

Data Science Academy

Big Data Real-Time Analytics com Python e Spark

Capítulo 2

Para gerar um arquivo pdf a partir do Jupyter Notebook, você precisar ter instalado em seu computador:

MiKTeX e Pandoc (Windows)

LaTeX e Pandoc (Mac)

TeX Live e Pandoc (linux)

Neste link você encontra todas as informações: <http://pandoc.org/installing.html>
[\(http://pandoc.org/installing.html\)](http://pandoc.org/installing.html)

Computação com Numpy

In [1]:

```
# Importando o módulo Numpy  
import numpy as np
```

Criando Estruturas de Dados NumPy

In [2]:

```
# Criando array unidimensional  
array1 = np.array([10, 20, 30, 40])
```

In [3]:

```
# Criando array bi-dimensional  
array2 = np.array([[100, 83, 15],[42, 78, 0]])
```

In [4]:

```
array1
```

Out[4]:

```
array([10, 20, 30, 40])
```

In [5]:

```
array2
```

Out[5]:

```
array([[100,  83,  15],  
       [ 42,  78,   0]])
```

In [6]:

```
# Verificando o formato (shape) do array  
array1.shape
```

Out[6]:

```
(4,)
```

In [7]:

```
# Verificando o formato (shape) do array  
array2.shape
```

Out[7]:

```
(2, 3)
```

In [8]:

```
# Verificando o número de dimensões  
array2.ndim
```

Out[8]:

```
2
```

Em Python TUDO é objeto, com métodos e atributos

Método - realiza ação no objeto

Atributo - característica do objeto

In [9]:

```
# Método  
array2.max()
```

Out[9]:

```
100
```

In [10]:

```
# Atributo  
array2.ndim
```

Out[10]:

```
2
```

Criando estruturas de dados com arange

In [11]:

```
array3 = np.arange(15)
```

In [12]:

```
array3
```

Out[12]:

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

In [13]:

```
# Usando start/end (exclusive)  
array4 = np.arange(0, 15, 5)
```

In [14]:

```
array4
```

Out[14]:

```
array([ 0,  5, 10])
```

Criando estruturas de dados com linspace

In [15]:

```
# Argumentos: (start, end, number of elements)  
array5 = np.linspace(0, 3, 4); array5
```

Out[15]:

```
array([0., 1., 2., 3.])
```

Criando estruturas de dados com outras funções

numpy.zeros

In [16]:

```
# Array 10x8 de zeros
array6 = np.zeros((10, 8)); array6
```

Out[16]:

```
array([[0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 0., 0., 0.]])
```

numpy.ones

In [17]:

```
# Array 2x3x2 de 1's
array7 = np.ones((2, 3, 2)); array7
```

Out[17]:

```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[[1., 1.],
         [1., 1.],
         [1., 1.]]]])
```

numpy.eye

In [18]:

```
# Produz uma Identitiy Matrix
array8 = np.eye(3); array8
```

Out[18]:

```
array([[1., 0., 0.],
       [0., 1., 0.],
       [0., 0., 1.]])
```

numpy.diag

In [19]:

```
# Matriz diagonal
array9 = np.diag((2,1,4,6)); array9
```

Out[19]:

```
array([[2, 0, 0, 0],
       [0, 1, 0, 0],
       [0, 0, 4, 0],
       [0, 0, 0, 6]])
```

numpy.random.rand

In [20]:

```
# A função rand(n) produz uma sequência de números uniformemente distribuídos com range de
np.random.seed(100) # Set seed
array10 = np.random.rand(5); array10
```

Out[20]:

```
array([0.54340494, 0.27836939, 0.42451759, 0.84477613, 0.00471886])
```

In [21]:

```
# A função randn(n) produz uma sequência de números com distribuição normal (Gaussian)
array11 = np.random.randn(5); array11
```

Out[21]:

```
array([-0.35467445, -0.78606433, -0.2318722 ,  0.20797568,  0.93580797])
```

numpy.empty

In [22]:

```
array12 = np.empty((3,2)); array12
```

Out[22]:

```
array([[0.00000000e+000,  0.00000000e+000],
       [1.77229088e-310,  3.50977866e+064],
       [0.00000000e+000,  0.00000000e+000]])
```

numpy.tile

In [23]:

```
np.array([[9, 4], [3, 7]])
```

Out[23]:

```
array([[9, 4],  
       [3, 7]])
```

In [24]:

```
np.tile(np.array([[9, 4], [3, 7]]), 4)
```

Out[24]:

```
array([[9, 4, 9, 4, 9, 4, 9, 4],  
       [3, 7, 3, 7, 3, 7, 3, 7]])
```

In [25]:

```
np.tile(np.array([[9, 4], [3, 7]]), (2,2))
```

Out[25]:

```
array([[9, 4, 9, 4],  
       [3, 7, 3, 7],  
       [9, 4, 9, 4],  
       [3, 7, 3, 7]])
```

Tipos de Dados NumPy

In [26]:

```
array13 = np.array([8, -3, 5, 9], dtype = 'float')
```

In [27]:

```
array13
```

Out[27]:

```
array([ 8., -3.,  5.,  9.])
```

In [28]:

```
array13.dtype
```

Out[28]:

```
dtype('float64')
```

In [29]:

```
array14 = np.array([2, 4, 6, 8])
```

In [30]:

```
array14
```

Out[30]:

```
array([2, 4, 6, 8])
```

In [31]:

```
array14.dtype
```

Out[31]:

```
dtype('int64')
```

In [32]:

```
array15 = np.array([2.0, 4, 6, 8])
```

In [33]:

```
array15.dtype
```

Out[33]:

```
dtype('float64')
```

In [34]:

```
array16 = np.array(['Análise', 'de', 'Dados', 'com Python'])
```

In [35]:

```
array16.dtype
```

Out[35]:

```
dtype('<U10')
```

In [36]:

```
array17 = np.array([True, False, True])
```

In [37]:

```
array17.dtype
```

Out[37]:

```
dtype('bool')
```

In [38]:

```
array18 = np.array([7, -3, 5.24])
```

In [39]:

array18.dtype

Out[39]:

dtype('float64')

In [40]:

array18.astype(int)

Out[40]:

array([7, -3, 5])

Operações com Arrays

In [41]:

Array começando por 0, com 30 elementos e multiplicado por 5
array19 = np.arange(0, 30) * 5

In [42]:

array19

Out[42]:

array([0, 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60,
 65, 70, 75, 80, 85, 90, 95, 100, 105, 110, 115, 120, 125,
 130, 135, 140, 145])

In [43]:

array19 = np.arange(0, 30)

In [44]:

array19

Out[44]:

array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16,
 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])

In [45]:

Elevando um array a potência
array20 = np.arange(5) ** 4

In [46]:

```
array20
```

Out[46]:

```
array([ 0,  1, 16, 81, 256])
```

In [47]:

```
# Somando um número a cada elemento do array
array21 = np.arange(0, 30) + 1
```

In [48]:

```
array21
```

Out[48]:

```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30])
```

In [49]:

```
array22 = np.arange(0, 30, 3) + 3
array23 = np.arange(1, 11)
```

In [50]:

```
array22
```

Out[50]:

```
array([ 3,  6,  9, 12, 15, 18, 21, 24, 27, 30])
```

In [51]:

```
array23
```

Out[51]:

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

In [52]:

```
# Subtração
array22 - array23
```

Out[52]:

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

In [53]:

```
# Soma  
array22 + array23
```

Out[53]:

```
array([ 4,  8, 12, 16, 20, 24, 28, 32, 36, 40])
```

In [54]:

```
# Divisão  
array22 / array23
```

Out[54]:

```
array([3., 3., 3., 3., 3., 3., 3., 3., 3.])
```

In [55]:

```
# Multiplicação  
array22 * array23
```

Out[55]:

```
array([ 3, 12, 27, 48, 75, 108, 147, 192, 243, 300])
```

In [56]:

```
# Podemos ainda comparar arrays  
array22 > array23
```

Out[56]:

```
array([ True,  True,  True,  True,  True,  True,  True,  True,  
       True])
```

In [57]:

```
arr1 = np.array([True, False, True, False])  
arr2 = np.array([False, False, True, False])  
np.logical_and(arr1, arr2)
```

Out[57]:

```
array([False, False,  True, False])
```

Podemos criar arrays com Python, sem usar o módulo NumPy. Porém o NumPy é muito mais rápido

In [58]:

```
array_numpy = np.arange(1000)  
%timeit array_numpy ** 4
```

```
3.24 µs ± 28.5 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In [59]:

```
array_python = range(1000)
%timeit [array_python[i] ** 4 for i in array_python]
350 µs ± 13.6 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

A métrica μs significa microsegundos.

Fim

Obrigado - Data Science Academy - facebook.com/dsacademybr
<http://facebook.com/dsacademybr>

Data Science Academy

Big Data Real-Time Analytics com Python e Spark

Capítulo 3

Estudo de Caso - Análise Exploratória de Dados - Parte 1

Analisando dados de aluguel de bikes como táxis na cidade de New York.

In [9]:

```
# Imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [10]:

```
# Importando o arquivo csv
df = pd.read_csv('data/taxis_bikes_nycity.csv')
```

In [11]:

```
print(type(df))

<class 'pandas.core.frame.DataFrame'>
```

In [12]:

```
df.head(10)
```

Out[12]:

	Data	Distancia	Tempo
0	8/2/15	1.70	NaN
1	8/3/15	1.40	NaN
2	8/4/15	2.10	NaN
3	8/6/15	2.38	NaN
4	8/7/15	2.30	NaN
5	8/8/15	3.40	NaN
6	8/9/15	2.50	NaN
7	8/10/15	3.36	0:28:37
8	8/11/15	1.67	0:13:07
9	8/12/15	1.42	0:10:35

In [13]:

```
df.dtypes
```

Out[13]:

```
Data          object
Distancia    float64
Tempo         object
dtype: object
```

In [6]:

```
df.columns
```

Out[6]:

```
Index(['Data', 'Distancia', 'Tempo'], dtype='object')
```

In [7]:

```
df.index
```

Out[7]:

```
RangeIndex(start=0, stop=81, step=1)
```

In [8]:

df['Data'].head()

Out[8]:

```
0    8/2/15
1    8/3/15
2    8/4/15
3    8/6/15
4    8/7/15
```

Name: Data, dtype: object

Regendo Coluna Dato

In [9]:

df = pd.read_csv('data/taxis_bikes_nycity.csv', parse_dates = ['Data'])

Convertendo coluna dato
data para tipo

In [10]:

df['Data'].head()

Out[10]:

```
0    2015-08-02
1    2015-08-03
2    2015-08-04
3    2015-08-06
4    2015-08-07
```

Name: Data, dtype: datetime64[ns]

In [11]:

df.set_index('Data', inplace = True) ?

In [12]:

```
df.head(10)
```

Out[12]:

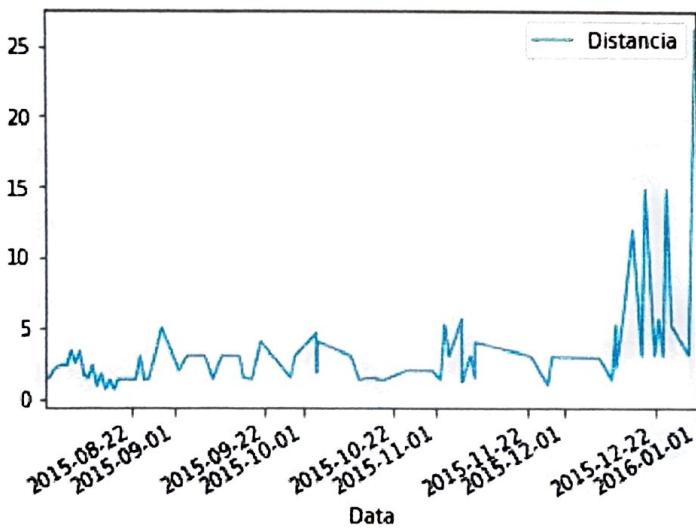
Data	Distancia	Tempo
2015-08-02	1.70	NaN
2015-08-03	1.40	NaN
2015-08-04	2.10	NaN
2015-08-06	2.38	NaN
2015-08-07	2.30	NaN
2015-08-08	3.40	NaN
2015-08-09	2.50	NaN
2015-08-10	3.36	0:28:37
2015-08-11	1.67	0:13:07
2015-08-12	1.42	0:10:35

In [13]:

```
df.plot()
```

Out[13]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x11a2e7eb8>
```

