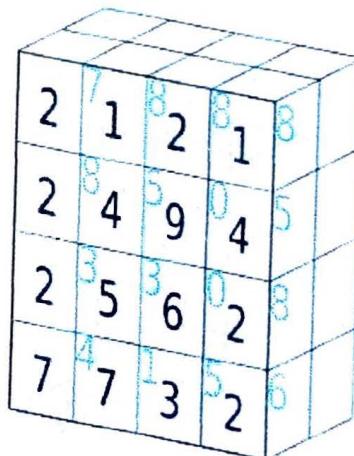
# tensor

't'
'e'
'n'
's'
'o'
'r'

tensor of dimensions [6]  
(vector of dimension 6)

3	1	4	1
5	9	2	6
5	3	5	8
9	7	9	3
2	3	8	4
6	2	6	4

tensor of dimensions [6,4]  
(matrix 6 by 4)



tensor of dimensions [4,4,2]

A figura acima mostra alguns tensores simplificados com dimensões mínimas. À medida que a dimensão continua crescendo, os dados se tornam mais e mais complexos. Por exemplo, se pegarmos um Tensor da forma (3x3), posso chamá-lo de matriz de 3 linhas e colunas. Se eu selecionar outro Tensor de forma (1000x3x3), posso chamá-lo como tensor ou conjunto de 1000 matrizes 3x3. Aqui chamamos (1000x3x3) como a forma ou dimensão do tensor resultante. Os tensores podem ser constantes ou variáveis.

In [2]:

```
from IPython.display import Image  
Image('imagens/tensor2.png')
```

Out[2]:

```
- 3 # Tensor Rank 0 é um escalar, nesse exemplo com shape [ ].  
- [1, 2, 3] # Tensor Rank 1 é um vetor, nesse exemplo com shape [3].  
- [[1, 2, 3], [4, 5, 6]]. # Tensor Rank 2 é uma matriz, nesse exemplo com shape [2, 3].  
- [[[1, 2, 3]], [[7, 8, 9]]] # Tensor Rank 3 é um tensor, nesse caso com shape [2, 1, 3].
```

## TensorFlow x NumPy

*tem limitações*

TensorFlow e NumPy são bastante semelhantes (ambos são bibliotecas de matriz N-d). NumPy é o pacote fundamental para computação científica com Python. Ele contém um poderoso objeto array N-dimensional, funções sofisticadas (broadcasting) e etc. Acredito que os usuários Python não podem viver sem o NumPy. O NumPy tem suporte a matriz N-d, mas não oferece métodos para criar funções de tensor e automaticamente computar derivadas, além de não ter suporte a GPU, e esta é uma das principais razões para a existência do TensorFlow. Abaixo uma comparação entre NumPy e TensorFlow, e você vai perceber que muitas palavras-chave são semelhantes.

In [3]:

```
from IPython.display import Image  
Image('imagens/tf_numpy.png')
```

Out[3]:

### Numpy

```
a = np.zeros((2,2)); b = np.ones((2,2))  
  
np.sum(b, axis=1)  
  
a.shape  
  
np.reshape(a, (1,4))  
  
b * 5 + 1  
  
np.dot(a,b)  
  
a[0,0], a[:,0], a[0,:]
```

### TensorFlow

```
a = tf.zeros((2,2)), b = tf.ones((2,2))  
  
tf.reduce_sum(a,reduction_indices=[1])  
  
a.get_shape()  
  
tf.reshape(a, (1,4))  
  
b * 5 + 1  
  
tf.matmul(a, b)  
  
a[0,0], a[:,0], a[0,:]
```

## Grafo Computacional

Conheça a Formação Inteligência Artificial, um programa completo, 100% online e 100% em português, com 402 horas em 9 cursos de nível intermediário/avançado, que vão ajudá-lo a se tornar um dos profissionais mais cobiçados do mercado de tecnologia. Clique no link abaixo, faça sua inscrição, comece hoje mesmo e aumente sua empregabilidade:

<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>  
(<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>)

Agora sabemos o que o tensor realmente significa e é hora de entender o fluxo. Este fluxo refere-se a um grafo computacional ou simplesmente um grafo.

Grafos computacionais são uma boa maneira de pensar em expressões matemáticas. O conceito de grafo foi introduzido por Leonhard Euler em 1736 para tentar resolver o problema das Pontes de Konigsberg. Grafos são modelos matemáticos para resolver problemas práticos do dia a dia, com várias aplicações no mundo real tais como: circuitos elétricos, redes de distribuição, relações de parentesco entre pessoas, análise de redes sociais, logística, redes de estradas, redes de computadores e muito mais. Grafos são muito usados para modelar problemas em computação.

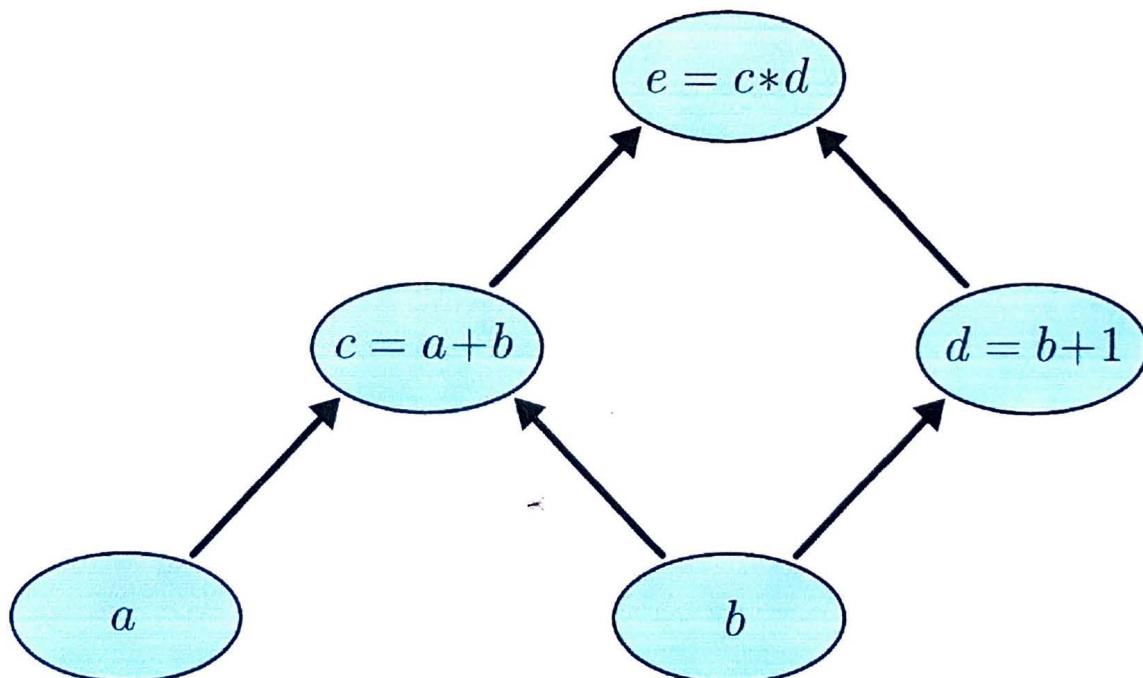
Um Grafo é um modelo matemático que representa relações entre objetos. Um grafo  $G = (V, E)$  consiste de um conjunto de vértices  $V$  (também chamados de nós), ligados por um conjunto de bordas ou arestas  $E$ .

Considere o diagrama abaixo:

In [4]:

```
from IPython.display import Image  
Image('imagens/grafo1.png')
```

Out[4]:



Existem três operações: duas adições e uma multiplicação. Ou seja:

- $c = a+b$
- $d = b+1$
- $e = c*d$

Para criar um grafo computacional, fazemos cada uma dessas operações nos nós, juntamente com as variáveis de entrada. Quando o valor de um nó é a entrada para outro nó, uma seta vai de um para outro e temos nesse caso um grafo direcionado.

Esses tipos de grafos surgem o tempo todo em Ciência da Computação, especialmente ao falar sobre programas funcionais. Eles estão intimamente relacionados com as noções de grafos de dependência e grafos de chamadas. Eles também são a principal abstração por trás do popular framework de Deep Learning, o TensorFlow.

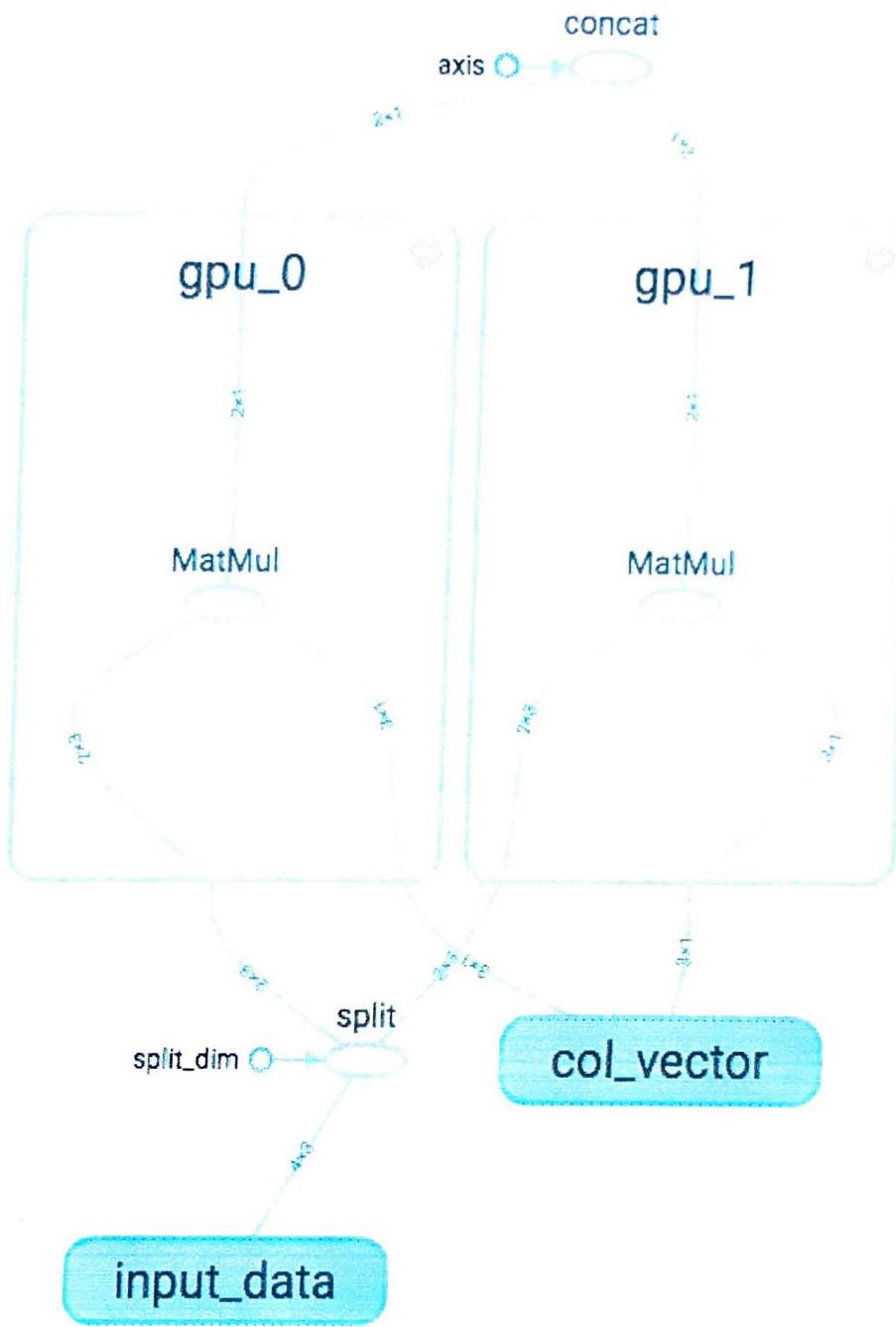
Para mais detalhes sobre grafos, leia um dos capítulos do Deep Learning Book:

<http://deeplearningbook.com.br/algoritmo-backpropagation-parte1-grafos-computacionais-e-chain-rule/>  
[\(http://deeplearningbook.com.br/algoritmo-backpropagation-parte1-grafos-computacionais-e-chain-rule/\)](http://deeplearningbook.com.br/algoritmo-backpropagation-parte1-grafos-computacionais-e-chain-rule/)

Um grafo para execução de um modelo de Machine Learning pode ser bem grande e podemos executar subgrafos (porções dos grafos) em dispositivos diferentes, como uma GPU. Exemplo:

In [5]:

```
from IPython.display import Image  
Image('imagens/grafos2.png')
```



A figura acima explica a execução paralela de sub-grafos. Aqui estão 2 operações de multiplicação de matrizes, já que ambas estão no mesmo nível. Os nós são executados em `gpu_0` e `gpu_1` em paralelo.