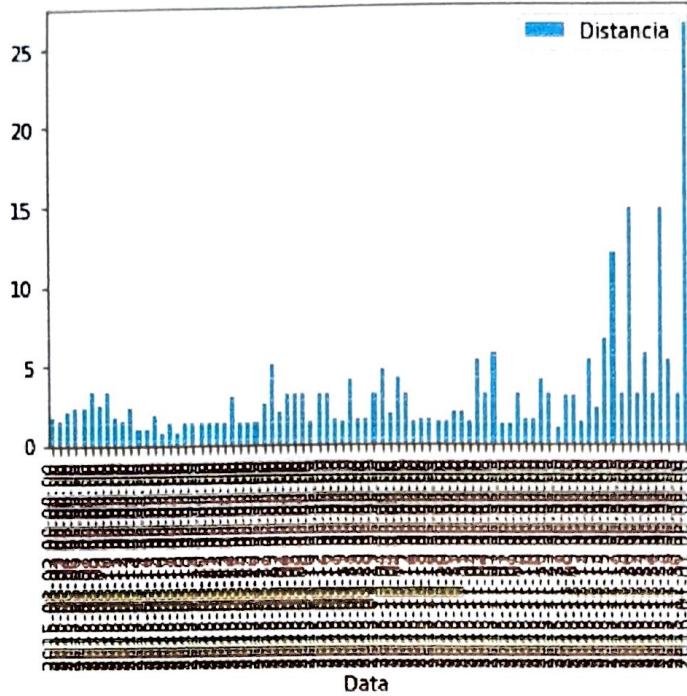


In [14]:

```
?df.plot
```

In [15]:

```
df.plot(kind = 'bar')
plt.show()
```

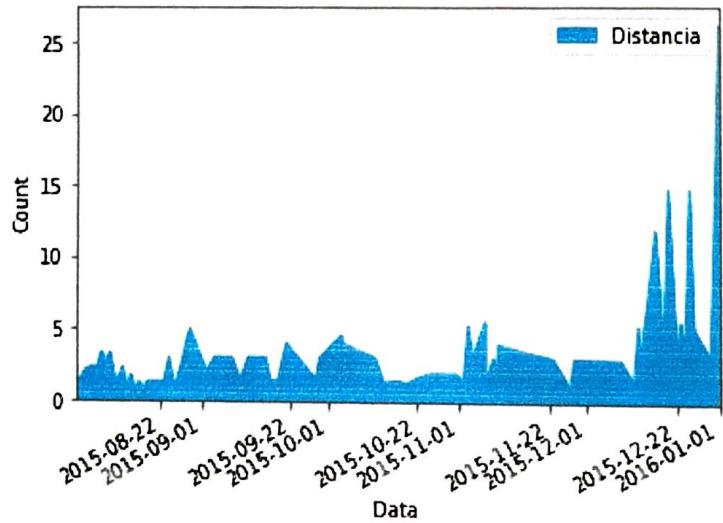


In [16]:

```
df.plot(kind = 'area')
plt.ylabel("Count")
```

Out[16]:

```
Text(0, 0.5, 'Count')
```



In [17]:

df.describe()

Out[17]:

	Distancia
count	81.000000
mean	3.137531
std	3.634519
min	0.650000
25%	1.370000
50%	2.100000
75%	3.050000
max	26.200000

In [18]:

df['2015-11']

Out[18]:

	Distancia	Tempo
Data		
2015-11-02	1.37	0:08:43
2015-11-03	5.30	0:39:26
2015-11-04	3.05	0:22:12
2015-11-07	5.63	0:49:05
2015-11-07	1.26	NaN
2015-11-07	1.20	NaN
2015-11-09	3.05	0:22:36
2015-11-10	1.50	0:09:00
2015-11-10	1.50	0:11:33
2015-11-10	4.00	NaN
2015-11-23	3.05	0:22:35
2015-11-27	1.00	NaN
2015-11-28	3.00	NaN

Pego todos dados linhas
do mês 11 - 2015

In [19]:

len(df['2015-11'])

Out[19]:

In [20]:

```
df.to_csv('data/dataframe_saved_v1.csv')
```

FIM

Salva o data frame obtido
~~assim não preciso executar tudo~~
de novo

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

Data Science Academy

Big Data Real-Time Analytics com Python e Spark

Capítulo 3

Estudo de Caso - Análise Exploratória de Dados - Parte 2

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
pd.__version__
```

Out[2]:

```
'0.24.2'
```

In [3]:

```
# Carregando os dados salvos em disco do arquivo anterior
df = pd.read_csv('data/dataframe_saved_v1.csv')
```

In [4]:

```
df.head()
```

Out[4]:

	Data	Distancia	Tempo
0	2015-08-02	1.70	NaN
1	2015-08-03	1.40	NaN
2	2015-08-04	2.10	NaN
3	2015-08-06	2.38	NaN
4	2015-08-07	2.30	NaN

In [5]:

```
df.dtypes
```

Out[5]:

```
Data      object  
Distancia    float64  
Tempo      object  
dtype: object
```

In [6]:

```
df = pd.read_csv('data/dataframe_saved_v1.csv', parse_dates = ['Data'])
```

Processo para converter Novamente

In [7]:

```
df.head()
```

Out[7]:

	Data	Distancia	Tempo
0	2015-08-02	1.70	NaN
1	2015-08-03	1.40	NaN
2	2015-08-04	2.10	NaN
3	2015-08-06	2.38	NaN
4	2015-08-07	2.30	NaN

In [8]:

```
df.dtypes
```

Out[8]:

```
Data      datetime64[ns]  
Distancia    float64  
Tempo      object  
dtype: object
```

In [9]:

```
cols = ['Data', 'Distancia', 'Tempo']
df.columns = cols
df.head()
```

Out[9]:

	Data	Distancia	Tempo
0	2015-08-02	1.70	NaN
1	2015-08-03	1.40	NaN
2	2015-08-04	2.10	NaN
3	2015-08-06	2.38	NaN
4	2015-08-07	2.30	NaN

definindo nome das colunas

In [10]:

```
# Transformando a coluna Data em indice
df.set_index('Data', inplace = True)
```

In [ii]:

df.head()

Out[11]:

Data	Distancia	Tempo
2015-08-02	1.70	NaN
2015-08-03	1.40	NaN
2015-08-04	2.10	NaN
2015-08-06	2.38	NaN
2015-08-07	2.30	NaN

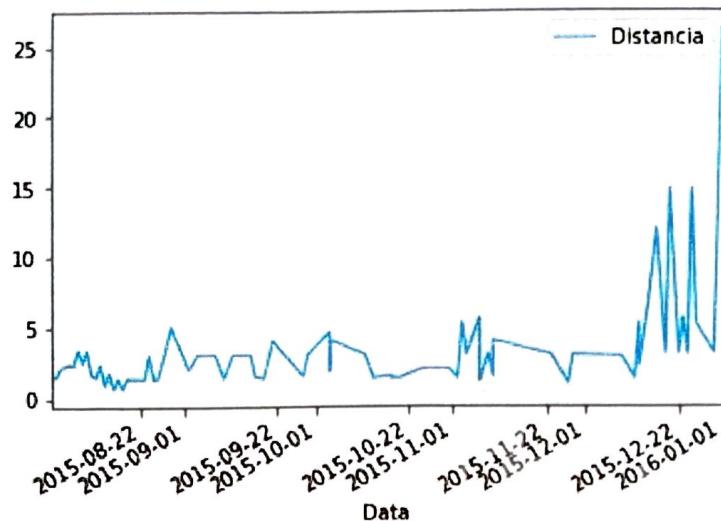
?

In [12]:

df.plot()

out[12]:

<matplotlib.axes._subplots.AxesSubplot at 0x12250acc0>



In [13]:

```
# Função para converter a coluna de duração no tempo em segundos
def calcula_total_segundos(time): ←
    if time is np.nan:
        return np.nan
    hrs, mins, seconds = str(time).split(':')
    seconds = int(seconds) + 60 * int(mins) + 60 * 60 * int(hrs)
    return seconds
```

In [14]:

df['Segundos'] = df.Tempo.map(calcula_total_segundos) # dataframe-coluna-função map(função)

←
Novo Coluna

data frame
coluna

In [15]:

df.head(10)

Out[15]:

Data	Distancia	Tempo	Segundos
2015-08-02	1.70	NaN	NaN
2015-08-03	1.40	NaN	NaN
2015-08-04	2.10	NaN	NaN
2015-08-06	2.38	NaN	NaN
2015-08-07	2.30	NaN	NaN
2015-08-08	3.40	NaN	NaN
2015-08-09	2.50	NaN	NaN
2015-08-10	3.36	0:28:37	1717.0
2015-08-11	1.67	0:13:07	787.0
2015-08-12	1.42	0:10:35	635.0

Criando nova varável
filter selection
de varável
seleção

In [16]:

df.describe()

Out[16]:

	Distancia	Segundos
count	81.000000	52.000000
mean	3.137531	1901.788462
std	3.634519	2560.424171
min	0.650000	376.000000
25%	1.370000	583.750000
50%	2.100000	1343.500000
75%	3.050000	1743.250000
max	26.200000	15643.000000

In [17]:

df.fillna(0).describe()

Out[17]:

	Distancia	Segundos
count	81.000000	81.000000
mean	3.137531	1220.901235
std	3.634519	2240.756985
min	0.650000	0.000000
25%	1.370000	0.000000
50%	2.100000	573.000000
75%	3.050000	1426.000000
max	26.200000	15643.000000

Preenche valores NA com 0
altera os valores do mundo por ex

isso

In [18]:

df['Minutos'] = df['Segundos'].map(lambda x: x / 60)

In [19]:

df.fillna(0).describe()

Out[19]:

	Distancia	Segundos	Minutos
count	81.000000	81.000000	81.000000
mean	3.137531	1220.901235	20.348354
std	3.634519	2240.756985	37.345950
min	0.650000	0.000000	0.000000
25%	1.370000	0.000000	0.000000
50%	2.100000	573.000000	9.550000
75%	3.050000	1426.000000	23.766667
max	26.200000	15643.000000	260.716667

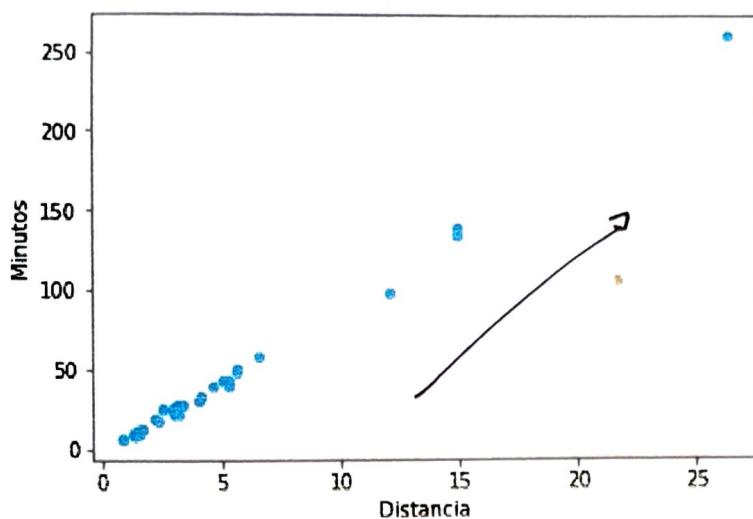
Pego cada segundo e
div de 60
para retornar minutos

In [20]:

```
df.plot(x = 'Distancia', y = 'Minutos', kind = 'scatter')
```

Out[20]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x122892fd0>
```



Ha\' umas relaçao positiva
Quanto maior a distancia
maior o tempo

In [21]:

```
df.corr()
```

Out[21]:

	Distancia	Segundos	Minutos
Distancia	1.000000	0.997203	0.997203
Segundos	0.997203	1.000000	1.000000
Minutos	0.997203	1.000000	1.000000

In [22]:

```
df.corr(method = 'spearman')
```

Out[22]:

	Distancia	Segundos	Minutos
Distancia	1.000000	0.96482	0.96482
Segundos	0.96482	1.000000	1.000000
Minutos	0.96482	1.000000	1.000000

coeficiente de colau para
o metodo de correlação

In [23]:

df.corr(method = 'kendall')

Out[23]:

	Distancia	Segundos	Minutos
Distancia	1.00000	0.88305	0.88305
Segundos	0.88305	1.00000	1.00000
Minutos	0.88305	1.00000	1.00000

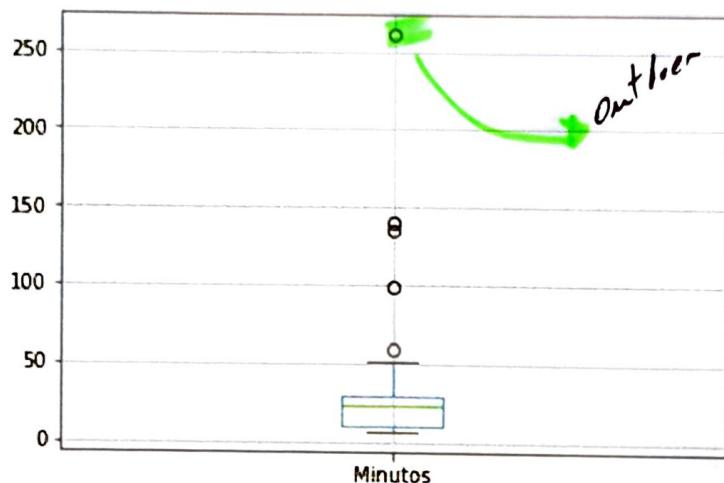
A fórmula de fazer o cálculo é diferente

In [24]:

df.boxplot('Minutos', return_type = 'axes')

Out[24]:

<matplotlib.axes._subplots.AxesSubplot at 0x192446bcf8>



In [25]:

df['Min_Por_Km'] = df['Minutos'] / df['Distancia']

Criando novo nome variação
min por Km

In [26]:

```
df.fillna(0).describe()
```

Out[26]:

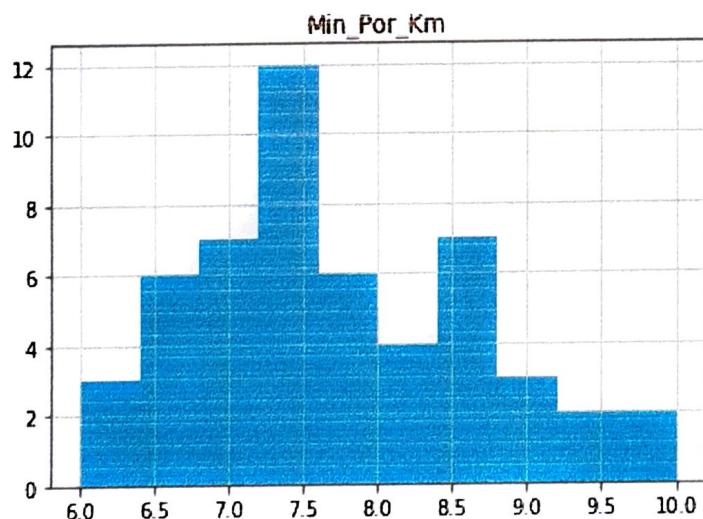
	Distancia	Segundos	Minutos	Min_Por_Km
count	81.000000	81.000000	81.000000	81.000000
mean	3.137531	1220.901235	20.348354	4.959450
std	3.634519	2240.756985	37.345950	3.803856
min	0.650000	0.000000	0.000000	0.000000
25%	1.370000	0.000000	0.000000	0.000000
50%	2.100000	573.000000	9.550000	6.962963
75%	3.050000	1426.000000	23.766667	7.792350
max	26.200000	15643.000000	260.716667	10.000000

In [27]:

```
df.hist('Min_Por_Km')
```

Out[27]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x12286db00>]],  
      dtype=object)
```

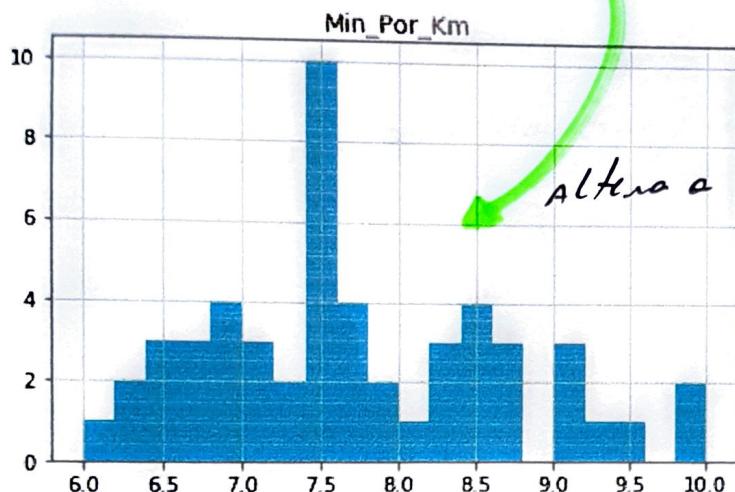


In [28]:

```
df.hist('Min_Por_Km', bins = 20)
```

Out[28]:

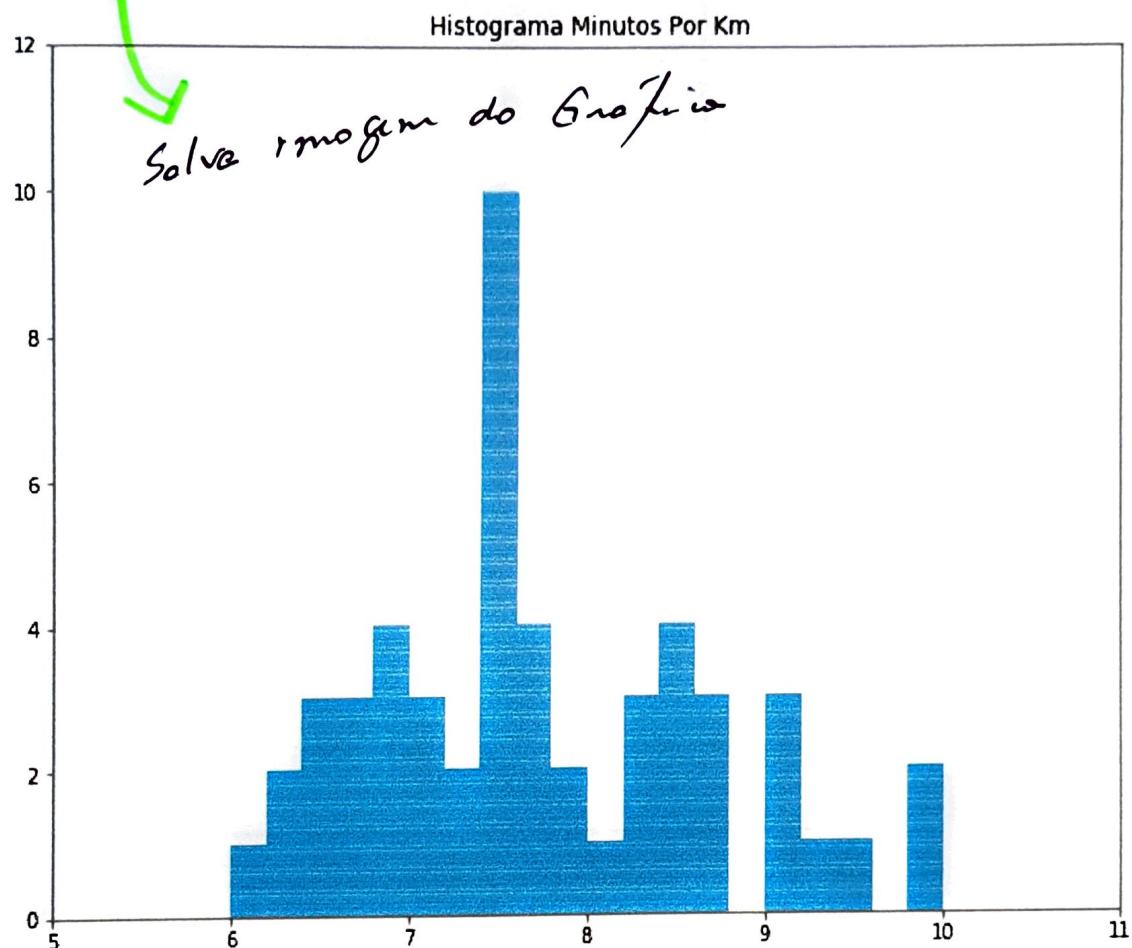
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x122843860>]],  
      dtype=object)
```



Altera a quontidade de colunas

In [29]:

```
df.hist('Min_Por_Km', bins = 20, figsize = (10, 8))
plt.xlim((5, 11))
plt.ylim((0, 12))
plt.title("Histograma Minutos Por Km")
plt.grid(False)
plt.savefig('imagens/hist_minutos_por_km.png')
```

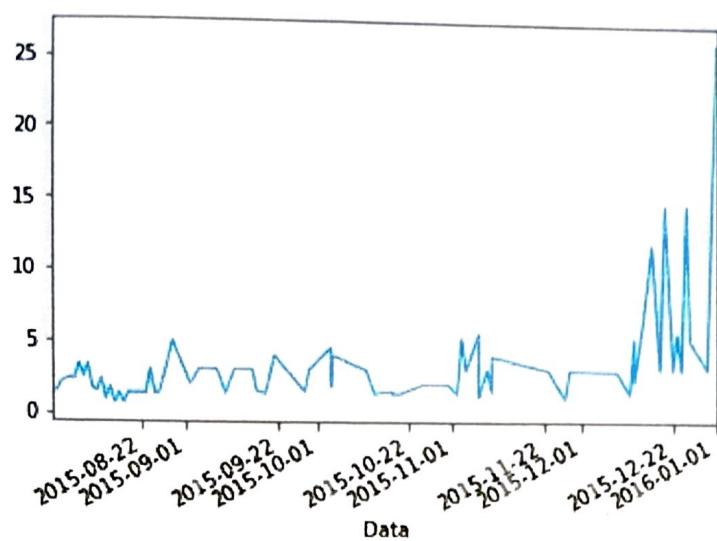


In [30]:

df['Distancia'].plot()

Out[30]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a2494e128>



In [31]:

df.head(15)

Out[31]:

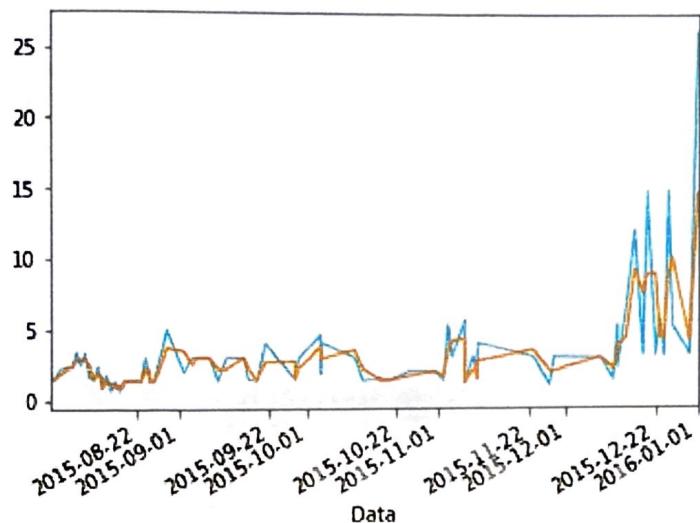
	Distancia	Tempo	Segundos	Minutos	Min_Por_Km
Data					
2015-08-02	1.70	NaN	NaN	NaN	NaN
2015-08-03	1.40	NaN	NaN	NaN	NaN
2015-08-04	2.10	NaN	NaN	NaN	NaN
2015-08-06	2.38	NaN	NaN	NaN	NaN
2015-08-07	2.30	NaN	NaN	NaN	NaN
2015-08-08	3.40	NaN	NaN	NaN	NaN
2015-08-09	2.50	NaN	NaN	NaN	NaN
2015-08-10	3.36	0:28:37	1717.0	28.616667	8.516865
2015-08-11	1.67	0:13:07	787.0	13.116667	7.854291
2015-08-12	1.42	0:10:35	635.0	10.583333	7.453052
2015-08-13	2.35	0:17:25	1045.0	17.416667	7.411348
2015-08-14	0.90	0:06:16	376.0	6.266667	6.962963
2015-08-14	0.90	0:06:16	376.0	6.266667	6.962963
2015-08-15	1.78	NaN	NaN	NaN	NaN
2015-08-16	0.65	NaN	NaN	NaN	NaN

In [32]:

```
# Calculando a média de distância em uma janela (window) de 2 horas
df['Distancia'].plot()
pd.Series(df['Distancia']).rolling(window = 2).mean().plot()
```