## Modelo de Programação TensorFlow

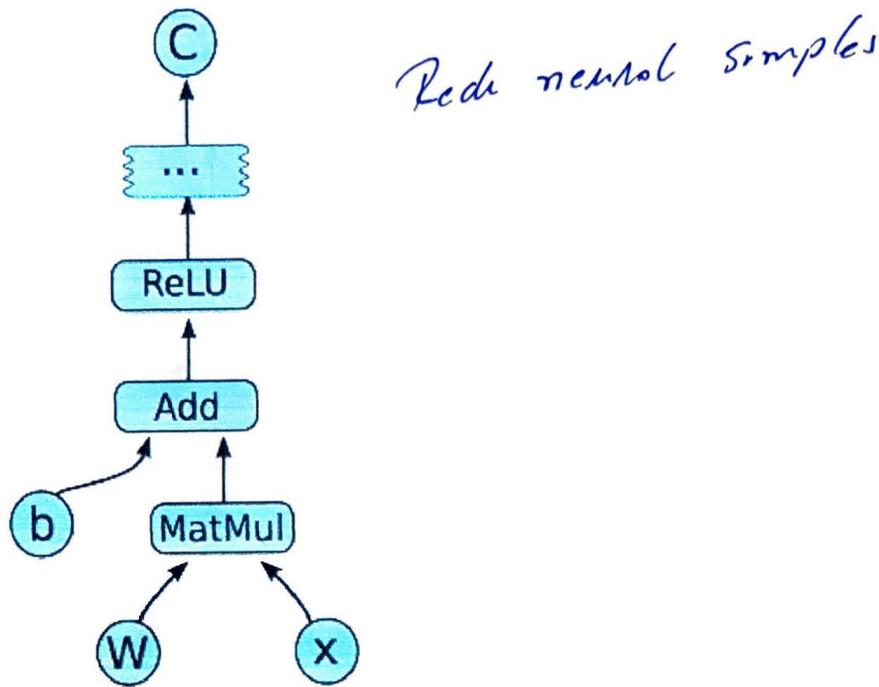
O principal objetivo de um programa TensorFlow é expressar uma computação numérica como um grafo direcionado. A figura abaixo é um exemplo de grafo de computação, que representa o cálculo de  $h = \text{ReLU}$

$(Wx + b)$ . Este é um componente muito clássico em muitas redes neurais, que conduz a transformação linear dos dados de entrada e, em seguida, alimenta uma linearidade (função de ativação linear retificada, neste caso).

In [6]:

```
from IPython.display import Image  
Image('imagens/grafos.png')
```

Out[6]:

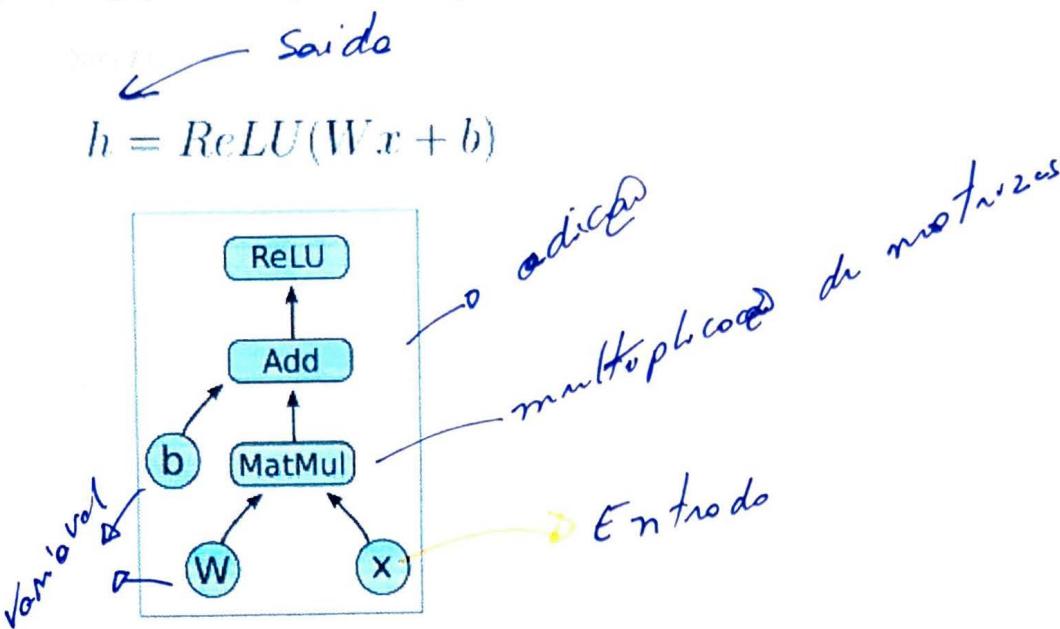


O grafo acima **representa um cálculo de fluxo de dados**; cada nó está em operação **com zero ou mais entradas e zero ou mais saídas**. As arestas do grafo são **tensores que fluem entre os nós**. Os clientes geralmente constroem um grafo computacional usando uma das linguagens frontend suportadas como Python e C ++ e, em seguida, iniciam o grafo em uma sessão a ser executada (Session é uma noção muito importante no TensorFlow, que estudaremos na sequência).

Vamos ver o grafo computacional acima em detalhes. Truncamos o grafo e deixamos a parte acima do nó **ReLU**, que é exatamente o **cálculo  $h = \text{ReLU}(Wx + b)$** .

In [7]:

```
from IPython.display import Image  
Image('imagens/grafico4.png')
```



Podemos ver o grafo como um sistema, que tem **entradas** (os dados  $x$ ), **saída** ( $h$  neste caso), variáveis com estado ( $W$  e  $b$ ) e um monte de operações (matmul, add e ReLU). Deixe-me apresentar-lhe um por um.

- **Placeholders:** para alimentar a entrada para treinar o modelo ou fazer inferência, devemos ter uma porta de entrada para o grafo. Espaços reservados (Placeholders) são nós cujos **valores** são alimentados em **tempo de execução**. Normalmente, queremos alimentar entradas de dados, rótulos e hiper-parâmetros no modelo.
- **Variáveis:** quando treinamos um modelo, usamos variáveis para manter e atualizar parâmetros. Ao contrário de muitos tensores que fluem ao longo das margens do grafo, uma variável é um tipo especial de operação. Na maioria dos modelos de aprendizado de máquina, existem muitos parâmetros que temos que aprender, que são atualizados durante o treinamento. Variáveis são nós com estado que armazenam parâmetros e produzem seus valores atuais de tempos em tempos. Seus estados são mantidos em múltiplas execuções de um grafo. Por exemplo, os valores desses nós não serão atualizados até que uma etapa completa de treinamento usando um mini lote de dados seja concluída.
- **Operações matemáticas:** Neste grafo, existem três tipos de operações matemáticas. A operação **MatMul** multiplica dois valores de matriz; A operação **Add** adiciona elementos e a operação **ReLU** é ativada com a função linear retificada de elementos.

Variáveis devem ser explicitamente inicializadas. Quando criamos uma variável, passamos um tensor como seu valor inicial para o construtor `variable()`. O inicializador pode ser constantes, sequências e valores aleatórios. Neste caso, inicializamos o vetor de polarização  $b$  por constantes que são zeros e inicializamos a matriz de ponderações  $W$  por uniforme aleatório. Observe que todos esses ops exigem que você especifique a forma dos tensores e que a forma se torne automaticamente a forma da variável. Neste caso, um tensor com forma  $(100,)$  e  $W$  é um tensor de classificação com forma  $(784, 100)$ .

Para executar o cálculo, devemos lançar o grafo em um `tf.Session`. O que é uma sessão? Podemos entender uma sessão como um ambiente para executar o grafo. Na verdade, para fazer computação numérica eficiente em Python, normalmente usamos bibliotecas como o NumPy que realizam operações custosas

computacionalmente, como a multiplicação de matrizes, usando código altamente eficiente implementado em outro idioma (C). Infelizmente, ainda há muita sobrecarga voltando para o Python em todas as operações. Essa sobrecarga é particularmente ruim se você quiser fazer cálculos em GPUs ou de maneira distribuída, onde pode haver um alto custo para transferir dados. Para mais detalhes, visite: [www.datascienceacademy.com.br](http://www.datascienceacademy.com.br) (<http://www.datascienceacademy.com.br>).

## Hello World

In [8]:

```
import tensorflow as tf
```

In [9]:

```
tf.__version__
```

```
Out[9]:
```

```
'1.8.0'
```

In [10]:

```
# Cria um tensor
# Esse tensor é adicionado como um nó ao grafo.
hello = tf.constant('Hello, TensorFlow!')
```

In [11]:

```
print(hello)
```

tensor com um conjunto de características

```
Tensor("Const:0", shape=(), dtype=string)
```

In [12]:

```
# Inicia a sessão TensorFlow
sess = tf.Session()
```

In [13]:

```
print(sess)
```

```
<tensorflow.python.client.session.Session object at 0x181e42e978>
```

In [14]:

```
# Executa o Grafo Computacional
print(sess.run(hello))
```

```
b'Hello, TensorFlow!'
```

## Operações Matemáticas com Tensores

## Soma

In [15]:

```
# Constantes  
const_a = tf.constant(5)  
const_b = tf.constant(9)
```

rank zero [ ]

In [16]:

```
print(const_a)
```

```
Tensor("Const_1:0", shape=(), dtype=int32)
```

In [17]:

```
# Soma  
total = const_a + const_b
```

cria o grafo

In [18]:

```
print(total)
```

```
Tensor("add:0", shape=(), dtype=int32)
```

In [19]:

```
# Sessão TF  
with tf.Session() as sess:  
    print("\nA soma do node1 com o node2 é: %f" % sess.run(total))
```

execução do Grafo

sessão é montada  
aberta enquanto  
precisan

A soma do node1 com o node2 é: 14.000000

In [20]:

```
# Criando os nodes no grafo computacional  
node1 = tf.constant(5, dtype=tf.int32)  
node2 = tf.constant(9, dtype=tf.int32)  
node3 = tf.add(node1, node2)
```

você pode determinar o tipo

```
# Cria a sessão TF  
sess = tf.Session()
```

```
# Executa o grafo  
print("\nA soma do node1 com o node2 é:", sess.run(node3))
```

```
# Fecha a sessão  
sess.close()
```

A soma do node1 com o node2 é: 14

## Subtração

In [21]:

```
# Tensores randômicos  
rand_a = tf.random_normal([3], 2.0)  
rand_b = tf.random_uniform([3], 1.0, 4.0)
```

→ matriz até 2.0

Matriz de 1.0 até 4.0

In [22]:

```
print(rand_a)
```

```
Tensor("random_normal:0", shape=(3,), dtype=float32)
```

In [23]:

```
print(rand_b)
```

```
Tensor("random_uniform:0", shape=(3,), dtype=float32)
```

In [24]:

```
# Subtração  
diff = tf.subtract(rand_a, rand_b)
```

In [25]:

```
# Sessão TF  
with tf.Session() as sess:  
    print('\nTensor rand_a: ', sess.run(rand_a))  
    print('\nTensor rand_b: ', sess.run(rand_b))  
    print('\nSubtração entre os 2 tensores é: ', sess.run(diff))
```

```
Tensor rand_a: [2.0543492 1.6592258 2.9290419]
```

```
Tensor rand_b: [2.482773 1.3773217 3.4157996]
```

```
Subtração entre os 2 tensores é: [-0.7432281 -0.912766 -0.14499974]
```

## Divisão

In [26]:

```
node1 = tf.constant(21, dtype=tf.int32)  
node2 = tf.constant(7, dtype=tf.int32)
```

In [27]:

```
div = tf.div(node1, node2)
```

In [28]:

```
# Sessão TF
with tf.Session() as sess:
    print('Divisão entre os 2 tensores é: ', sess.run(div))
```

Divisão entre os 2 tensores é: 3

## Multiplicação

Para aprender a teoria sobre operações com matrizes: <https://pt.khanacademy.org/math/precalculus/precalc-matrices> (<https://pt.khanacademy.org/math/precalculus/precalc-matrices>)

Para aprender a teoria e prática sobre operações com matrizes:

<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>  
(<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>)

In [29]:

```
# Criando tensores
tensor_a = tf.constant([[4., 2.]])
tensor_b = tf.constant([[3., 7.]])
```

In [30]:

```
print(tensor_a)
```

1 linha e 2 colunas

$$\begin{array}{|c|c|} \hline 4 & 2 \\ \hline \end{array} \times \Rightarrow \begin{array}{|c|c|} \hline 12 & 14 \\ \hline \end{array}$$
  
$$\begin{array}{|c|} \hline 3 \\ \hline 7 \\ \hline \end{array}$$
  
$$\begin{array}{|c|c|} \hline 12 & 14 \\ \hline 28 & 14 \\ \hline \end{array}$$

Tensor("Const\_7:0", shape=(1, 2), dtype=float32)

In [31]:

```
print(tensor_b)
```

2 linhas e 1 coluna

Tensor("Const\_8:0", shape=(2, 1), dtype=float32)

In [32]:

```
# Multiplicação
# tf.multiply(X, Y) executa multiplicação element-wise
# https://www.tensorflow.org/api_docs/python/tf/multiply
prod = tf.multiply(tensor_a, tensor_b)
```

X      Y

Retorno no mesmo formato do  
Primeiro elemento  $\Rightarrow X$

In [33]:

```
# Executa a sessão
with tf.Session() as sess:
    print("\nTensor_a: \n", sess.run(tensor_a))
    print("\nTensor_b: \n", sess.run(tensor_b))
    print("\nProduto Element-wise Entre os Tensores: \n", sess.run(prod))
```

tensor\_a:  
[[4. 2.]

tensor\_b:  
[[3.]  
[7.]]

Produto Element-wise Entre os Tensores:

```
[[12. 6.]
 [28. 14.]]
```

In [34]:

```
# Outro exemplo de Multiplicação de Matrizes
mat_a = tf.constant([[2, 3], [9, 2], [4, 5]])
mat_b = tf.constant([[6, 4, 5], [3, 7, 2]])
```

In [35]:

```
print(mat_a)
Tensor("Const_9:0", shape=(3, 2), dtype=int32)
```

In [36]:

```
print(mat_b)
Tensor("Const_10:0", shape=(2, 3), dtype=int32)
```

In [37]:

```
# Multiplicação
# tf.matmul(X, Y) executa multiplicação entre matrizes
# https://www.tensorflow.org/api_docs/python/tf/matmul
mat_prod = tf.matmul(mat_a, mat_b)
```

tem mais funções que o matmul

In [38]:

```
# Executa a sessão
with tf.Session() as sess:
    print('\nTensor mat_a:\n', sess.run(mat_a))
    print('\nTensor mat_b:\n', sess.run(mat_b))
    print('\nProduto Element-wise Entre os Tensores (Matrizes):\n', sess.run(mat_prod))
```

Tensor mat\_a:

[[2 3]  
[9 2] 3  
[4 5]]

Tensor mat\_b:

[[6 4 5]  
[3 7 2]]  
3

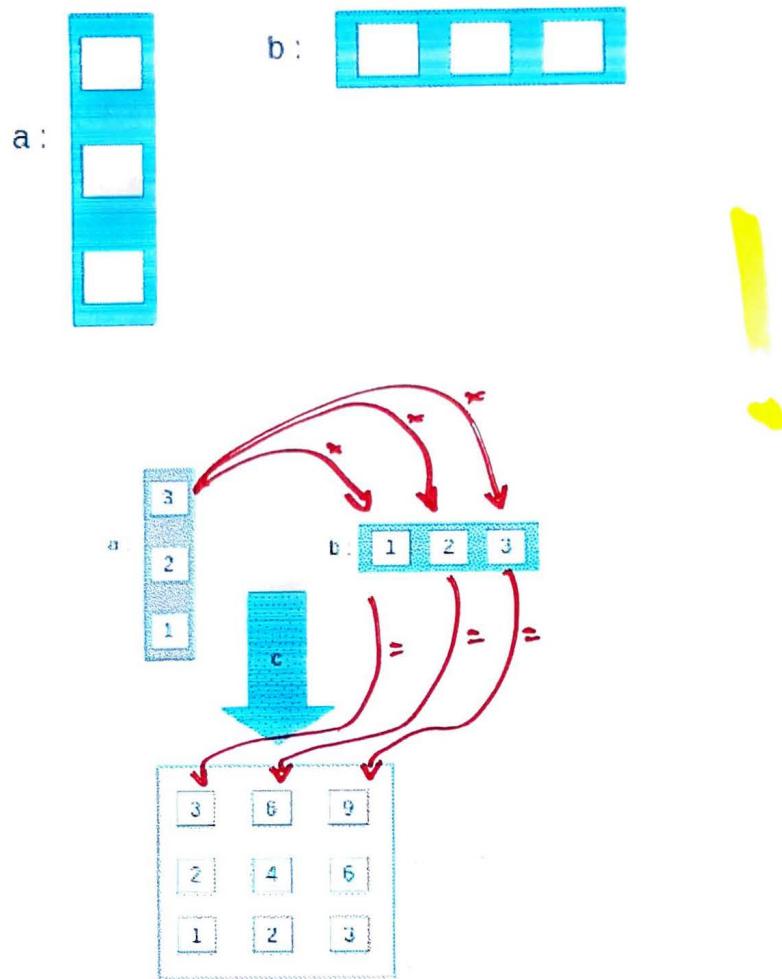
Produto Element-wise Entre os Tensores (Matrizes):

[[21 29 16]  
[60 50 49] 3  
[39 51 30]]

3

In [39]:

```
from IPython.display import Image  
Image('imagens/matrizes.png')
```



## Usando Variáveis

O TensorFlow também possui nódes variáveis que podem conter dados variáveis. Elas são usados principalmente para manter e atualizar parâmetros de um modelo de treinamento.

Variáveis são buffers na memória contendo tensores. Elas devem ser inicializados e podem ser salvas durante e após o treinamento. Você pode restaurar os valores salvos posteriormente para exercitar ou analisar o modelo.

Uma diferença importante a notar entre uma constante e uma variável é:

O valor de uma constante é armazenado no grafo e seu valor é replicado onde o grafo é carregado. Uma variável é armazenada separadamente e pode estar em um servidor de parâmetros.

In [40]:

```
# Import
import tensorflow as tf

# Criando um node no grafo computacional
node = tf.Variable(tf.zeros([2,2])) → função zeros = cria uma matriz de zeros

# Executando o grafo computacional
with tf.Session() as sess:
    # Inicializando as variáveis
    sess.run(tf.global_variables_initializer())

    # Avaliando o node
    print("\nTensor Original:\n", sess.run(node))

    # Adição element-wise no tensor
    node = node.assign(node + tf.ones([2,2])) → Somar outro node

    # Avaliando o node novamente
    print("\nTensor depois da adição:\n", sess.run(node))
```

$\begin{matrix} 0,0 & + 1,1 \\ 0,0 & 1,1 \end{matrix} = \begin{matrix} 1,1 \\ 1,1 \end{matrix}$

O tensor é uma variável

Tensor Original:

```
[[0. 0.]
 [0. 0.]]
```

Tensor depois da adição:

```
[[1. 1.]
 [1. 1.]]
```

## Usando Placeholders

→ Espaços reservados

Quando você não sabe quais dados alimentar no tensor

Um grafo pode ser parametrizado para aceitar dados externos e podemos reservar áreas conhecidas como espaços reservados (Placeholders). Um espaço reservado é uma promessa para fornecer um valor mais tarde.

Ao avaliar o grafo envolvendo nós de espaço reservado, um parâmetro feed\_dict é passado para a sessão.

In [41]:

```
# Import
import tensorflow as tf

# Criando nodes no Grafo Computacional
a = tf.placeholder(tf.int32, shape=(3,1))
b = tf.placeholder(tf.int32, shape=(1,3))
c = tf.matmul(a,b)

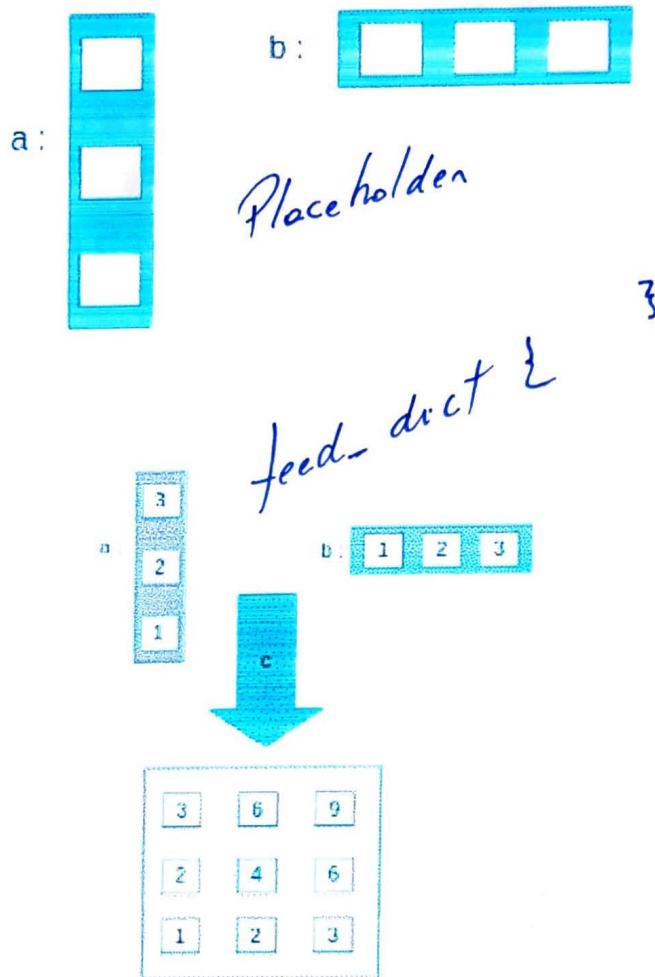
# Executando o Grafo Computacional
with tf.Session() as sess:
    print(sess.run(c, feed_dict={a:[[3],[2],[1]], b:[[1,2,3]]}))
```

```
[[3 6 9]
 [2 4 6]
 [1 2 3]]
```

→ Recebe no memória uma área tipo int32 com shape(3,1)

In [42]:

```
from IPython.display import Image  
Image('images/ndarrays.png')
```



Conheça a Formação Inteligência Artificial, um programa completo, 100% online e 100% em português, com 402 horas em 9 cursos de nível intermediário/avançado, que vão ajudá-lo a se tornar um dos profissionais mais cobiçados do mercado de tecnologia. Clique no link abaixo, faça sua inscrição, comece hoje mesmo e aumente sua empregabilidade:

<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>  
(<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>)

**Fim**

**Obrigado - Data Science Academy - [facebook.com/dsacademybr](https://facebook.com/dsacademybr)**  
(<http://facebook.com/dsacademybr>)

# Data Science Academy - Python Fundamentos - Capítulo 11

Download: <http://github.com/dsacademybr>  
<http://github.com/dsacademybr>)

## Prevendo a Ocorrência de Diabetes

### Conjunto de Dados do Repositório de Machine Learning da UCI / Kaggle

<https://www.kaggle.com/uciml/pima-indians-diabetes-database/data> (<https://www.kaggle.com/uciml/pima-indians-diabetes-database/data>)

In [11]:

```
# Importando os módulos
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline → uso apenas no jupyter notebook.
```

In [12]:

```
# Carregando o dataset
df = pd.read_csv("pima-data.csv")
```

In [13]:

```
# Verificando o formato dos dados
df.shape
```

Out[13]:

(768, 10)