Out[32]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a24863198>
```

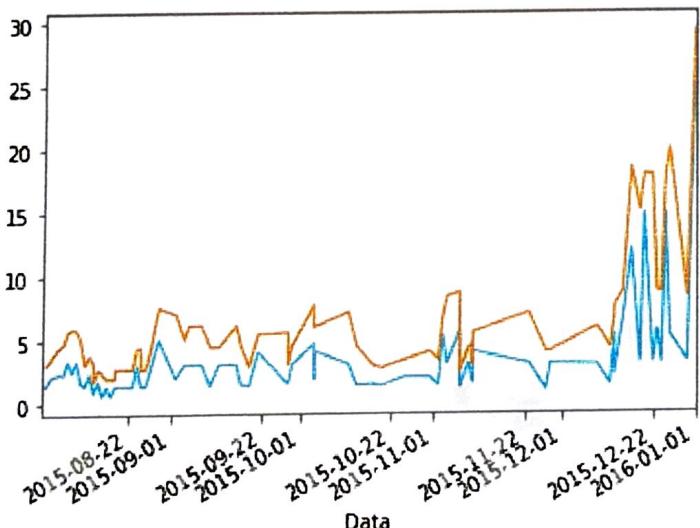


In [33]:

```
# Calculando a soma de distância em uma janela (window) de 2 horas
df['Distancia'].plot()
pd.Series(df['Distancia']).rolling(window = 2).sum().plot()
```

Out[33]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a24c42ba8>
```



In [34]:

```
df.index
```

Out[34]:

```
DatetimeIndex(['2015-08-02', '2015-08-03', '2015-08-04', '2015-08-06',
                 '2015-08-07', '2015-08-08', '2015-08-09', '2015-08-10',
                 '2015-08-11', '2015-08-12', '2015-08-13', '2015-08-14',
                 '2015-08-15', '2015-08-16', '2015-08-17',
                 '2015-08-18', '2015-08-19', '2015-08-19', '2015-08-20',
                 '2015-08-21', '2015-08-22', '2015-08-23', '2015-08-24',
                 '2015-08-25', '2015-08-25', '2015-08-26', '2015-08-27',
                 '2015-08-29', '2015-09-02', '2015-09-04', '2015-09-05',
                 '2015-09-08', '2015-09-10', '2015-09-12', '2015-09-16',
                 '2015-09-17', '2015-09-19', '2015-09-21', '2015-09-28',
                 '2015-09-28', '2015-09-29', '2015-10-04', '2015-10-04',
                 '2015-10-04', '2015-10-12', '2015-10-14', '2015-10-18',
                 '2015-10-18', '2015-10-18', '2015-10-20', '2015-10-25',
                 '2015-10-31', '2015-11-02', '2015-11-03', '2015-11-04',
                 '2015-11-07', '2015-11-07', '2015-11-07', '2015-11-09',
                 '2015-11-10', '2015-11-10', '2015-11-10', '2015-11-23',
                 '2015-11-27', '2015-11-28', '2015-12-09', '2015-12-12',
                 '2015-12-13', '2015-12-13', '2015-12-15', '2015-12-17',
                 '2015-12-19', '2015-12-20', '2015-12-22', '2015-12-23',
                 '2015-12-24', '2015-12-25', '2015-12-26', '2015-12-30',
                 '2016-01-01'],
                dtype='datetime64[ns]', name='Data', freq=None)
```

In [35]:

df['2015-11':'2015-12']

out[35]:

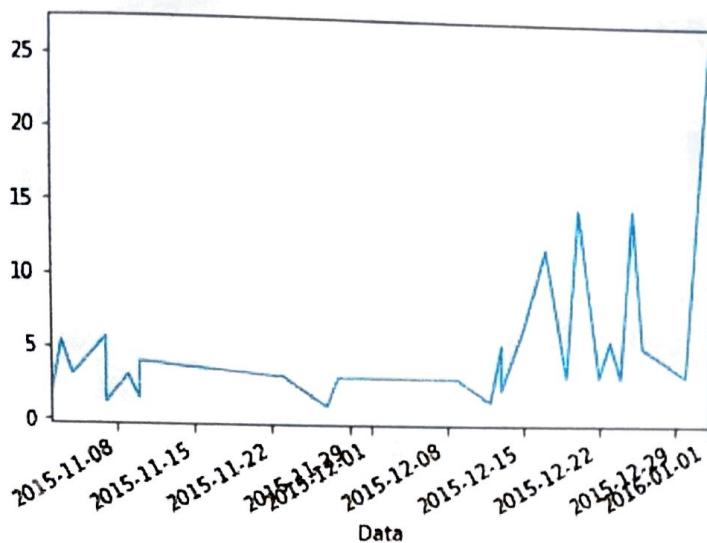
Data	Distancia	Tempo	Segundos	Minutos	Min_Por_Km
2015-11-02	1.37	0:08:43	523.0	8.716667	6.362530
2015-11-03	5.30	0:39:26	2366.0	39.433333	7.440252
2015-11-04	3.05	0:22:12	1332.0	22.200000	7.278689
2015-11-07	5.63	0:49:05	2945.0	49.083333	8.718176
2015-11-07	1.26	NaN	NaN	NaN	NaN
2015-11-07	1.20	NaN	NaN	NaN	NaN
2015-11-09	3.05	0:22:36	1356.0	22.600000	7.409836
2015-11-10	1.50	0:09:00	540.0	9.000000	6.000000
2015-11-10	1.50	0:11:33	693.0	11.550000	7.700000
2015-11-10	4.00	NaN	NaN	NaN	NaN
2015-11-23	3.05	0:22:35	1355.0	22.583333	7.404372
2015-11-27	1.00	NaN	NaN	NaN	NaN
2015-11-28	3.00	NaN	NaN	NaN	NaN
2015-12-09	2.93	0:25:00	1500.0	25.000000	8.532423
2015-12-12	1.37	0:09:15	555.0	9.250000	6.751825
2015-12-13	5.30	0:43:36	2616.0	43.600000	8.226415
2015-12-13	2.21	0:18:59	1139.0	18.983333	8.589744
2015-12-15	6.50	0:58:43	3523.0	58.716667	9.033333
2015-12-17	12.00	1:39:00	5940.0	99.000000	8.250000
2015-12-19	3.10	0:26:15	1575.0	26.250000	8.467742
2015-12-20	14.80	2:15:00	8100.0	135.000000	9.121622
2015-12-22	3.10	0:28:00	1680.0	28.000000	9.032258
2015-12-23	5.63	0:51:50	3110.0	51.833333	9.206631
2015-12-24	3.05	0:25:08	1508.0	25.133333	8.240437
2015-12-25	14.80	2:20:00	8400.0	140.000000	9.459459
2015-12-26	5.20	NaN	NaN	NaN	NaN
2015-12-30	3.15	0:22:10	1330.0	22.166667	7.037037

In [36]:

```
df['2015-11':'2016-1-1']['Distancia'].plot()
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a24d1ee48>
```



Exibindo Gráficos de um range de dados

In [37]:

```
df.loc['2015-8-12']
```

Out[37]:

Distancia	1.42
Tempo	0:10:35
Segundos	635
Minutos	10.5833
Min_Por_Km	7.45305
Name:	2015-08-12 00:00:00, dtype: object

In [38]:

```
df.to_csv('data/dataframe_saved_v2.csv')
```

In [39]:

df.reset_index()

Volto o index para que eu

out[39]:

	Data	Distancia	Tempo	Segundos	Minutos	Min_Por_Km
0	2015-08-02	1.70	NaN	NaN	NaN	NaN
1	2015-08-03	1.40	NaN	NaN	NaN	NaN
2	2015-08-04	2.10	NaN	NaN	NaN	NaN
3	2015-08-06	2.38	NaN	NaN	NaN	NaN
4	2015-08-07	2.30	NaN	NaN	NaN	NaN
5	2015-08-08	3.40	NaN	NaN	NaN	NaN
6	2015-08-09	2.50	NaN	NaN	NaN	NaN
7	2015-08-10	3.36	0:28:37	1717.0	28.616667	8.516865
8	2015-08-11	1.67	0:13:07	787.0	13.116667	7.854291
9	2015-08-12	1.42	0:10:35	635.0	10.583333	7.453052
10	2015-08-13	2.35	0:17:25	1045.0	17.416667	7.411348
11	2015-08-14	0.90	0:06:16	376.0	6.266667	6.962963
12	2015-08-14	0.90	0:06:16	376.0	6.266667	6.962963
13	2015-08-15	1.78	NaN	NaN	NaN	NaN
14	2015-08-16	0.65	NaN	NaN	NaN	NaN
15	2015-08-17	1.34	NaN	NaN	NaN	NaN
16	2015-08-18	0.65	NaN	NaN	NaN	NaN
17	2015-08-19	1.37	NaN	NaN	NaN	NaN
18	2015-08-19	1.37	NaN	NaN	NaN	NaN
19	2015-08-20	1.37	0:09:33	573.0	9.550000	6.970803
20	2015-08-21	1.37	0:09:24	564.0	9.400000	6.861314
21	2015-08-22	1.37	0:09:05	545.0	9.083333	6.630170
22	2015-08-23	1.34	NaN	NaN	NaN	NaN
23	2015-08-24	3.00	NaN	NaN	NaN	NaN
24	2015-08-25	1.37	0:08:55	535.0	8.916667	6.508516
25	2015-08-25	1.34	NaN	NaN	NaN	NaN
26	2015-08-26	1.37	0:10:30	630.0	10.500000	7.664234
27	2015-08-27	2.50	0:25:00	1500.0	25.000000	10.000000
28	2015-08-29	5.00	0:43:27	2607.0	43.450000	8.690000
29	2015-09-02	2.00	NaN	NaN	NaN	NaN
...
51	2015-10-25	2.00	NaN	NaN	NaN	NaN
52	2015-10-31	2.01	NaN	NaN	NaN	NaN

	Data	Distancia	Tempo	Segundos	Minutos	Min_Por_Km
53	2015-11-02	1.37	0:08:43	523.0	8.716667	6.362530
54	2015-11-03	5.30	0:39:26	2366.0	39.433333	7.440252
55	2015-11-04	3.05	0:22:12	1332.0	22.200000	7.278689
56	2015-11-07	5.63	0:49:05	2945.0	49.083333	8.718176
57	2015-11-07	1.26	Nan	Nan	Nan	Nan
58	2015-11-07	1.20	Nan	Nan	Nan	Nan
59	2015-11-09	3.05	0:22:36	1356.0	22.600000	7.409836
60	2015-11-10	1.50	0:09:00	540.0	9.000000	6.000000
61	2015-11-10	1.50	0:11:33	693.0	11.550000	7.700000
62	2015-11-10	4.00	Nan	Nan	Nan	Nan
63	2015-11-23	3.05	0:22:35	1355.0	22.583333	7.404372
64	2015-11-27	1.00	Nan	Nan	Nan	Nan
65	2015-11-28	3.00	Nan	Nan	Nan	Nan
66	2015-12-09	2.93	0:25:00	1500.0	25.000000	8.532423
67	2015-12-12	1.37	0:09:15	555.0	9.250000	6.751825
68	2015-12-13	5.30	0:43:36	2616.0	43.600000	8.226415
69	2015-12-13	2.21	0:18:59	1139.0	18.983333	8.589744
70	2015-12-15	6.50	0:58:43	3523.0	58.716667	9.033333
71	2015-12-17	12.00	1:39:00	5940.0	99.000000	8.250000
72	2015-12-19	3.10	0:26:15	1575.0	26.250000	8.467742
73	2015-12-20	14.80	2:15:00	8100.0	135.000000	9.121622
74	2015-12-22	3.10	0:28:00	1680.0	28.000000	9.032258
75	2015-12-23	5.63	0:51:50	3110.0	51.833333	9.206631
76	2015-12-24	3.05	0:25:08	1508.0	25.133333	8.240437
77	2015-12-25	14.80	2:20:00	8400.0	140.000000	9.459459
78	2015-12-26	5.20	Nan	Nan	Nan	Nan
79	2015-12-30	3.15	0:22:10	1330.0	22.166667	7.037037
80	2016-01-01	26.20	4:20:43	15643.0	260.716667	9.951018

81 rows × 6 columns

FIM

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

Data Science Academy

Big Data Real-Time Analytics com Python e Spark

Capítulo 7

*** Atenção: ***

Utilize Java JDK 11 e Apache Spark 2.4.2

Variável de ambiente
~~Python3~~ → python ok

Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook

no prompt de comando ir ate a pasta onde estao os arquivos jupyter e digitar **pyspark** isso vai abrir o jupyter notebook

Introdução ao PySpark

In [1]:

```
import sys
print(sys.version)

3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)]
```

In [2]:

```
print(sc)

<SparkContext master=local[*] appName=PySparkShell>
```

In [3]:

```
print(sc.version)
```

2.4.2

In [4]:

```
# Testando o Spark e criando uma RDD
lst = [25, 90, 81, 37, 776, 3320]
testData = sc.parallelize(lst)
```

In [5]:

`?sc.parallelize`

In [6]:

`type(testData)`

Out[6]:

`pyspark.rdd.RDD`

In [7]:

`testData.count()`

Out[7]:

6

In [8]:

`testData.collect()`

Out[8]:

[25, 90, 81, 37, 776, 3320]

Executando Aplicação PySpark

RDD's são coleções distribuídas de itens. RDD's podem ser criadas a partir do Hadoop (arquivos no HDFS), através da transformação de outras RDD's, a partir de bancos de dados (relacionais e não-relacionais) ou a partir de arquivos locais.