↓

observações

Variáveis

In [14]:

```
# Verificando as primeiras linhas do dataset  
df.head(5)
```

Output:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	dia
0	6	148	72	35	0	33.6	0.627	50	1.3790	
1	1	85	66	29	0	26.6	0.351	31	1.1426	
2	8	183	64	0	0	23.3	0.672	32	0.0000	
3	1	89	66	23	94	28.1	0.167	21	0.9062	
4	0	137	40	35	168	43.1	2.288	33	1.3790	

In [15]:

```
# Verificando as últimas linhas do dataset  
df.tail(5)
```

Output:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	c
763	10	101	76	48	180	32.9	0.171	63	1.8912	
764	2	122	70	27	0	36.8	0.340	27	1.0638	
765	5	121	72	23	112	26.2	0.245	30	0.9062	
766	1	126	60	0	0	30.1	0.349	47	0.0000	
767	1	93	70	31	0	30.4	0.315	23	1.2214	

In [16]:

```
# Verificando se existem valores nulos  
df.isnull().values.any()
```

Output:

False

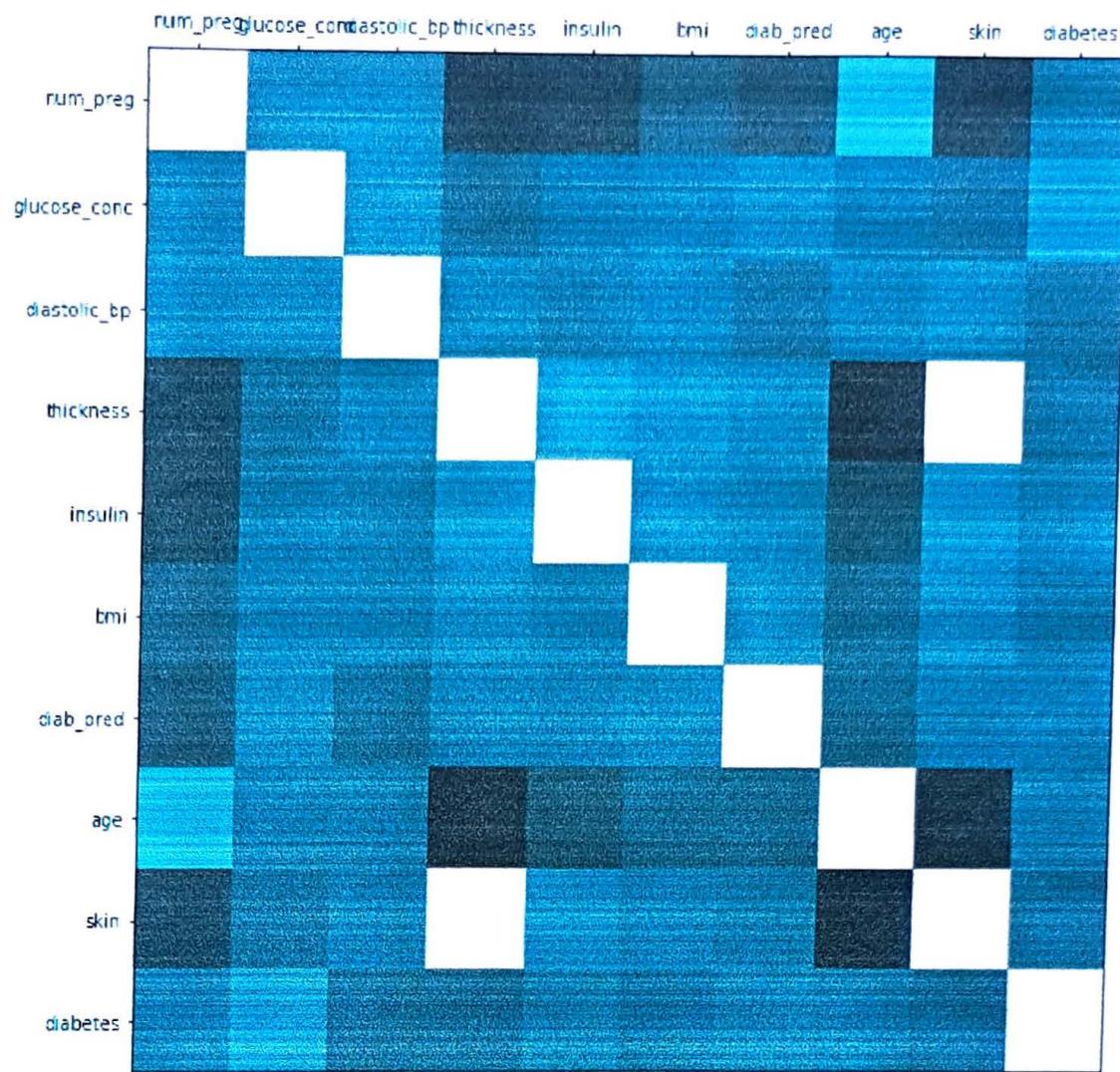
Significa que não há valores Nulos

In [17]:

```
# Identificando a correlação entre as variáveis  
# Correlação não implica causalidade  
def plot_corr(df, size=10):  
    corr = df.corr()  
    fig, ax = plt.subplots(figsize = (size, size))  
    ax.matshow(corr)  
    plt.xticks(range(len(corr.columns)), corr.columns)  
    plt.yticks(range(len(corr.columns)), corr.columns)
```

In [18]:

```
# Criando o gráfico  
plot_corr(df)
```



In [20]:

```
# Visualizando a correlação em tabela  
# Coeficiente de correlação:  
# +1 = forte correlação positiva  
# 0 = não há correlação  
# -1 = forte correlação negativa  
df.corr().transpose()
```

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred
num_preg	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-0.033523
glucose_conc	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	0.137331
diastolic_bp	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	0.041261
thickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183921
insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	0.185071
bmi	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140641
diab_pred	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	1.000000
age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	0.033561
skin	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	0.183921
diabetes	0.221898	0.466581	0.065068	0.074752	0.130548	0.292695	0.173841

In [21]:

```
df.describe()
```

Out[21]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred
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	0.471876
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	0.331329
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	0.243750
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	0.372500
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	0.626250
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000

In [22]:

```
# Definindo as classes  
diabetes_map = {True : 1, False : 0}
```

modificando valores True e False  
para 1 e 0

In [23]:

```
# Aplicando o mapeamento ao dataset  
df['diabetes'] = df['diabetes'].map(diabetes_map)
```

In [24]:

```
# Verificando as primeiras linhas do dataset  
df.head(5)
```

Out[24]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	dia	diabetes
0	6	148	72	35	0	33.6	0.627	50	1.3790	1	1
1	1	85	66	29	0	26.6	0.351	31	1.1426	0	0
2	8	183	64	0	0	23.3	0.672	32	0.0000	1	1
3	1	89	66	23	94	28.1	0.167	21	0.9062	0	0
4	0	137	40	35	168	43.1	2.288	33	1.3790	1	1

In [25]:

```
# Verificando como os dados estão distribuídos  
num_true = len(df.loc[df['diabetes'] == True])  
num_false = len(df.loc[df['diabetes'] == False])  
print("Número de Casos Verdadeiros: {} ({:.2f}%)".format(num_true, (num_true / num_true) * 100))  
print("Número de Casos Falsos : {} ({:.2f}%)".format(num_false, (num_false / num_true) * 100))
```

Número de Casos Verdadeiros: 268 (34.90%) desenvolvem diabetes  
Número de Casos Falsos : 500 (65.10%) não desenvolvem

O ideal é ter um balanço entre 50% e 50%

## Spliting

70% para dados de treino e 30% para dados de teste

In [30]:

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

In [31]:

```
# Seleção de variáveis preditoras (Feature Selection)  
atributos = ['num_preg', 'glucose_conc', 'diastolic_bp', 'thickness', 'insulin', 'bmi', 'diab_pred', 'age', 'skin', 'dia']
```

In [32]:

```
# Variável a ser prevista  
atrib_prev = ['diabetes']
```

In [33]:

```
# Criando objetos  
X = df[atributos].values  
Y = df[atrib_prev].values
```

In [34]:

X

```
array([[ 6. , 148. , 72. , ..., 33.6 , 0.627 , 50. ],
       [ 1. , 85. , 66. , ..., 26.6 , 0.351 , 31. ],
       [ 8. , 183. , 64. , ..., 23.3 , 0.672 , 32. ],
       ...,
       [ 5. , 121. , 72. , ..., 26.2 , 0.245 , 30. ],
       [ 1. , 126. , 60. , ..., 30.1 , 0.349 , 47. ],
       [ 1. , 93. , 70. , ..., 30.4 , 0.315 , 23. ]])
```

conjugantes de  
variables  
preditivas

In [35]:

Y

*Varietà*  
*tangere*

In [36]:

```
# Definindo a taxa de split  
split_test_size = 0.30
```

tamanhos do conjunto de teste

In [37]:

```
# Criando dados de treino e de teste
X_treino, X_teste, Y_treino, Y_teste = train_test_split(X, Y, test_size = split_test_size,
                                                       random_state = random_state)
A
# Retorna 4 vetores
30%
```

In [38]:

```
# Imprimindo os resultados
print("({0:.2f}%) nos dados de treino".format((len(X_treino)/len(df.index)) * 100))
print("({0:.2f}%) nos dados de teste".format((len(X_teste)/len(df.index)) * 100))
```

69.92% nos dados de treino  
30.08% nos dados de teste

In [39]:

X\_treino

array[:]

```
array([[ 1.    ,  95.    ,  60.    , ..., 23.9   ,  0.26   , 22.    ],
       [ 5.    , 105.    ,  72.    , ..., 36.9   ,  0.159  , 28.    ],
       [ 0.    , 135.    ,  68.    , ..., 42.3   ,  0.365  , 24.    ],
       ...,
       [ 10.   , 101.   ,  86.   , ..., 45.6   ,  1.136  , 38.    ],
       [ 0.   , 141.   ,  0.    , ..., 42.4   ,  0.205  , 29.    ],
       [ 0.   , 125.   ,  96.   , ..., 22.5   ,  0.262  , 21.    ]])
```

conjunto de  
Xtreino

## Verificando o Split

In [40]:

```
print("Original True : {0} ({1:.2f}%)".format(len(df.loc[df['diabetes'] == 1]),
                                              (len(df.loc[df['diabetes'] == 1])/len(df.index)) * 100))
print("Original False : {0} ({1:.2f}%)".format(len(df.loc[df['diabetes'] == 0]),
                                              (len(df.loc[df['diabetes'] == 0])/len(df.index)) * 100))
print("")
print("Training True : {0} ({1:.2f}%)".format(len(Y_treino[Y_treino[:] == 1]),
                                              (len(Y_treino[Y_treino[:] == 1])/len(Y_treino)) * 100))
print("Training False : {0} ({1:.2f}%)".format(len(Y_treino[Y_treino[:] == 0]),
                                              (len(Y_treino[Y_treino[:] == 0])/len(Y_treino)) * 100))
print("")
print("Test True : {0} ({1:.2f}%)".format(len(Y_teste[Y_teste[:] == 1]),
                                             (len(Y_teste[Y_teste[:] == 1])/len(Y_teste)) * 100))
print("Test False : {0} ({1:.2f}%)".format(len(Y_teste[Y_teste[:] == 0]),
                                             (len(Y_teste[Y_teste[:] == 0])/len(Y_teste)) * 100))
```

Original True : 268 (34.90%)      diabete      768 > 100%  
Original False : 500 (65.10%)

Training True : 188 (35.01%)      537 = 70%  
Training False : 349 (64.99%)

Test True : 80 (34.63%)      231 = 30%  
Test False : 151 (65.37%)

## Valores Missing Ocultos

In [41]:

```
# Verificando se existem valores nulos  
df.isnull().values.any()
```

Out[41]:

False

In [42]:

```
df.head(5)
```

Out[42]:

	num_preg	glucose_conc	diastolic_bp	thickness	insulin	bmi	diab_pred	age	skin	dia
0	6	148	72	35	0	33.6	0.627	50	1.3790	
1	1	85	66	29	0	26.6	0.351	31	1.1426	
2	8	183	64	0	0	23.3	0.672	32	0.0000	
3	1	89	66	23	94	28.1	0.167	21	0.9062	
4	0	137	40	35	168	43.1	2.288	33	1.3790	