RDD = distribuídos em cluster

In [9]:

```
# Criando uma RDD a partir de um arquivo csv
sentimentoRDD = sc.textFile("data/sentimentos.csv")
```

Só o fato de chamar a função e passar um arquivo já cria uma RDD

In [10]:

`type(sentimentoRDD)`

Out[10]:

`pyspark.rdd.RDD`

In [11]:

```
# Ação - Contando o número de registros
sentimentoRDD.count()
```

Out[11]:

100

In [12]:

```
# Listando os 5 primeiros registros
sentimentoRDD.take(5)
```

→ 5 frases

Out[12]:

```
['positivo,Esse livro é incrível.',  

 'positivo,Um dos melhores livros que eu já li.',  

 'positivo,um dos melhores livros que eu já li',  

 'positivo,Acho que ele tem um conteúdo que vai além do que está em sua descrição.',  

 'positivo,O Sol é para todos é profundo e emocionante']
```

RDD é imutável

In [13]:

```
# Transformando os dados - transformação para Letras maiúsculas
transfRDD = sentimentoRDD.map(lambda x : x.upper())
transfRDD.take(5)
```

Out[13]:

```
['POSITIVO,ESSE LIVRO É INCRÍVEL.',  

 'POSITIVO,UM DOS MELHORES LIVROS QUE EU JÁ LI.',  

 'POSITIVO,UM DOS MELHORES LIVROS QUE EU JÁ LI',  

 'POSITIVO,ACHO QUE ELE TEM UM CONTEÚDO QUE VAI ALÉM DO QUE ESTÁ EM SUA DESCRIÇÃO.',  

 'POSITIVO,O SOL É PARA TODOS É PROFUNDO E EMOCIONANTE']
```

In [14]:

```
sentimentoRDD.take(5)
```

Out[14]:

```
['positivo,Esse livro é incrível.',  

 'positivo,Um dos melhores livros que eu já li.',  

 'positivo,um dos melhores livros que eu já li',  

 'positivo,Acho que ele tem um conteúdo que vai além do que está em sua descrição.',  

 'positivo,O Sol é para todos é profundo e emocionante']
```

In [15]:

```
arquivo = sc.textFile("data/sentimentos.csv")
```

In [16]:

```
type(arquivo)
```

Out[16]:

```
pyspark.rdd.RDD
```

In [17]:

```
arquivo.count()
```

Out[17]:

```
100
```

In [18]:

```
arquivo.first()
```

Out[18]:

```
'positivo,Esse livro é incrivel.'
```

In [19]:

```
linhasComSol = arquivo.filter(lambda line: "Sol" in line)
```

In [20]:

```
type(linhasComSol)
```

Out[20]:

```
pyspark.rdd.PipelinedRDD
```

In [21]:

```
linhasComSol.count()
```

Out[21]:

```
3
```

Primeiro a função map() determina o comprimento de cada linha do arquivo, criando uma RDD. A função reduce() é chamada para encontrar a linha com maior número de caracteres. O argumento para as funções map() e reduce() são funções anônimas criadas com lambda (da linguagem Python).

In [22]:

```
arquivo.map(lambda line: len(line.split())).reduce(lambda a, b: a if (a > b) else b)
```

Out[22]:

```
27
```

Esta linha pode ser reescrita da seguinte forma:

In [23]:

```
def max(a, b):
    if a > b:
        return a
    else:
        return b
```

Recebe os linhas

arquivo.map(lambda line: len(line.split())).reduce(max)

chama a função max

Out[23]:

27

contém as linhas

Operação de MapReduce

As operações de **MapReduce** foram popularizadas pelo Hadoop e podem ser feitas com **Spark** até 100x mais rápido.

In [24]:

usado para localizar padrões

```
contaPalavras = arquivo.flatMap(lambda line: line.split()).map(lambda palavra: (palavra, 1))
```

Divisão de linhas

In [25]: Arquivo RDD

execução de jobs

localhost: 4040

```
contaPalavras.collect()
```

'em', 1),
('descrição.', 1),
('positivo,O', 2),
('para', 5),
('todos', 4),
('positivo,Me', 1),
('este', 1),
('livro,', 1),
('antigo', 1),
('uma', 4),
('história', 1),
('antiga', 1),
('positivo,The', 6),
('Da', 38),
('Vinci', 45),
('Code', 24),
('is', 17),
('good', 3),
('movie...', 1),

Acesse o monitoramento do Spark em: <http://localhost:4040/> (<http://localhost:4040/>)

Fim

Obrigado - Data Science Academy - facebook.com/dsacademybr

Data Science Academy

Se você não precisa de um cluster
você não precisa do spark
mapamento e redução.

Big Data Real-Time Analytics com Python e Spark

Capítulo 7

*** Atenção: ***

Utilize Java JDK 11 e Apache Spark 2.4.2

Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook

Acesse <http://localhost:4040> (<http://localhost:4040>) sempre que quiser acompanhar a execução dos jobs

RDD = Resilient Distributed Dataset

RDD quase igual as Pandas mas é feito para rodar em sistema distribuído

Transformações

In [1]:

```
# Criando uma lista em Python
lista1 = [124, 901, 652, 102, 397]
```

In [2]:

```
type(lista1)
```

lista em Python

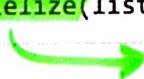
Out[2]:

```
list
```

Não é um obj dist. bndido

In [3]:

```
# Carregando dados de uma coleção
lstRDD = sc.parallelize(lista1)
```



convertendo a lista pt o obj dist. bndido

In [4]:

```
type(lstRDD)
```

RDD

Out[4]:

```
pyspark.rdd.RDD
```

é um obj distribuído

In [5]:

1stRDD.collect()

funcão para o head()

Out[5]:

[124, 901, 652, 102, 397]

In [6]:

1stRDD.count()

Out[6]:

5

In [7]:

Carregando um arquivo e criando um RDD.
autoDataRDD = sc.textFile("data/carros.csv")

In [8]:

type(autoDataRDD)

Out[8]:

pyspark.rdd.RDD

Quando carrega um arquivo por um
TextFile já crie um RDD

In [9]:

Operação de Ação
autoDataRDD.first()

Out[9]:

'MAKE,FUELTYPE,ASPIRE,DOORS,BODY,DRIVE,CYLINDERS,HP,RPM,MPG-CITY,MPG-HWY,PRI
CE'

In [10]:

autoDataRDD.take(5)

Retorno os 5 primeiros
linhos

Out[10]:

['MAKE,FUELTYPE,ASPIRE,DOORS,BODY,DRIVE,CYLINDERS,HP,RPM,MPG-CITY,MPG-HWY,PR
ICE',
'subaru,gas,std,two,hatchback,fwd,four,69,4900,31,36,5118',
'chevrolet,gas,std,two,hatchback,fwd,three,48,5100,47,53,5151',
'mazda,gas,std,two,hatchback,fwd,four,68,5000,30,31,5195',
'toyota,gas,std,two,hatchback,fwd,four,62,4800,35,39,5348']

In [11]:

```
# Cada ação gera um novo processo de computação dos dados.
# Mas podemos persistir os dados em cache para que ele possa ser usado por outras ações, se
# de nova computação.
autoDataRDD.cache()
```

Out[11]:

```
data/carros.csv MapPartitionsRDD[3] at textFile at NativeMethodAccessorImpl.
java:0
```

In [12]:

```
for line in autoDataRDD.collect():
    print(line)
```

Percorre e imprime todo o conjunto de dados

```
MAKE,FUELTYPE,ASPIRE,DOORS,BODY,DRIVE,CYLINDERS,HP,RPM,MPG-CITY,MPG-HWY,PR
ICE
subaru,gas,std,two,hatchback,fwd,four,69,4900,31,36,5118
chevrolet,gas,std,two,hatchback,fwd,three,48,5100,47,53,5151
mazda,gas,std,two,hatchback,fwd,four,68,5000,30,31,5195
toyota,gas,std,two,hatchback,fwd,four,62,4800,35,39,5348
mitsubishi,gas,std,two,hatchback,fwd,four,68,5500,37,41,5389
honda,gas,std,two,hatchback,fwd,four,60,5500,38,42,5399
nissan,gas,std,two,sedan,fwd,four,69,5200,31,37,5499
dodge,gas,std,two,hatchback,fwd,four,68,5500,37,41,5572
plymouth,gas,std,two,hatchback,fwd,four,68,5500,37,41,5572
mazda,gas,std,two,hatchback,fwd,four,68,5000,31,38,6095
mitsubishi,gas,std,two,hatchback,fwd,four,68,5500,31,38,6189
dodge,gas,std,two,hatchback,fwd,four,68,5500,31,38,6229
plymouth,gas,std,four,hatchback,fwd,four,68,5500,31,38,6229
chevrolet,gas,std,two,hatchback,fwd,four,70,5400,38,43,6295
toyota,gas,std,two,hatchback,fwd,four,62,4800,31,38,6338
dodge,gas,std,two,hatchback,fwd,four,68,5500,31,38,6377
honda,gas,std,two,hatchback,fwd,four,58,4800,49,54,6479
```

In [13]:

```
# Map() é criação de um novo RDD - Transformação - Lazy Evaluation
tsvData = autoDataRDD.map(lambda x : x.replace(",","\t"))
tsvData.take(5)
```

Out[13]:

Substitui ; por \t

```
['MAKE\tFUELTYPE\tASPIRE\tDOORS\tBODY\tDRIVE\tCYLINDERS\tHP\tRPM\tMPG-CITY\t
MPG-HWY\tPRICE',
'subaru\tgas\tstd\ttwo\thatchback\tfwd\tfour\t69\t4900\t31\t36\t5118',
'chevrolet\tgas\tstd\ttwo\thatchback\tfwd\tthree\t48\t5100\t47\t53\t5151',
'mazda\tgas\tstd\ttwo\thatchback\tfwd\tfour\t68\t5000\t30\t31\t5195',
'toyota\tgas\tstd\ttwo\thatchback\tfwd\tfour\t62\t4800\t35\t39\t5348']
```

In [14]:

autoDataRDD.first()

Out[14]:

'MAKE,FUELTYPE,ASPIRE,DOORS,BODY,DRIVE,CYLINDERS,HP,RPM,MPG-CITY,MPG-HWY,PRI
CE'

In [15]:

Filter() e criação de um novo RDD - Transformação - Lazy Evaluation
toyotaData = autoDataRDD.filter(lambda x: "toyota" in x)

In [16]:

busca palavras toyota

Ação

toyotaData.count()

conta o novo RDD que contém os registros
toyota

Out[16]:

32

Ações

In [17]:

Ação

toyotaData.take(20)

Out[17]:

```
[ 'toyota,gas,std,two,hatchback,fwd,four,62,4800,35,39,5348',
  'toyota,gas,std,two,hatchback,fwd,four,62,4800,31,38,6338',
  'toyota,gas,std,four,hatchback,fwd,four,62,4800,31,38,6488',
  'toyota,gas,std,four,wagon,fwd,four,62,4800,31,37,6918',
  'toyota,gas,std,four,sedan,fwd,four,70,4800,30,37,6938',
  'toyota,gas,std,four,hatchback,fwd,four,70,4800,30,37,7198',
  'toyota,gas,std,four,sedan,fwd,four,70,4800,38,47,7738',
  'toyota,diesel,std,four,hatchback,fwd,four,56,4500,38,47,7788',
  'toyota,gas,std,four,wagon,4wd,four,62,4800,27,32,7898',
  'toyota,diesel,std,four,sedan,fwd,four,56,4500,34,36,7898',
  'toyota,gas,std,two,sedan,rwd,four,70,4800,29,34,8058',
  'toyota,gas,std,two,hatchback,rwd,four,70,4800,29,34,8238',
  'toyota,gas,std,four,hatchback,fwd,four,70,4800,28,34,8358',
  'toyota,gas,std,two,hardttop,rwd,four,116,4800,24,30,8449',
  'toyota,gas,std,four,wagon,4wd,four,62,4800,27,32,8778',
  'toyota,gas,std,four,sedan,fwd,four,92,4200,29,34,8948',
  'toyota,gas,std,four,sedan,fwd,four,70,4800,28,34,9258',
  'toyota,gas,std,two,sedan,rwd,four,112,6600,26,29,9298',
  'toyota,gas,std,two,hatchback,rwd,four,112,6600,26,29,9538',
  'toyota,gas,std,two,hardttop,rwd,four,116,4800,24,30,9639 ]
```

In [18]:

Pode salvar o conjunto de dados, o RDD.