In [43]:

```
print("# Linhas no dataframe {}".format(len(df)))  
print("# Linhas missing glucose_conc: {}".format(len(df.loc[df['glucose_conc'] == 0]))))  
print("# Linhas missing diastolic_bp: {}".format(len(df.loc[df['diastolic_bp'] == 0]))))  
print("# Linhas missing thickness: {}".format(len(df.loc[df['thickness'] == 0]))))  
print("# Linhas missing insulin: {}".format(len(df.loc[df['insulin'] == 0]))))  
print("# Linhas missing bmi: {}".format(len(df.loc[df['bmi'] == 0]))))  
print("# Linhas missing age: {}".format(len(df.loc[df['age'] == 0]))))  
  
# Linhas no dataframe 768  
# Linhas missing glucose_conc: 5  
# Linhas missing diastolic_bp: 35  
# Linhas missing thickness: 227  
# Linhas missing insulin: 374  
# Linhas missing bmi: 11  
# Linhas missing age: 0
```

Alterar os valores missing  
ocultos para média

## Tratando Dados Missing - Impute

Substituindo os valores iguais a zero, pela média dos dados

In [44]:

```
from sklearn.preprocessing import Imputer
```

In [45]:

```
# Criando objeto
preenche_0 = Imputer(missing_values = 0, strategy = "mean", axis = 0)

# Substituindo os valores iguais a zero, pela média dos dados
X_treino = preenche_0.fit_transform(X_treino)
X_teste = preenche_0.fit_transform(X_teste)

c:\users\gival\appdata\local\programs\python\python36\lib\site-packages\sklearn\utils\deprecation.py:58: DeprecationWarning: Class Imputer is deprecated; Imputer was deprecated in version 0.20 and will be removed in 0.22. Import impute.SimpleImputer from sklearn instead.
warnings.warn(msg, category=DeprecationWarning)
```

In [46]:

```
X_treino
```

Out[46]:

```
array([[ 1.        ,  95.        ,  60.        , ... , 23.9        ,
       0.26      ,  22.        , ... ],
      [ 5.        , 105.        ,  72.        , ... , 36.9        ,
       0.159     ,  28.        , ... ],
      [ 4.34056399, 135.        ,  68.        , ... , 42.3        ,
       0.365     ,  24.        , ... ],
      ...,
      [ 10.        , 101.        ,  86.        , ... , 45.6        ,
       1.136     ,  38.        , ... ],
      [ 4.34056399, 141.        , 72.24131274, ... , 42.4        ,
       0.205     ,  29.        , ... ],
      [ 4.34056399, 125.        ,  96.        , ... , 22.5        ,
       0.262     ,  21.        , ... ]])
```

**50 a 80% do tempo de trabalho de um Cientista de Dados é usado na preparação dos dados.**

## Construindo e treinando o modelo

In [47]:

```
# Utilizando um classificador Naive Bayes
from sklearn.naive_bayes import GaussianNB
```

In [48]:

```
# Criando o modelo preditivo
modelo_v1 = GaussianNB()
```

*teorema do Bayes  
funcão probabilística*

In [49]:

```
# Treinando o modelo  
modelo_v1.fit(X_treino, Y_treino.ravel())
```

função para ajustar o shape

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

## Verificando a exatidão no modelo nos dados de treino

In [50]:

```
from sklearn import metrics
```

In [51]:

```
nb_predict_train = modelo_v1.predict(X_treino)
```

Passo apenas os VARIÁVEIS  
de treino

In [52]:

```
print("Exatidão (Accuracy): {:.4f}".format(metrics.accuracy_score(Y_treino, nb_predict_train)))  
print()
```

Exatidão (Accuracy): 0.7542

de cada 100 previsões  
ele acerta 75%

✓  
é o que eu  
esperava com Sórdida  
Previsão  
que o  
modelo fez

## Verificando a exatidão no modelo nos dados de teste

In [53]:

```
nb_predict_test = modelo_v1.predict(X_teste)
```

In [54]:

```
print("Exatidão (Accuracy): {:.4f}".format(metrics.accuracy_score(Y_teste, nb_predict_test)))  
print()
```

Exatidão (Accuracy): 0.7359

Foi treinado com os dados  
de treino mas ~~nunca~~ nunca  
viu os dados de test

## Métricas

In [55]:

```
from IPython.display import Image  
Image('ConfusionMatrix.jpg')
```

Taxas de erros dos modelos

		Predicted		
		+	-	
Actual	+	TP Type II error	FN	Sensitivity (recall) $TP/( \bullet + \bullet )$
	-	FP Type I error	TN	False positive rate $FP/( \bullet + \bullet )$
Precision		False omission rate $TP/( \bullet + \bullet )$		Accuracy $( TP + TN ) / ( \bullet + \bullet )$
FDR		Negative predictive value $FP/( \bullet + \bullet )$		$F_1$ score $2TP / ( 2TP + FP + FN )$

In [56]:

```
# Criando uma Confusion Matrix  
print("Confusion Matrix")  
  
print("{0}".format(metrics.confusion_matrix(Y_teste, nb_predict_test, labels = [1, 0])))  
print()  
  
print("Classification Report")  
print(metrics.classification_report(Y_teste, nb_predict_test, labels = [1, 0]))
```

Confusion Matrix

```
[[ 52  28]  
 [ 33 118]]
```

Classification Report

	precision	recall	f1-score	support
1	0.61	0.65	0.63	80
0	0.81	0.78	0.79	151
micro avg	0.74	0.74	0.74	231
macro avg	0.71	0.72	0.71	231
weighted avg	0.74	0.74	0.74	231

## Otimizando o modelo com RandomForest

In [57]:

```
from sklearn.ensemble import RandomForestClassifier
```

In [58]:

```
modelo_v2 = RandomForestClassifier(random_state = 42)
modelo_v2.fit(X_treino, Y_treino.ravel()) → treina o modelo
c:\users\gival\appdata\local\programs\python\python36\lib\site-packages\skle
arn\ensemble\forest.py:246: FutureWarning: The default value of n_estimators
will change from 10 in version 0.20 to 100 in 0.22.
"10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [58]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                      oob_score=False, random_state=42, verbose=0, warm_start=False)
```

In [59]:

*Este algoritmo de Machine Learning tem um conjunto diferente de parâmetros*

```
# Verificando os dados de treino
```

```
rf_predict_train = modelo_v2.predict(X_treino)
print("Exatidão (Accuracy): {:.4f}".format(metrics.accuracy_score(Y_treino, rf_predict_tr))
```

Exatidão (Accuracy): 0.9870

*Não da para considerar isso ocorreis em treino porque São dados que o modelo já conhece*

In [60]:

```
# Verificando nos dados de teste
rf_predict_test = modelo_v2.predict(X_teste)
print("Exatidão (Accuracy): {:.4f}".format(metrics.accuracy_score(Y_teste, rf_predict_te))
print()
```

Exatidão (Accuracy): 0.7100

*a importância de testar com dados de treino*

In [61]:

```
print("Confusion Matrix")
print("{}\n".format(metrics.confusion_matrix(Y_teste, rf_predict_test, labels = [1, 0])))
print("Classification Report")
print(metrics.classification_report(Y_teste, rf_predict_test, labels = [1, 0]))
```

Confusion Matrix

43	37
30	121

Classification Report

	precision	recall	f1-score	support
1	0.59	0.54	0.56	80
0	0.77	0.80	0.78	151
micro avg	0.71	0.71	0.71	231
macro avg	0.68	0.67	0.67	231
weighted avg	0.70	0.71	0.71	231

Supervised  
Classification

Regression

## Regressão Logística

In [62]:

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

In [63]:

```
# Terceira versão do modelo usando Regressão Logística
modelo_v3 = LogisticRegression(C = 0.7, random_state = 42)
modelo_v3.fit(X_treino, Y_treino.ravel())
lr_predict_test = modelo_v3.predict(X_teste)
```

```
c:\users\gival\appdata\local\programs\python\python36\lib\site-packages\sklearn\linear_model\logistic.py:433: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)
```

In [64]:

```
print("Exatidão (Accuracy) = 0.7446".format(metrics.accuracy_score(Y_teste, lr_predict_test)))
print()
print("Classification Report")
print(metrics.classification_report(Y_teste, lr_predict_test, labels = [1, 0]))
```

Exatidão (Accuracy): 0.7446

Classification Report

	precision	recall	f1-score	support
1	0.66	0.55	0.60	80
0	0.78	0.85	0.81	151
micro avg	0.74	0.74	0.74	231
macro avg	0.72	0.70	0.71	231
weighted avg	0.74	0.74	0.74	231

In [65]:

```
### Resumindo
## Exatidão nos dados de teste

# Modelo usando algoritmo Naive Bayes      = 0.7359
# Modelo usando algoritmo Random Forest     = 0.7100
# Modelo usando algoritmo Regressão Logística = 0.7446
```

## Fazendo Previsões Com o Modelo Treinado

In [66]:

```
import pickle
```

In [67]:

```
# Salvando o modelo para usar mais tarde
filename = 'modelo_treinado_v3.sav' → nome
pickle.dump(modelo_v3, open(filename, 'wb'))
```

Salvando modelo

In [68]:

X\_teste

```
array([[6.0000000e+00, 9.8000000e+01, 5.8000000e+01, ...,
       3.4000000e+01, 4.3000000e-01, 4.3000000e+01],
      [2.0000000e+00, 1.1200000e+02, 7.5000000e+01, ...,
       3.5700000e+01, 1.4800000e-01, 2.1000000e+01],
      [2.0000000e+00, 1.0800000e+02, 6.4000000e+01, ...,
       3.0800000e+01, 1.5800000e-01, 2.1000000e+01],
      ...,
      [4.85714286e+00, 1.2700000e+02, 8.0000000e+01, ...,
       3.6300000e+01, 8.0400000e-01, 2.3000000e+01],
      [6.0000000e+00, 1.0500000e+02, 7.0000000e+01, ...,
       3.0800000e+01, 1.2200000e-01, 3.7000000e+01],
      [5.0000000e+00, 7.7000000e+01, 8.2000000e+01, ...,
       3.5800000e+01, 1.5600000e-01, 3.5000000e+01]])
```

Vamos usar  $L = \text{o modelo-treinado V3.}$

In [69]:

```
# Carregando o modelo e fazendo previsão com novos conjuntos de dados
# (X_teste, Y_teste devem ser novos conjuntos de dados preparados com o procedimento de lim
loaded_model = pickle.load(open(filename, "rb")) # obvendo o modelo
resultado1 = loaded_model.predict(X_teste[15].reshape(1, -1)) # treinado
resultado2 = loaded_model.predict(X_teste[18].reshape(1, -1))
print(resultado1)
print(resultado2)
```

[0]  
[1]

Fim

Obrigado - Data Science Academy - [facebook.com/dsacademybr](http://facebook.com/dsacademybr)  
<http://facebook.com/dsacademybr>

# Data Science Academy - Python Fundamentos - Capítulo 12

Download: <http://github.com/dsacademybr>  
<http://github.com/dsacademybr>)

## Detecção de Emoções em Imagens com Inteligência Artificial

### Teste

In [1]:

```
from scipy import misc
import numpy as np
import matplotlib.cm as cm
import tensorflow as tf
import os, sys, inspect
from datetime import datetime
from matplotlib import pyplot as plt
import matplotlib.image as mpimg
from modulos import utils
from modulos.utils import testResult

from tensorflow.python.framework import ops
from sklearn.metrics.classification import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
import warnings
%matplotlib inline
```

C:\Users\gival\Anaconda3\lib\site-packages\h5py\\_init\_.py:36: FutureWarning: Conversion of the second argument of issubdtype from `float` to `np.float` is deprecated. In future, it will be treated as `np.float64 == np.dtype(float).type`.

```
    from ._conv import register_converters as _register_converters
```

In [2]:

```
warnings.filterwarnings("ignore")
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
ops.reset_default_graph()
```

Ponto de  
Ponto final  
grafia

In [3]:

```
emotion = {0:'anger',
           1:'disgust',
           2:'fear',
           3:'happy',
           4:'sad',
           5:'surprise',
           6:'neutral'}
```