Nesse caso, o Spark solicita os dados ao processo Master e então gera um arquivo de saída

savedRDD = open("data/carros_v2.csv","w")

savedRDD.write("\n".join(autoDataRDD.collect()))

savedRDD.close()

Operações Set

In [19]:

```
# Set operations
palavras1 = sc.parallelize(["Big Data", "Data Science", "Analytics", "Visualization"])
palavras2 = sc.parallelize(["Big Data", "R", "Python", "Scala"])
```

↳ Cria um RDD

In [20]:

```
# União
for unions in palavras1.union(palavras2).distinct().collect():
    print(unions)
```

Data Science
R
Analytics
Visualization
Scala
Big Data
Python

União Palavras 1
e Palavras 2

collect retorna o resultado

Sem repetição

In [21]:

```
# Intersecção
for intersects in palavras1.intersection(palavras2).collect():
    print(intersects)*
```

Palavras que são comuns aos dois RDD

Big Data

In [22]:

```
rdd01 = sc.parallelize(range(1,10))
rdd02 = sc.parallelize(range(10,21))
rdd01.union(rdd02).collect()
```

Out[22]:

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

In [23]:

```
rdd01 = sc.parallelize(range(1,10))
rdd02 = sc.parallelize(range(5,15))
rdd01.intersection(rdd02).collect()
```

Out[23]:

[5, 6, 7, 8, 9]

Left/Right Outer Join

In [24]:

```
names1 = sc.parallelize(("banana", "uva", "laranja")).map(lambda a: (a, 1))
names2 = sc.parallelize(("laranja", "abacaxi", "manga")).map(lambda a: (a, 1))
names1.join(names2).collect()
```

Out[24]: operações que tinham em comum entre os dois
[('laranja', (1, 1))]

In [25]:

```
names1.leftOuterJoin(names2).collect()
```

Out[25]:

[('uva', (1, None)), ('banana', (1, None)), ('laranja', (1, 1))]

In [26]:

```
names1.rightOuterJoin(names2).collect()
```

Out[26]:

[('manga', (None, 1)), ('laranja', (1, 1)), ('abacaxi', (None, 1))]

■ Distinct

In [27]:

```
# Distinct
lista1 = [124, 901, 652, 102, 397, 124, 901, 652]
lstRDD = sc.parallelize(lista1)
for numbData in lstRDD.distinct().collect():
    print(numbData)
```

voltar valores únicos sem repetição.

124
652
901
397
102

Transformação e Limpeza

In [28]:

```
# RDD Original
autoDataRDD.collect()
```

Out[28]:

```
[ 'MAKE,FUELTYPE,ASPIRE,DOORS,BODY,DRIVE,CYLINDERS,HP,RPM,MPG-CITY,MPG-HWY,
PRICE',
'subaru,gas,std,two,hatchback,fwd,four,69,4900,31,36,5118',
'chevrolet,gas,std,two,hatchback,fwd,three,48,5100,47,53,5151',
'mazda,gas,std,two,hatchback,fwd,four,68,5000,30,31,5195',
'toyota,gas,std,two,hatchback,fwd,four,62,4800,35,39,5348',
'mitsubishi,gas,std,two,hatchback,fwd,four,68,5500,37,41,5389',
'honda,gas,std,two,hatchback,fwd,four,60,5500,38,42,5399',
'nissan,gas,std,two,sedan,fwd,four,69,5200,31,37,5499',
'dodge,gas,std,two,hatchback,fwd,four,68,5500,37,41,5572',
'plymouth,gas,std,two,hatchback,fwd,four,68,5500,37,41,5572',
'mazda,gas,std,two,hatchback,fwd,four,68,5000,31,38,6095',
'mitsubishi,gas,std,two,hatchback,fwd,four,68,5500,31,38,6189',
'dodge,gas,std,four,hatchback,fwd,four,68,5500,31,38,6229',
'plymouth,gas,std,four,hatchback,fwd,four,68,5500,31,38,6229',
'chevrolet,gas,std,two,hatchback,fwd,four,70,5400,38,43,6295',
'toyota,gas,std,two,hatchback,fwd,four,62,4800,31,38,6338',
'dodge,gas,std,two,hatchback,fwd,four,68,5500,31,38,6377']
```

In [29]:

```
# Transformação e Limpeza
def LimpaRDD(autoStr) :

    # Verifica a indexação
    if isinstance(autoStr, int) :
        return autoStr

    # Separa cada índice pela vírgula (separador de colunas)
    attList = autoStr.split(",")

    # Converte o número de portas para um num
    if attList[3] == "two" :
        attList[3] = "2"
    elif attList[3] == "four" :
        attList[3] = "4"

    # Converte para uppercase
    attList[5] = attList[4].upper()

    return ",".join(attList)
```

OCRESCENTA UMA NOVA COLUNA

In [30]:

```
# Transformação
RDD_limpo = autoDataRDD.map(LimpaRDD)
```

In [31]:

```
print(RDD_limpo)
```

PythonRDD[73] at RDD at PythonRDD.scala:53

In [32]:

```
# Ação
RDD_limpo.collect()
```

Out[32]:

```
['MAKE', 'FUELTYPE', 'ASPIRE', 'DOORS', 'BODY', 'CYLINDERS', 'HP', 'RPM', 'MPG-CITY', 'MPG-HWY', 'P  
RICE',  
'subaru,gas,std,2,hatchback,HATCHBACK,four,69,4900,31,36,5118',  
'chevrolet,gas,std,2,hatchback,HATCHBACK,three,48,5100,47,53,5151',  
'mazda,gas,std,2,hatchback,HATCHBACK,four,68,5000,30,31,5195',  
'toyota,gas,std,2,hatchback,HATCHBACK,four,62,4800,35,39,5348',  
'mitsubishi,gas,std,2,hatchback,HATCHBACK,four,68,5500,37,41,5389',  
'honda,gas,std,2,hatchback,HATCHBACK,four,60,5500,38,42,5399',  
'nissan,gas,std,2,sedan,SEDAN,four,69,5200,31,37,5499',  
'dodge,gas,std,2,hatchback,HATCHBACK,four,68,5500,37,41,5572',  
'plymouth,gas,std,2,hatchback,HATCHBACK,four,68,5500,37,41,5572',  
'mazda,gas,std,2,hatchback,HATCHBACK,four,68,5000,31,38,6095',  
'mitsubishi,gas,std,2,hatchback,HATCHBACK,four,68,5500,31,38,6189',  
'dodge,gas,std,4,hatchback,HATCHBACK,four,68,5500,31,38,6229',  
'plymouth,gas,std,4,hatchback,HATCHBACK,four,68,5500,31,38,6229',  
'chevrolet,gas,std,2,hatchback,HATCHBACK,four,70,5400,38,43,6295',  
'toyota,gas,std,2,hatchback,HATCHBACK,four,62,4800,31,38,6338',  
'dodge,gas,std,2,hatchback,HATCHBACK,four,68,5500,31,38,6377'.
```

Apenas quando chama a ação que executa todas as funções

Ações

In [33]:

```
# reduce() - soma de valores
```

```
lista2 = [124, 901, 652, 102, 397, 124, 901, 652] → cria lista em Python
```

```
lstRDD = sc.parallelize(lista2) → converte em RDD
```

```
lstRDD.collect() → ação que imprime resultado
```

```
lstRDD.reduce(lambda x,y: x + y)
```

Out[33]:

Recebe x e y e soma os dois nº e ter chagar a um único nº

3853

In [34]:

```
# Encontrando a linha com menor número de caracteres
```

```
autoDataRDD.reduce(lambda x,y: x if len(x) < len(y) else y)
```

Out[34]:

bmw,gas,std,two,sedan,rwd,six,182,5400,16,22,41315' Se não retorna x

In [35]:

```
# Criando uma função para redução
def getMPG(autoStr) :
    indice
    if isinstance(autoStr, int) :
        return autoStr
    attList = autoStr.split(",")
    if attList[9].isdigit() :
        return int(attList[9])
    else:
        return 0
```

In [36]:

```
# Encontrando a média de MPG para todos os carros
media_mpg = round(autoDataRDD.reduce(lambda x,y : getMPG(x) + getMPG(y)) / (autoDataRDD.co
print(media_mpg)
```

25.15 RDD

Pego apenas 1 atributo, milhar por ponto
Posição 9 MPG-CITY
Nº com 2 dígitos -1
Retorno o cabeçalho

In [38]:

```
times = sc.parallelize(["Flamengo", "Vasco", "Botafogo", "Fluminense", "Palmeiras", "Bahia"]
times.takeSample(True, 3)
```

Out[38]: Retorno 1 amostra
['Fluminense', 'Bahia', 'Botafogo']

In [39]:

```
times = sc.parallelize(["Flamengo", "Vasco", "Botafogo", "Fluminense", "Palmeiras", "Bahia"]
times.map(lambda k: (k,1)).countByKey().items()
```

Out[39]: Transforma em chave e valor
Processa tem chave e valor
dict_items([('Flamengo', 2), ('Vasco', 1), ('Botafogo', 1), ('Fluminense', 1), ('Palmeiras', 1), ('Bahia', 2)])

In [40]:

```
autoDataRDD.saveAsTextFile("data/autoDataRDD.txt")
```

Salvando o RDD
tudo é feito em memória

Fim

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

Data Science Academy

Big Data Real-Time Analytics com Python e Spark

Capítulo 7

*** Atenção: ***

Utilize Java JDK 11 e Apache Spark 2.4.2

Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook

Mini-Projeto 1 - Analisando Dados do Uber com Spark

Dataset: <https://github.com/fivethirtyeight/uber-tlc-foil-response> (<https://github.com/fivethirtyeight/uber-tlc-foil-response>)

Esse conjunto de dados contém dados de mais de 4,5 milhões de captações Uber na cidade de Nova York de abril a setembro de 2014 e 14,3 milhões de captações Uber de janeiro a junho de 2015. Dados em nível de viagem sobre 10 outras empresas de veículos de aluguel (FHV) bem como dados agregados para 329 empresas de FHV, também estão incluídos. Todos os arquivos foram recebidos em 3 de agosto, 15 de setembro e 22 de setembro de 2015.

- 1- Quantos são e quais são as bases de carros do Uber (onde os carros ficam esperando passageiros)?
- 2- Qual o total de veículos que passaram pela base B02617?
- 3- Qual o total de corridas por base? Apresente de forma decrescente.