Amo's serial as  
emoções

In [4]:

```
def rgb2gray(rgb):
    return np.dot(rgb[...,:3], [0.299, 0.587, 0.114])
```

converte imagem  
coloreando p/ escala de  
cinza

In [5]:

```
sess = tf.InteractiveSession()
```

Cria sessão

In [6]:

```
new_saver = tf.train.import_meta_graph('modelo/model.ckpt-900.meta')
new_saver.restore(sess, 'modelo/model.ckpt-900')
tf.get_default_graph().as_graph_def()
```

```
        size: 5
    }
    dim {
        size: 5
    }
    dim {
        size: 1
    }
    dim {
        size: 32
    }
}
attr {
    key: "shared_name"
    value {
        s: ""
    }
}
```

Restaura o modelo  
que já foi treinado

In [7]:

```
x = sess.graph.get_tensor_by_name("input:0")
y_conv = sess.graph.get_tensor_by_name("output:0")
```

p/ círculo

In [ ]:

```
img = mpimg.imread('images teste/image05.jpg')
gray = rgb2gray(img)
plt.imshow(gray, cmap = plt.get_cmap('gray'))
plt.show()
```

converte imagem para  
e plota no

In [21]:

```
image_0 = np.resize(gray,(1,48,48,1))
tResult = testResult()
num_evaluations = 50
```

converte imagem p/ o mesmo  
tamanho das da treinamento

In [22]:

```
for i in range(0, num_evaluations):
    result = sess.run(y_conv, feed_dict={x:image_0})
    label = sess.run(tf.argmax(result, 1))
    label = label[0]
    label = int(label)
    tResult.evaluate(label)
tResult.display_result(num_evaluations)
```

```
anger = 0.0%
disgust = 0.0%
fear = 94.0%
happy = 6.0%
sad = 0.0%
surprise = 0.0%
neutral = 0.0%
```

Para adquirir conhecimento técnico sólido e especializado em Deep Learning, Visão Computacional, Processamento de Linguagem Natural e outros temas relacionados à Inteligência Artificial, confira nosso programa completo: [Formação Inteligência Artificial](https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial) (<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>).

## Fim

Obrigado - Data Science Academy - [facebook.com/dsacademybr](https://facebook.com/dsacademybr)  
([http://facebook.com/dsacademybr](https://facebook.com/dsacademybr))

# Data Science Academy - Python Fundamentos - Capítulo 12

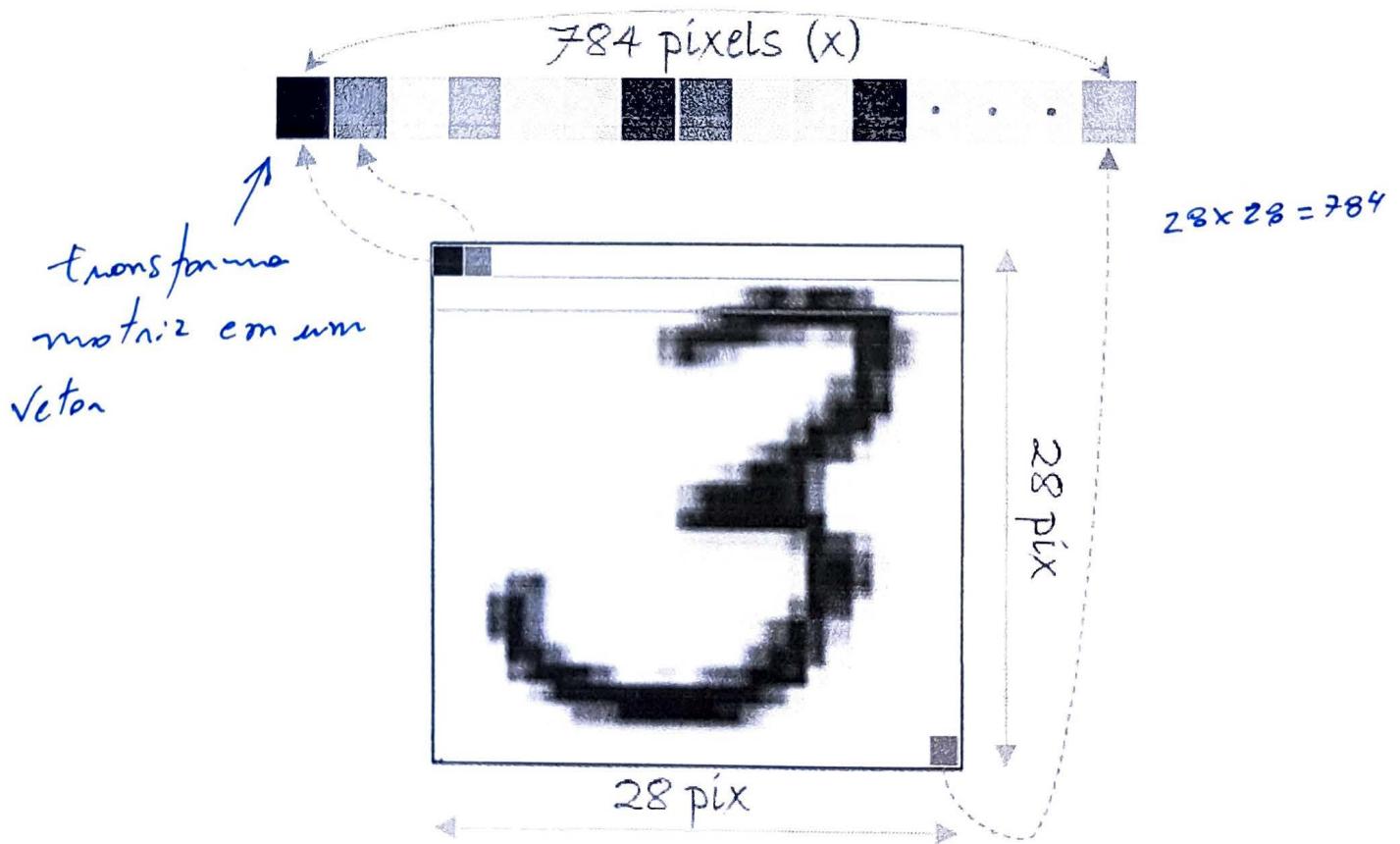
Download: <http://github.com/dsacademybr>  
[\(http://github.com/dsacademybr\)](http://github.com/dsacademybr)

## Detecção de Emoções em Imagens com Inteligência Artificial

<https://www.kaggle.com/c/facial-keypoints-detector> (<https://www.kaggle.com/c/facial-keypoints-detector>)

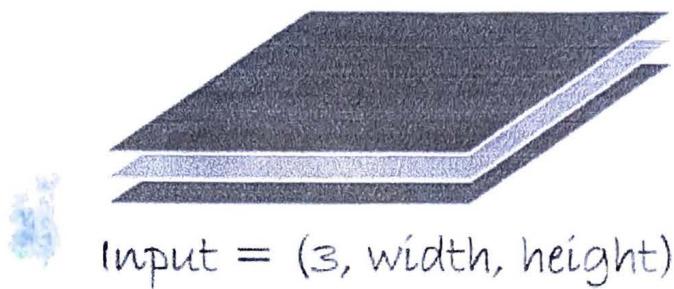
## Redes Neurais Convolucionais

Em redes neurais convolucionais, os dados de entrada são muitas vezes moldados como uma matriz 3D (número de canais, largura da imagem, altura), que preserva a relação espacial entre os pixels. Na figura abaixo, a imagem 3 é um único canal (tons de cinza) de dados, portanto, a dimensão de entrada é especificada como uma tupla (1, largura da imagem, altura da imagem).



Imagens de cor de cena natural são frequentemente apresentadas como canais de cor Vermelho-Verde-Azul (RGB). A dimensão de entrada dessas imagens é especificada como uma tupla (3, largura da imagem, altura da imagem). Se houver dados de entrada RGB como uma varredura volumétrica com largura de volume, altura

de volume e profundidade de volume representando os 3 eixos, o formato de dados de entrada será especificado por uma tupla de 4 valores (3, largura de volume, altura de volume, profundidade de volume). Desta forma, podemos especificar as imagens de entrada em espaço arbitrário de dimensão superior.

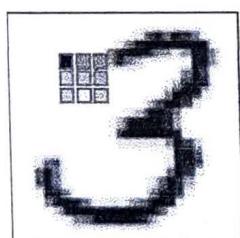


CNN é uma rede feedforward composta de diversas camadas de tal forma que a saída de uma camada torna-se a entrada para a próxima camada (semelhante ao MLP). Em MLP, todos os pares possíveis de pixels de entrada são conectados aos nós de saída com cada par tendo um peso, conduzindo assim a uma explosão combinatória de parâmetros a serem aprendidos e também aumentando a possibilidade de overfitting. As camadas de convolução aproveitam a disposição espacial dos pixels e aprendem vários filtros que reduzem significativamente a quantidade de parâmetros na rede. O tamanho do filtro é um parâmetro da camada de convolução.

Nesta seção, apresentamos os fundamentos das operações de convolução.

## Camada de Convolução

Uma camada de convolução é um conjunto de filtros. Cada filtro é definido por uma matriz de peso ( $W$ ) e bias ( $b$ ).



$$\begin{array}{c} \text{filters} \\ \hline \text{---} \end{array} \quad \begin{array}{ccc} \text{---} & \text{---} & \text{---} \end{array} + \begin{array}{c} \text{---} \\ b \end{array} \quad \begin{array}{ccc} \text{---} & \text{---} & \text{---} \end{array} + \begin{array}{c} \text{---} \\ b \end{array} \quad \dots \quad \begin{array}{ccc} \text{---} & \text{---} & \text{---} \end{array} + \begin{array}{c} \text{---} \\ b \end{array}$$

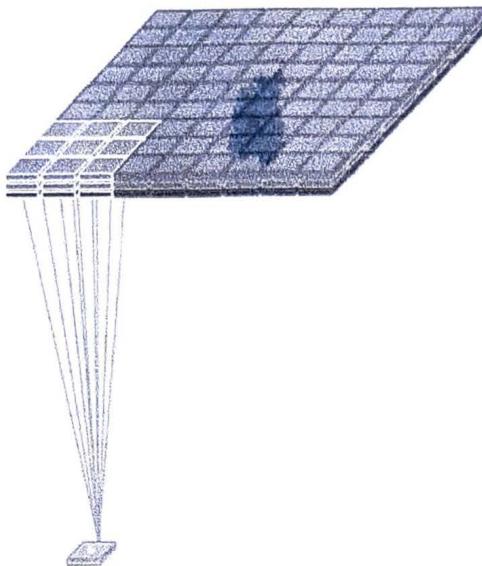
$$\vec{z} = W \vec{x}^T + b \qquad \vec{z} = W \vec{x}^T + b \qquad \dots \qquad \vec{z} = W \vec{x}^T + b$$

Estes filtros são varridos através da imagem que realiza o dot product entre os pesos e o valor de entrada correspondente ( $\vec{x}^T$ ). O valor de bias é adicionado à saída do dot product e a soma resultante é opcionalmente mapeada através de uma função de ativação. Esse processo é ilustrado na seguinte animação.

In [1]:

```
from IPython.display import display, Image  
Image(url="https://www.cntk.ai/jup/cntk103d_conv2d_final.gif", width= 300)
```

Out[1]:



As camadas de convolução incorporam as seguintes características-chave:

- Em vez de estar totalmente conectado a todos os pares de nós de entrada e saída, cada nó de convolução é **conectado localmente** a um subconjunto de nós de entrada localizados em uma região de entrada menor, também chamada de campo receptivo (RF). A figura acima ilustra pequenas regiões  $3 \times 3$  na imagem como a região RF. No caso de uma imagem RGB, haveria três dessas  $3 \times 3$  regiões, uma de cada um dos 3 canais de cor.
- Em vez de ter um único conjunto de pesos (como em uma camada Densa), camadas convolucionais têm vários conjuntos (mostrado na figura com várias cores), chamado **filtros**. Cada filtro detecta características dentro de cada RF possível na imagem de entrada. A saída da convolução é um conjunto de sub-camadas  $n$  (mostradas na animação abaixo) onde  $n$  é o número de filtros (consulte a figura acima).
- Dentro de uma subcamada, em vez de cada nó ter seu próprio conjunto de pesos, um único conjunto de **pesos compartilhados** são usados por todos os nós nessa subcamada. Isso reduz o número de parâmetros a serem aprendidos. Isso também abre a porta para vários aspectos da aprendizagem profunda que permitiu a construção de soluções muito práticas: -- Manuseio de imagens maiores (digamos  $512 \times 512$ ) -- Tentando maiores tamanhos de filtro (correspondente a um RF maior) como  $11 \times 11$  – Aprender mais filtros (digamos 128) – Explorar arquiteturas mais profundas (mais de 100 camadas) – Alcançar a invariância de tradução (a capacidade de reconhecer um recurso independentemente de onde eles aparecem na imagem).

## Strides e Padding

**Como os filtros são posicionados?** Em geral, os filtros são dispostos em telhas sobrepostas, da esquerda para a direita e de cima para baixo. Cada camada de convolução tem um parâmetro para especificar a `filter_shape`, especificando a largura e a altura do filtro no caso das imagens de cena mais naturais. Há um parâmetro (`strides`) que controla a distância até a etapa para a direita ao mover os filtros através de vários RF's em uma linha, e até que ponto para descer quando se move para a próxima linha. O parâmetro booleano `pad` controla se a entrada deve ser preenchida em torno das bordas para permitir um mosaico completo dos RFs perto das bordas.

A animação acima mostra os resultados com um `filter_shape = (3, 3)`, `strides = (2, 2)` e `pad = False`. As duas animações abaixo mostram os resultados quando `pad` é definido como True. Primeiro, com um passo de 2 e segundo tendo um passo de 1.

Nota: a forma da saída é diferente entre as duas configurações. Muitas vezes a sua decisão de `pad` e os valores de `stride` é baseada na forma da camada de saída necessária.

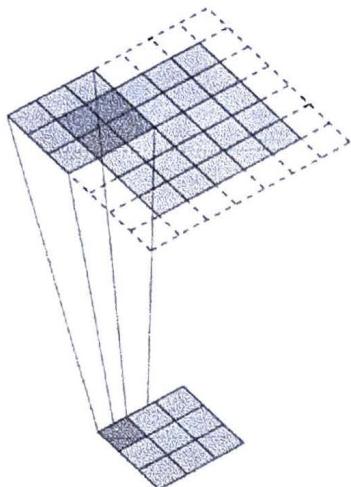
In [2]:

```
from IPython.display import display, Image

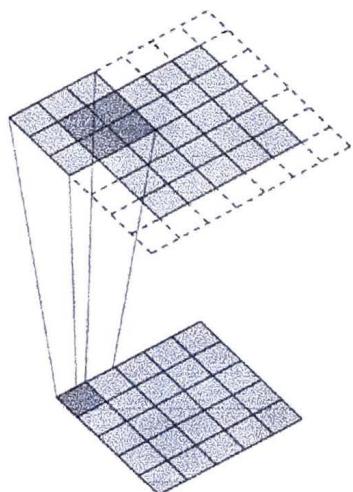
# Plot images com strides de 2 e 1 e padding habilitado
images = [("https://www.cntk.ai/jup/cntk103d_padding_strides.gif" , 'Stride = 2'),
          ("https://www.cntk.ai/jup/cntk103d_same_padding_no_strides.gif", 'Stride = 1')]

for im in images:
    print(im[1])
    display(Image(url=im[0], width=200, height=200))
```

Stride = 2



Stride = 1



## Pooling Layer

Muitas vezes, é necessário controlar o número de parâmetros, especialmente em redes profundas. Para cada camada de saída da camada de convolução (cada camada, corresponde à saída de um filtro), pode-se ter uma camada de agrupamento (Pooling). As camadas de agrupamento são tipicamente introduzidas para:

- Reduzir a dimensionalidade da camada anterior (acelerando a rede),
- Torna o modelo mais tolerante a alterações no local do objeto na imagem. Por exemplo, mesmo quando um dígito é deslocado para um lado da imagem em vez de estar no meio.

E comum inserir periodicamente uma camada de agrupamento entre as camadas Convolucionais sucessivas em uma arquitetura ConvNet. Sua função é reduzir progressivamente o tamanho espacial da representação para reduzir a quantidade de parâmetros e de computação na rede e, portanto, também para controlar o overfitting. A Camada de Agrupamento opera independentemente em cada fatia de profundidade da entrada e redimensiona-a espacialmente, usando a operação MAX. A forma mais comum é uma camada de pooling com filtros de tamanho  $2 \times 2$  aplicado com um stride de 2 downsamples cada fatia de profundidade na entrada por 2 ao longo de largura e altura, descartando 75% das ativações. Cada operação MAX, neste caso, seria tomar um máximo de 4 números (pequena região  $2 \times 2$  em alguma fatia de profundidade). A dimensão da profundidade permanece inalterada.

Vale ressaltar que existem apenas duas variações comumente observadas na camada de Max Pooling encontradas na prática: Uma camada de agrupamento com  $F = 3$ ,  $S = 2$  (também chamada de pool de sobreposição) e mais comumente  $F = 2$ ,  $S = 2$ . Agrupando tamanhos com campos receptivos maiores pode destruir a rede e travar a máquina.

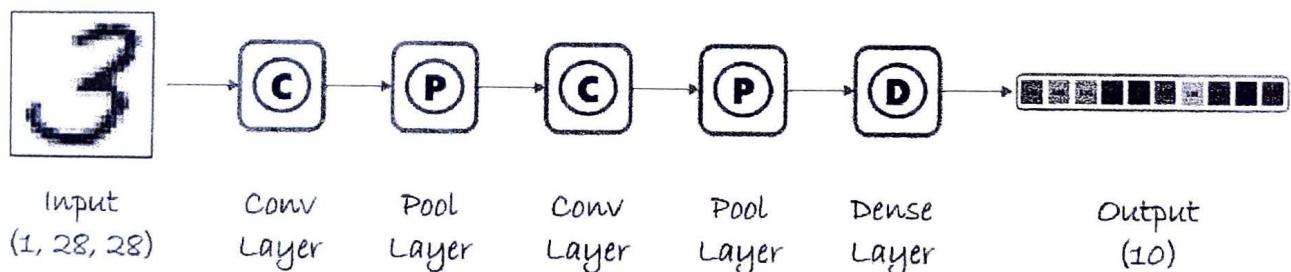
O cálculo em um nó de pooling é muito mais simples do que um nó de feedforward normal. Ele não tem peso, bias ou função de ativação. Ele usa uma função de agregação simples (como max ou average) para calcular sua saída. A função mais comumente usada é "max" - um nó de pooling máximo simplesmente fornece o máximo dos valores de entrada correspondentes à posição do filtro da entrada. A figura abaixo mostra os valores de entrada em uma região  $4 \times 4$ . O tamanho máximo da janela de agrupamento é  $2 \times 2$  e começa a partir do canto superior esquerdo. O valor máximo dentro da janela torna-se a saída da região. Cada vez que o modelo é deslocado pela quantidade especificada pelo parâmetro stride (como mostrado na figura abaixo) e a operação de pooling máximo é repetida.

5	7	2	6
1	9	3	1
2	4	2	0
0	3	8	5

2x2 max pooling  
stride 2

9	6
4	8

## Rede Convolucional Típica

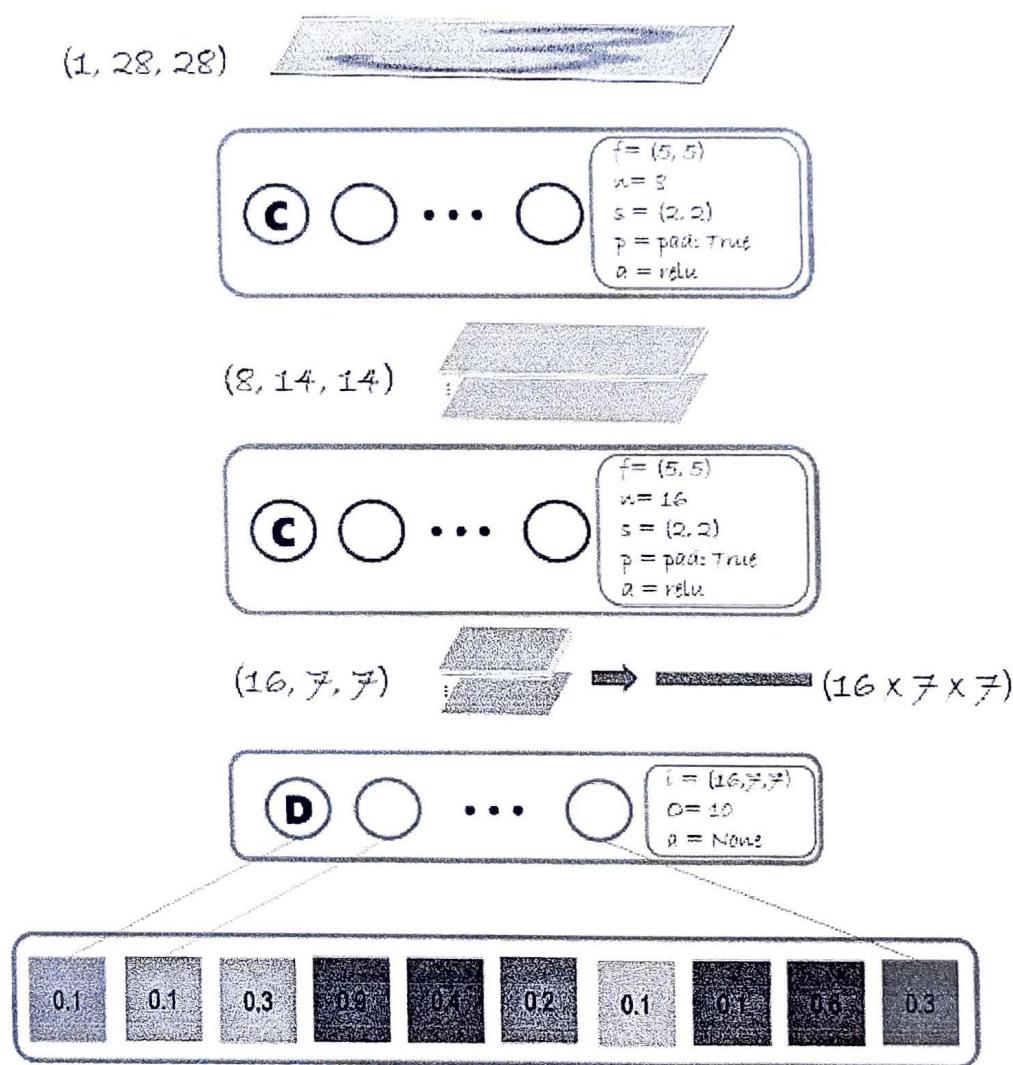


Uma CNN típica contém um conjunto de camadas alternadas de convolução e agrupamento (Pooling) seguido por uma camada de saída densa para a classificação. Você encontrará variantes desta estrutura em muitas redes profundas clássicas (VGG, AlexNet, etc.). Isto está em contraste com a rede MLP, que consiste em 2

camadas densas seguidas por uma camada de saída densa.

As ilustrações são apresentadas no contexto de imagens bidimensionais (2D), mas o conceito e os componentes podem operar em qualquer dado dimensional. O esquema acima mostra 2 camadas de convolução e 2 camadas de agrupamento máximo. Uma estratégia típica é aumentar o número de filtros nas camadas mais profundas, reduzindo o tamanho espacial de cada camada intermediária. Camadas intermediárias.

A figura a seguir ilustra o modelo que vamos construir. Observe que os parâmetros no modelo abaixo devem ser experimentados. Estes são frequentemente chamados de hiperparâmetros de rede. Aumentar a forma do filtro leva a um aumento no número de parâmetros do modelo, aumenta o tempo de computação e ajuda o modelo a se ajustar melhor aos dados. No entanto, corre-se o risco de overfitting (<https://en.wikipedia.org/wiki/Overfitting>). Normalmente, o número de filtros nas camadas mais profundas é maior do que o número de filtros nas camadas anteriores. Escolhemos 8 e 16 como número de filtros para a primeira e segunda camadas, respectivamente. Estes hiperparâmetros devem ser experimentados durante a construção do modelo.