In [1]:

```
from pandas import read_csv
```

In [2]:

```
# Criando um objeto Pandas
uberFile = read_csv("data/uber.csv")
```

In [3]:

type(uberFile)

Out[3]:

pandas.core.frame.DataFrame

In [4]:

Visualizando as primeiras linhas

uberFile.head(10)

Out[4]:

Data Frame Python

função python

	dispatching_base_number	date	active_vehicles	trips
0	B02512	1/1/2015	190	1132
1	B02765	1/1/2015	225	1765
2	B02764	1/1/2015	3427	29421
3	B02682	1/1/2015	945	7679
4	B02617	1/1/2015	1228	9537
5	B02598	1/1/2015	870	6903
6	B02598	1/2/2015	785	4768
7	B02617	1/2/2015	1137	7065
8	B02512	1/2/2015	175	875
9	B02682	1/2/2015	890	5506

In [5]:

Transformando o dataframe (Pandas) em um Dataframe (Spark)

uberDF = sqlContext.createDataFrame(uberFile)

In [6]:

converte pr DataFrame spark

type(uberDF)

Out[6]:

pyspark.sql.dataframe.DataFrame

In [7]:

Criando o RDD a partir do arquivo csv

uberRDD = sc.textFile("data/uber.csv")

In [8]:

Cria RDD

type(uberRDD)

Out[8]:

pyspark.rdd.RDD

In [9]:

```
# Total de registros
uberRDD.count()
```

Out[9]:

355

In [10]:

```
# Primeiro registro
uberRDD.first()
```

Out[10]:

'dispatching_base_number,date,active_vehicles,trips'

In [11]:

```
# Dividindo o arquivo em colunas, separadas pelo caracter ","
uberLinhas = uberRDD.map(lambda line: line.split(","))
```

In [12]:

type(uberLinhas)

Out[12]:

pyspark.rdd.PipelinedRDD

In [13]:

```
# Número de bases de carros do Uber
uberLinhas.map(lambda linha: linha[0]).distinct().count() -1
```

Out[13]:

6 transformações

Recebe linha como
entroada

contagem

Retorna o 1º linha
coleção

distintos
valores únicos
não repetidos

In [14]:

Bases de carros do Uber

uberLinhas.map(lambda linha: linha[0]).distinct().collect()

Out[14]:

```
['dispatching_base_number',
 'B02765',
 'B02682',
 'B02598',
 'B02512',
 'B02764',
 'B02617']
```

Exibe a lista de
bases de carros de forma
distinta

Retorna linha
na posição 0
apenas 1ª coluna

In [15]:

```
# Total de veículos que passaram pela base B02617
uberLinhas.filter(lambda linha: "B02617" in linha).count()
```

Out[15]:

59
 Filtros informações
 do Uber linhas

In [16]:

```
# Gravando os dados dos veículos da base B02617 em um novo RDD
b02617_RDD = uberLinhas.filter(lambda linha: "B02617" in linha)
```

In [17]:

```
# Total de dias em que o número de corridas foi superior a 16.000
b02617_RDD.filter(lambda linha: int(linha[3]) > 16000).count()
```

Out[17]:

4
 dias que 16.000

In [18]:

```
# Dias em que o total de corridas foi superior a 16.000
b02617_RDD.filter(lambda linha: int(linha[3]) > 16000).collect()
```

Out[18]:

```
[['B02617', '2/13/2015', '1590', '16996'],
 ['B02617', '2/14/2015', '1486', '16999'],
 ['B02617', '2/20/2015', '1574', '16856'],
 ['B02617', '2/21/2015', '1443', '16098']]
```

In [19]:

```
# Criando um novo RDD
uberRDD2 = sc.textFile("data/uber.csv").filter(lambda line: "base" not in line).map(lambda
```

In [20]:

```
# Aplicando redução para calcular o total por base
uberRDD2.map(lambda kp: (kp[0], int(kp[3]))).reduceByKey(lambda k,v: k + v).collect()
```

Out[20]: corridas por base

```
[('B02765', 193670),
 ('B02682', 662509),
 ('B02598', 540791),
 ('B02512', 93786),
 ('B02764', 1914449),
 ('B02617', 725025)]
```

mapendo

coluna 3

In [21]:

```
# Aplicando redução para calcular o total por base, em ordem decrescente
uberRDD2.map(lambda kp: (kp[0], int(kp[3]))) .reduceByKey(lambda k,v: k + v).takeOrdered(10)
```

Out[21]:

```
[('B02764', 1914449),
 ('B02617', 725025),
 ('B02682', 662509),
 ('B02598', 540791),
 ('B02765', 193670),
 ('B02512', 93786)]
```

Fim

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

Data Science Academy

Big Data Real-Time Analytics com Python e Spark

Capítulo 8

*** Atenção: ***

*Alguns funcionários não são compatíveis
com o Java !!*

Utilize **Java JDK 1.8** e Apache Spark 2.4.2

Java JDK 1.8:

Redshift banco de dados no nuvem Amazon

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
[\(https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html\)](https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html)

Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook

Spark SQL

O Spark SQL é usado para acessar dados estruturados com Spark.

Acesse <http://localhost:4040> (<http://localhost:4040>) sempre que quiser acompanhar a execução dos jobs.

Pacotes adicionais podem ser encontrados aqui: <https://spark-packages.org/> (<https://spark-packages.org/>) (usaremos um destes pacotes para conexão com o MongoDB).

Spark SQL - Spark Session e SQL Context

In [1]:

```
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
from pyspark.sql import Row
```

In [2]:

```
print(sc)
```

crie o Session Spark

```
<SparkContext master=local[*] appName=PySparkShell>
```

In [3]:

```
# Spark Session - usado para trabalhar com o Spark
spSession = SparkSession.builder.master("local").appName("DSA-SparkSQL").getOrCreate()
```

In [4]:

```
# Criando o SQL Context para trabalhar com Spark SQL
sqlContext = SQLContext(sc)
```

In [5]:

```
# Importando o arquivo e criando um RDD
linhasRDD1 = sc.textFile("data/carros.csv")
```

App Name

Nome do session

Spark context

Spark SQL

→ Session

→ SQL context

→ Dataframe

Se a session já existir
Pego a session existente
e crio uma nova session

In [6]:

linhasRDD1.count()

As três linhas acima são necessárias
para criar a conexão SQL

Out[6]:

198 linhas do RDD

In [7]:

```
# Removendo a primeira linha - Transformação 1
linhasRDD2 = linhasRDD1.filter(lambda x: "FUELTYPE" not in x)
```

In [8]:

linhasRDD2.count()

Retorno todos os linhos que não tem
Fueltype
ou seja, retira o cabeçalho

Out[8]:

197 → 1 linha o menos
cabeçalho

In [9]:

```
# Dividindo o conjunto de dados em colunas - Transformação 2
linhasRDD3 = linhasRDD2.map(lambda line: line.split(","))
```

In [10]:

```
# Dividindo o conjunto de dados em colunas - Transformação 3
linhasRDD4 = linhasRDD3.map(lambda p: Row(make = p[0], body = p[4], hp = int(p[7])))
```

In [11]:

print(linhasRDD4)

Função SQL, define dinamicamente cada
linha
RDD aceita qualquer tipo de dado
função Row vai tentar dar uma
estrutura de Dataframe.

PythonRDD[4] at RDD at PythonRDD.scala:53

In [12]:

?Row

In [13]:

```
linhasRDD4.collect()
```

Out[13]:

```
[Row(body='hatchback', hp=69, make='subaru'),  
 Row(body='hatchback', hp=48, make='chevrolet'),  
 Row(body='hatchback', hp=68, make='mazda'),  
 Row(body='hatchback', hp=62, make='toyota'),  
 Row(body='hatchback', hp=68, make='mitsubishi'),  
 Row(body='hatchback', hp=60, make='honda'),  
 Row(body='sedan', hp=69, make='nissan'),  
 Row(body='hatchback', hp=68, make='dodge'),  
 Row(body='hatchback', hp=68, make='plymouth'),  
 Row(body='hatchback', hp=68, make='mazda'),  
 Row(body='hatchback', hp=68, make='mitsubishi'),  
 Row(body='hatchback', hp=68, make='dodge'),  
 Row(body='hatchback', hp=68, make='plymouth'),  
 Row(body='hatchback', hp=70, make='chevrolet'),  
 Row(body='hatchback', hp=62, make='toyota'),  
 Row(body='hatchback', hp=68, make='dodge'),  
 Row(body='hatchback', hp=58, make='honda'),  
 Row(body='hatchback', hp=62, make='toyota').
```

RDD

In [14]:

```
# Criando um dataframe a partir do RDD  
linhasDF = spSession.createDataFrame(linhasRDD4)
```

Spark
Session

Cria Dataframe

In [15]:

`linhasDF.show()`

```
+-----+-----+  
| body | hp | make |  
+-----+-----+  
| hatchback | 69 | subaru |  
| hatchback | 48 | chevrolet |  
| hatchback | 68 | mazda |  
| hatchback | 62 | toyota |  
| hatchback | 68 | mitsubishi |  
| hatchback | 60 | honda |  
| sedan | 69 | nissan |  
| hatchback | 68 | dodge |  
| hatchback | 68 | plymouth |  
| hatchback | 68 | mazda |  
| hatchback | 68 | mitsubishi |  
| hatchback | 68 | dodge |  
| hatchback | 68 | plymouth |  
| hatchback | 70 | chevrolet |  
| hatchback | 62 | toyota |  
| hatchback | 68 | dodge |  
| hatchback | 58 | honda |  
| hatchback | 62 | toyota |  
| hatchback | 76 | honda |  
| sedan | 70 | chevrolet |  
+-----+-----+  
only showing top 20 rows
```

*Data Frame
a part of
a RDD*

In [16]:

`type(linhasDF)`

Out[16]:

`pyspark.sql.dataframe.DataFrame`

In [17]:

```
# Mesma coisa que: SELECT * FROM LinhasDF  
linhasDF.select("*").show()
```

```
+-----+-----+  
| body | hp | make |  
+-----+-----+  
| hatchback | 69 | subaru |  
| hatchback | 48 | chevrolet |  
| hatchback | 68 | mazda |  
| hatchback | 62 | toyota |  
| hatchback | 68 | mitsubishi |  
| hatchback | 60 | honda |  
| sedan | 69 | nissan |  
| hatchback | 68 | dodge |  
| hatchback | 68 | plymouth |  
| hatchback | 68 | mazda |  
| hatchback | 68 | mitsubishi |  
| hatchback | 68 | dodge |  
| hatchback | 68 | plymouth |  
| hatchback | 70 | chevrolet |  
| hatchback | 62 | toyota |  
| hatchback | 68 | dodge |  
| hatchback | 58 | honda |  
| hatchback | 62 | toyota |  
| hatchback | 76 | honda |  
| sedan | 70 | chevrolet |  
+-----+  
only showing top 20 rows
```

In [18]:

Mesma coisa que: `SELECT * FROM LinhasDF ORDER BY make`
`linhasDF.orderBy("make").show()`

```
+---+---+-----+
| body| hp | make |
+---+---+-----+
| hatchback|154|alfa-romero|
| convertible|111|alfa-romero|
| convertible|111|alfa-romero|
| sedan|110| audi|
| wagon|110| audi|
| sedan|140| audi|
| sedan|110| audi|
| sedan|115| audi|
| sedan|102| audi|
| sedan|121| bmw|
| sedan|121| bmw|
| sedan|182| bmw|
| sedan|182| bmw|
| sedan|101| bmw|
| sedan|182| bmw|
| sedan|121| bmw|
| sedan|101| bmw|
| hatchback| 70| chevrolet|
| sedan| 70| chevrolet|
| hatchback| 48| chevrolet|
+---+---+-----+
```

only showing top 20 rows

In [19]:

Registrando o dataframe como uma Temp Table
`linhasDF.createOrReplaceTempView("linhasTB")`

tabela criada em memória

In [20]:

`!java -version`

```
java version "1.8.0_111"
Java(TM) SE Runtime Environment (build 1.8.0_111-b14)
Java HotSpot(TM) 64-Bit Server VM (build 25.111-b14, mixed mode)
```

In [21]:

```
# Executando queries SQL ANSI  
spSession.sql("select * from linhasTB where make = 'nissan'").show()
```

body	hp	make
sedan	69	nissan
sedan	69	nissan
sedan	69	nissan
sedan	55	nissan
sedan	69	nissan
wagon	69	nissan
sedan	69	nissan
hatchback	69	nissan
wagon	69	nissan
hardtop	69	nissan
hatchback	97	nissan
sedan	97	nissan
sedan	152	nissan
sedan	152	nissan
wagon	152	nissan
hatchback	160	nissan
hatchback	160	nissan
hatchback	200	nissan

Este comando só funciona com
tabelas temporárias

In [22]:

Executando queries SQL ANSI

spSession.sql("select make, body, avg(hp) from linhasTB group by make, body").show()

make	body	avg(hp)
nissan	wagon	96.66666666666667
subaru	sedan	90.2
plymouth	sedan	68.0
dodge	hatchback	90.2
nissan	sedan	89.0
honda	sedan	89.8
mitsubishi	hatchback	105.0
mazda	sedan	82.66666666666667
alfa-romero	convertible	111.0
mercedes-benz	convertible	155.0
plymouth	wagon	88.0
mercedes-benz	wagon	123.0
isuzu	hatchback	90.0
toyota	convertible	116.0
mazda	hatchback	89.4
chevrolet	sedan	70.0
mercury	hatchback	175.0
porsche	hatchback	143.0
honda	wagon	76.0
porsche	convertible	207.0

only showing top 20 rows

Spark SQL e Arquivos CSV

In [23]:

carrosDF = spSession.read.csv("data/carros.csv", header = True)

indica que é P/ importar
o cabeçalho

In [24]:

type(carrosDF)

Out[24]:

pyspark.sql.dataframe.DataFrame

Spark Session

In [25]:

carrosDF.show()