#### Compreendendo os parâmetros :

Nosso modelo tem duas camadas de convolução, cada uma com peso e bias. Isso adiciona até 4 tensores de parâmetro. Adicionalmente, a camada densa tem tensores de peso e de bias. Assim, os tensores de 6 parâmetros.

Vamos agora contar o número de parâmetros:

- Primeira camada de convolução \*: Existem 8 filtros cada um de tamanho  $(1 \times 5 \times 5)$  onde 1 é o número de canais na imagem de entrada. Isto adiciona até 200 valores na matriz de peso e 8 valores de bias.
- Segunda camada de convolução \*: Existem 16 filtros cada um de tamanho  $(8 \times 5 \times 5)$  onde 8 é o número de canais na entrada para a segunda camada (= saída da primeira camada). Isto adiciona até 3200 valores na matriz de peso e 16 valores de bias.
- Última camada densa \*: Existem  $16 \times 7 \times 7$  valores de entrada e produz 10 valores de saída correspondentes aos 10 dígitos no conjunto de dados MNIST. Isto corresponde a  $(16 \times 7 \times 7) \times 10$  valores de peso e 10 valores de bias.

Adicionando estes acima dá os 11274 parâmetros no modelo.

## Construindo e Treinando o Modelo

### Definindo os Dados e Hyperparâmetros

In [3]:

```
# Imports
import os
import sys
import inspect
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from modulos import utils
from datetime import datetime
from tensorflow.python.framework import ops
from sklearn.metrics.classification import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
import warnings
```

*importando os pacotes*

In [4]:

```
# Versão do TensorFlow
# Para instalar a mesma versão do TF, use:
# CPU: pip install tensorflow==1.8 (no prompt ou terminal)
# GPU: pip install tensorflow_gpu==1.8 (no prompt ou terminal)
tf.__version__
```

Out[4]:

'1.8.0'

In [5]:

```
warnings.filterwarnings("ignore")
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
ops.reset_default_graph()
np.random.seed(123456789)
```

In [6]:

de fornendo os data sets

```
FLAGS = tf.flags.FLAGS
tf.flags.DEFINE_string("data_dir", "dataset/", "Caminho para o diretório com dados de treinamento")
tf.flags.DEFINE_string("logs_dir", "modelo/", "Caminho para o diretório onde o modelo será salvo")
tf.flags.DEFINE_string("mode", "train", "mode: train (Default)/ test")
```

In [7]:

# Hyperparâmetros

BATCH\_SIZE = 128 → momento do conjunto de dados em cada passada  
LEARNING\_RATE = 1e-3 → (notação científica) como o gradiente sera aplicado  
MAX\_ITERATIONS = 1000  
REGULARIZATION = 1e-3  
IMAGE\_SIZE = 48  
NUM\_LABELS = 7  
VALIDATION\_PERCENT = 0.1

## Funções Auxiliares Para Construção do Modelo

In [8]:

```
def add_to_regularization_loss(W, b):
    tf.add_to_collection("losses", tf.nn.l2_loss(W))
    tf.add_to_collection("losses", tf.nn.l2_loss(b))
```

In [9]:

```
def weight_variable(shape, stddev=0.02, name=None):
    initial = tf.truncated_normal(shape, stddev=stddev)
    if name is None:
        return tf.Variable(initial)
    else:
        return tf.get_variable(name, initializer=initial)
```

In [10]:

```
def bias_variable(shape, name=None):
    initial = tf.constant(0.0, shape=shape)
    if name is None:
        return tf.Variable(initial)
    else:
        return tf.get_variable(name, initializer=initial)
```

## Construção do Modelo

In [11]:

```
def emotionCNN(dataset):

    # Camada de Convolução 1
    with tf.name_scope("conv1") as scope:
        tf.summary.histogram("W_conv1", weights['wc1'])
        tf.summary.histogram("b_conv1", biases['bc1'])
        conv_1 = tf.nn.conv2d(dataset, weights['wc1'], strides=[1, 1, 1, 1], padding="SAME")
        h_conv1 = tf.nn.bias_add(conv_1, biases['bc1'])
        h_1 = tf.nn.relu(h_conv1)
        h_pool1 = tf.nn.max_pool(h_1, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")
        add_to_regularization_loss(weights['wc1'], biases['bc1'])

    # Camada de Convolução 2
    with tf.name_scope("conv2") as scope:
        tf.summary.histogram("W_conv2", weights['wc2'])
        tf.summary.histogram("b_conv2", biases['bc2'])
        conv_2 = tf.nn.conv2d(h_pool1, weights['wc2'], strides=[1, 1, 1, 1], padding="SAME")
        h_conv2 = tf.nn.bias_add(conv_2, biases['bc2'])
        h_2 = tf.nn.relu(h_conv2)
        h_pool2 = tf.nn.max_pool(h_2, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding="SAME")
        add_to_regularization_loss(weights['wc2'], biases['bc2'])

    # Camada Totalmente Conectada 1
    with tf.name_scope("fc_1") as scope:
        prob = 0.5
        image_size = IMAGE_SIZE // 4
        h_flat = tf.reshape(h_pool2, [-1, image_size * image_size * 64])
        tf.summary.histogram("W_fc1", weights['wf1'])
        tf.summary.histogram("b_fc1", biases['bf1'])
        h_fc1 = tf.nn.relu(tf.matmul(h_flat, weights['wf1']) + biases['bf1'])
        h_fc1_dropout = tf.nn.dropout(h_fc1, prob)

    # Camada Totalmente Conectada 2
    with tf.name_scope("fc_2") as scope:
        tf.summary.histogram("W_fc2", weights['wf2'])
        tf.summary.histogram("b_fc2", biases['bf2'])
        pred = tf.matmul(h_fc1_dropout, weights['wf2']) + biases['bf2']

    return pred
```

In [12]:

```
# Pesos e Bias do Modelo
weights = {
    'wc1': weight_variable([5, 5, 1, 32], name="W_conv1"),
    'wc2': weight_variable([3, 3, 32, 64], name="W_conv2"),
    'wf1': weight_variable([int((IMAGE_SIZE // 4) * (IMAGE_SIZE // 4)) * 64, 256], name="W_fc1"),
    'wf2': weight_variable([256, NUM_LABELS], name="W_fc2")
}

biases = {
    'bc1': bias_variable([32], name="b_conv1"),
    'bc2': bias_variable([64], name="b_conv2"),
    'bf1': bias_variable([256], name="b_fc1"),
    'bf2': bias_variable([NUM_LABELS], name="b_fc2")
}
```

In [13]:

Calcular o perda

```
def loss(pred, label):
    cross_entropy_loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits=pred,
        tf.summary.scalar('Entropy', cross_entropy_loss)
    reg_losses = tf.add_n(tf.get_collection("losses"))
    tf.summary.scalar('Reg_loss', reg_losses)
    return cross_entropy_loss + REGULARIZATION * reg_losses
```

In [14]:

```
def train(loss, step):
    return tf.train.AdamOptimizer(LEARNING_RATE).minimize(loss, global_step=step)
```

In [15]:

```
def get_next_batch(images, labels, step):
    offset = (step * BATCH_SIZE) % (images.shape[0] - BATCH_SIZE)
    batch_images = images[offset: offset + BATCH_SIZE]
    batch_labels = labels[offset:offset + BATCH_SIZE]
    return batch_images, batch_labels
```

In [16]:

```
# Listas para resultados de treinamento
train_error_list = []
train_step_list = []      Recebe a taxa de erros de treino

# Listas para resultados de validação
valid_error_list = []
valid_step_list = []      erros de Validação
```

## Treinamento

In [17]:

```
def main(argv=None):  
  
    # Carrega os dados  
    train_images, train_labels, valid_images, valid_labels, test_images = utils.read_data(F  
  
    print("\nTamanho do Dataset de Treino: %s" % train_images.shape[0])  
    print('Tamanho do Dataset de Validação: %s' % valid_images.shape[0])  
    print("Tamanho do Dataset de Teste: %s" % test_images.shape[0])  
  
    global_step = tf.Variable(0, trainable=False)  
    dropout_prob = tf.placeholder(tf.float32) define Placeholders  
    input_dataset = tf.placeholder(tf.float32, [None, IMAGE_SIZE, IMAGE_SIZE, 1], name="inp  
    input_labels = tf.placeholder(tf.float32, [None, NUM_LABELS]) reservoar os dados espaços no memória  
  
    pred = emotionCNN(input_dataset) Função que constrói o  
    output_pred = tf.nn.softmax(pred, name="output") modelo  
    loss_val = loss(pred, input_labels)  
    train_op = train(loss_val, global_step)  
  
    summary_op = tf.summary.merge_all()  
    init_op = tf.global_variables_initializer() inicializa as variáveis  
  
    with tf.Session() as sess: agora começa o treinamento  
        sess.run(init_op)  
        summary_writer = tf.summary.FileWriter(FLAGS.logs_dir, sess.graph)  
        saver = tf.train.Saver()  
        ckpt = tf.train.get_checkpoint_state(FLAGS.logs_dir)  
        if ckpt and ckpt.model_checkpoint_path: se não tiver 5 horas para  
            & saver.restore(sess, ckpt.model_checkpoint_path) processar o modelo e deixar problema com 1 hora, você pode  
            print("Modelo Restaurado!") esse trabalho sem o checkpoint  
  
        Volta do ntf, me check Point & for step in range(MAX_ITERATIONS):  
            batch_image, batch_label = get_next_batch(train_images, train_labels, step)  
            feed_dict = {input_dataset: batch_image, input_labels: batch_label}  
  
            sess.run(train_op, feed_dict=feed_dict) executa as funções  
            if step % 10 == 0:  
                train_loss, summary_str = sess.run([loss_val, summary_op], feed_dict=feed_dict)  
                summary_writer.add_summary(summary_str, global_step=step)  
                train_error_list.append(train_loss)  
                train_step_list.append(step)  
                print("Taxa de Erro no Treinamento: %f" % train_loss)  
  
            if step % 100 == 0:  
                valid_loss = sess.run(loss_val, feed_dict={input_dataset: valid_images, input_labels: valid_labels})  
                valid_error_list.append(valid_loss)  
                valid_step_list.append(step)  
                print("%s Taxa de Erro na Validação: %f" % (datetime.now(), valid_loss))  
                saver.save(sess, FLAGS.logs_dir + 'model.ckpt', global_step=step)  
  
    # Plot do erro durante o treinamento  
    plt.plot(train_step_list, train_error_list, 'r--', label='Erro no Treinamento Por Iteração')  
    plt.title('Erro no Treinamento Por Iteração')  
    plt.xlabel('Iteração')  
    plt.ylabel('Erro no Treinamento')  
    plt.legend(loc='upper right')  
    plt.show()
```

```
# Plot do erro durante a validação
plt.plot(valid_step_list, valid_error_list, 'r--', label='Erro na Validação Por Iteração')
plt.title('Erro na Validação Por Iteração')
plt.xlabel('Iteração')
plt.ylabel('Erro na Validação')
plt.legend(loc='upper right')
plt.show()

print(train_error_list)
print(valid_error_list)
```

[ ] listas vermelhas

In [18]:

```
if __name__ == "__main__":
    tf.app.run()
    print("Treinamento concluído")
```

```
Lendo train.csv ...
(4178, 48, 48, 1)
(4178, 7)
Lendo test.csv ...
```

Salvando ...

```
Tamanho do Dataset de Treino: 3761
Tamanho do Dataset de Validação: 417
Tamanho do Dataset de Teste: 1312
WARNING:tensorflow:From <ipython-input-13-dec8419985f4>:2: softmax_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is deprecated and will be removed in a future version.
Instructions for updating:
```

Future major versions of TensorFlow will allow gradients to flow into the labels input on backprop by default.

See @{\$tf.nn.softmax\_cross\_entropy\_with\_logits\_v2}.

Para adquirir conhecimento técnico sólido e especializado em Deep Learning, Visão Computacional, Processamento de Linguagem Natural e outros temas relacionados à Inteligência Artificial, confira nosso programa completo: [Formação Inteligência Artificial \(<https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial>\)](https://www.datascienceacademy.com.br/pages/formacao-inteligencia-artificial).

## Fim

Obrigado - Data Science Academy - [facebook.com/dsacademybr](http://facebook.com/dsacademybr)  
[\(http://facebook.com/dsacademybr\)](http://facebook.com/dsacademybr)