	MAKE	FUELTYPE	ASPIRE	DOORS	BODY	DRIVE	CYLINDERS	HP	RPM	MPG-CIT	Y	MPG-HWY	PRICE
1	subaru	gas	std	two	hatchback	fwd	four	69	4900	3	36	5118	
7	chevrolet	gas	std	two	hatchback	fwd	three	48	5100	4	53	5151	
0	mazda	gas	std	two	hatchback	fwd	four	68	5000	3	31	5195	
5	toyota	gas	std	two	hatchback	fwd	four	62	4800	3	39	5348	
7	mitsubishi	gas	std	two	hatchback	fwd	four	68	5500	3	41	5389	
8	honda	gas	std	two	hatchback	fwd	four	60	5500	3	42	5399	
1	nissan	gas	std	two	sedan	fwd	four	69	5200	3	37	5499	
7	dodge	gas	std	two	hatchback	fwd	four	68	5500	3	41	5572	
7	plymouth	gas	std	two	hatchback	fwd	four	68	5500	3	41	5572	
1	mazda	gas	std	two	hatchback	fwd	four	68	5000	3	38	6095	
1	mitsubishi	gas	std	two	hatchback	fwd	four	68	5500	3	38	6189	
1	dodge	gas	std	four	hatchback	fwd	four	68	5500	3	38	6229	
1	plymouth	gas	std	four	hatchback	fwd	four	68	5500	3	38	6229	
8	chevrolet	gas	std	two	hatchback	fwd	four	70	5400	3	43	6295	
1	toyota	gas	std	two	hatchback	fwd	four	62	4800	3	38	6338	
1	dodge	gas	std	two	hatchback	fwd	four	68	5500	3	38	6377	
9	honda	gas	std	two	hatchback	fwd	four	58	4800	4	54	6479	
1	toyota	gas	std	four	hatchback	fwd	four	62	4800	3	38	6488	
0	honda	gas	std	two	hatchback	fwd	four	76	6000	3	34	6529	
8	chevrolet	gas	std	four	sedan	fwd	four	70	5400	3	43	6575	

only showing top 20 rows

In [26]:

```
# Registrando o dataframe como uma Temp Table
carrosDF.createOrReplaceTempView("carrosTB")
```

criando tabela temporária

In [27]:

```
# Executando queries SQL ANSI
```

```
spSession.sql("select make, hp, price from carrosTB where CYLINDERS = 'three'").show()
```

make	hp	price
chevrolet	48	5151

In [28]:

Guardando o resultado em um novo objeto

```
carrosTT = spSession.sql("select make, hp, price from carrosTB where CYLINDERS = 'three'")
```

In [29]:

```
carrosTT.show()
```

make	hp	price
chevrolet	48	5151

Aplicando Machine Learning

In [30]:

congelo o dataframe e crie RDD

```
# Carregando o arquivo CSV e mantendo o objeto em cache
carros = sc.textFile("data/carros.csv")
carros.cache()
```

congelo o RDD em cache , persiste o RDD

Out[30]:

```
data/carros.csv MapPartitionsRDD[54] at textFile at NativeMethodAccessorImp
1.java:0
```

In [31]:

```
# Remove a primeira linha (header)
```

```
primeiraLinha = carros.first()
linhas = carros.filter(lambda x: x != primeiraLinha)
linhas.count()
```

Out[31]:

é linha do RDD

congelo tudo que for diferente da primeira linha.

transformação.

In [32]:

```
# Importando função row
from pyspark.sql import Row
```

In [33]:

```
# Convertendo para um vetor de linhas
def transformToNumeric(inputStr) :
    RDD Recebido
    attList = inputStr.split(",")
    Separa em colunas
    doors = 1.0 if attList[3] == "two" else 2.0 → doors Recebe 1.0 se List[3] = two
    body = 1.0 if attList[4] == "sedan" else 2.0
    Recebe 2.0 se não Recebe 2.0

    # Filtrando colunas não necessárias nesta etapa
    valores = Row(DOORS = doors, BODY = float(body), HP = float(attList[7]), RPM = float(attList[8]))
    return valores
```

In [34]:

```
# Aplicando a função aos dados e persistindo o resultado em memória
autoMap = linhas.map(transformToNumeric)
autoMap.persist()
autoMap.collect()
```

Out[34]:

RDD sem o cabeçalho

Função

Para manter na memória

```
[Row(BODY=2.0, DOORS=1.0, HP=69.0, MPG=31.0, RPM=4900.0),
Row(BODY=2.0, DOORS=1.0, HP=48.0, MPG=47.0, RPM=5100.0),
Row(BODY=2.0, DOORS=1.0, HP=68.0, MPG=30.0, RPM=5000.0),
Row(BODY=2.0, DOORS=1.0, HP=62.0, MPG=35.0, RPM=4800.0),
Row(BODY=2.0, DOORS=1.0, HP=68.0, MPG=37.0, RPM=5500.0),
Row(BODY=2.0, DOORS=1.0, HP=60.0, MPG=38.0, RPM=5500.0),
Row(BODY=1.0, DOORS=1.0, HP=69.0, MPG=31.0, RPM=5200.0),
Row(BODY=2.0, DOORS=1.0, HP=68.0, MPG=37.0, RPM=5500.0),
Row(BODY=2.0, DOORS=1.0, HP=68.0, MPG=37.0, RPM=5500.0),
Row(BODY=2.0, DOORS=1.0, HP=68.0, MPG=31.0, RPM=5000.0),
Row(BODY=2.0, DOORS=1.0, HP=68.0, MPG=31.0, RPM=5500.0),
Row(BODY=2.0, DOORS=2.0, HP=68.0, MPG=31.0, RPM=5500.0),
Row(BODY=2.0, DOORS=2.0, HP=68.0, MPG=31.0, RPM=5500.0),
Row(BODY=2.0, DOORS=1.0, HP=70.0, MPG=38.0, RPM=5400.0),
Row(BODY=2.0, DOORS=1.0, HP=62.0, MPG=31.0, RPM=4800.0),
Row(BODY=2.0, DOORS=1.0, HP=68.0, MPG=31.0, RPM=5500.0),
Row(BODY=2.0, DOORS=1.0, HP=58.0, MPG=49.0, RPM=4800.0),
Row(BODY=2.0, DOORS=2.0, HP=62.0, MPG=31.0, RPM=4800.0)].
```

In [35]:

Criando o Dataframe

```
carrosDF = spSession.createDataFrame(autoMap)
carrosDF.show()
```

Spark Session

RDD

Cria o Data Frame

BODY	DOORS	HP	MPG	RPM
2.0	1.0	69.0	31.0	4900.0
2.0	1.0	48.0	47.0	5100.0
2.0	1.0	68.0	30.0	5000.0
2.0	1.0	62.0	35.0	4800.0
2.0	1.0	68.0	37.0	5500.0
2.0	1.0	60.0	38.0	5500.0
1.0	1.0	69.0	31.0	5200.0
2.0	1.0	68.0	37.0	5500.0
2.0	1.0	68.0	37.0	5500.0
2.0	1.0	68.0	31.0	5000.0
2.0	1.0	68.0	31.0	5500.0
2.0	2.0	68.0	31.0	5500.0
2.0	1.0	70.0	38.0	5400.0
2.0	1.0	62.0	31.0	4800.0
2.0	1.0	68.0	31.0	5500.0
2.0	1.0	58.0	49.0	4800.0
2.0	2.0	62.0	31.0	4800.0
2.0	1.0	76.0	30.0	6000.0
1.0	2.0	70.0	38.0	5400.0

only showing top 20 rows

In [36]:

Sumarizando as estatísticas do conjunto de dados
 summStats = carrosDF.describe().toPandas()
 summStats

Out[36]:

summary	BODY	DOORS	HP	MPG
0 count	197	197	197	197
1 mean	1.532994923857868	1.5685279187817258	103.60406091370558	25.15228426395939
2 stddev	0.5001812579359883	0.49654352778167493	37.639205349518356	6.437862917085915
3 min	1.0	1.0	48.0	13.0
4 max	2.0	2.0	262.0	49.0

In [37]:

```
# Extrair as médias
medias = summStats.iloc[1,1:5].values.tolist()
medias
```

converte para lista

localização por índice

Out[37]:

```
['1.532994923857868',
 '1.5685279187817258',
 '103.60406091370558',
 '25.15228426395939']
```

In [38]:

```
# Extrair o desvio padrão
desvios_padroes = summStats.iloc[2,1:5].values.tolist()
desvios_padroes
```

Out[38]:

```
['0.5001812579359883',
 '0.49654352778167493',
 '37.639205349518356',
 '6.437862917085915']
```

In [39]:

Inserindo a média e o desvio padrão em uma variável do tipo broadcast

```
bcMedias = sc.broadcast(medias)
bcDesviosP = sc.broadcast(desvios_padroes)
```

Variável Global

In [40]:

```
# Importando a Função Vectors
from pyspark.ml.linalg import Vectors
```

Algebra linear

Função

In [41]:

```
# Função para normalizar os dados e criar um vetor denso
```

```
def centerAndScale(inRow) :
```

```
    global bcMedias
```

```
    global bcDesviosP
```

```
    meanArray = bcMedias.value
    stdArray = bcDesviosP.value
```

Amostra da média

Amostra do desvio padrão

```
    retArray = []
```

→ Cria um array vazio

```
    for i in range(len(meanArray)):
        retArray.append((float(inRow[i]) - float(meanArray[i])) / float(stdArray[i]))
```

```
    return Vectors.dense(retArray)
```

Normalizar os dados

Amostra de média

Amostra de desvio padrão

Normalizar os dados

append adiciona os itens

In [42]:

```
# Aplicando a normalização aos dados
csAuto = carrosDF.rdd.map(centerAndScale)
csAuto.collect()
```

Out[42]:

```
[DenseVector([0.9337, -1.145, -0.9194, 0.9083]),
 DenseVector([0.9337, -1.145, -1.4773, 3.3936]),
 DenseVector([0.9337, -1.145, -0.9459, 0.753]),
 DenseVector([0.9337, -1.145, -1.1053, 1.5297]),
 DenseVector([0.9337, -1.145, -0.9459, 1.8403]),
 DenseVector([0.9337, -1.145, -1.1585, 1.9956]),
 DenseVector([-1.0656, -1.145, -0.9194, 0.9083]),
 DenseVector([0.9337, -1.145, -0.9459, 1.8403]),
 DenseVector([0.9337, -1.145, -0.9459, 1.8403]),
 DenseVector([0.9337, -1.145, -0.9459, 0.9083]),
 DenseVector([0.9337, -1.145, -0.9459, 0.9083]),
 DenseVector([0.9337, 0.869, -0.9459, 0.9083]),
 DenseVector([0.9337, 0.869, -0.9459, 0.9083]),
 DenseVector([0.9337, -1.145, -0.8928, 1.9956]),
 DenseVector([0.9337, -1.145, -1.1053, 0.9083]),
 DenseVector([0.9337, -1.145, -0.9459, 0.9083]),
 DenseVector([0.9337, -1.145, -1.2116, 3.7043]),
 DenseVector([0.9337, 0.869, -1.1053, 0.9083]).
```

In [43]:

```
# Criando um Spark Dataframe com as features (atributos)
autoRows = csAuto.map(lambda f: Row(features = f))
autoDF = spSession.createDataFrame(autoRows)
autoDF.select("features").show(10)
```

features
[0.93367168148051...]
[0.93367168148051...]
[0.93367168148051...]
[0.93367168148051...]
[0.93367168148051...]
[0.93367168148051...]
[-1.0656035495158...]
[0.93367168148051...]
[0.93367168148051...]
[0.93367168148051...]

only showing top 10 rows

cria data frame

Select as 10 primeiros features

Função PI normalizar os dados
converte P/ um RDD para aplicar a transformação

In [44]:

```
# Importando o algoritmo K-Means para clusterização
from pyspark.ml.clustering import KMeans
kmeans = KMeans(k = 3, seed = 1)
modelo = kmeans.fit(autoDF)
previsoes = modelo.transform(autoDF)
previsoes.show()
```

Aprendizagem Não Supervisionada

Para reproduzir o mesmo resultado

divide em 3 grupos

treinamento

features	prediction
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[-1.0656035495158...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[0.93367168148051...	1
[-1.0656035495158...	0

only showing top 20 rows

Previsões

cluster nº 1

cluster nº 0

In [45]:

```
# Plot dos resultados
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

Graph

In [46]:

```
# Função para Leitura dos dados e plotagem
def unstripData(instr) :
    return ( instr["prediction"], instr["features"][0], instr["features"][1], instr["feature
```

Retorno os dados Normalizados
ou Original

In [47]:

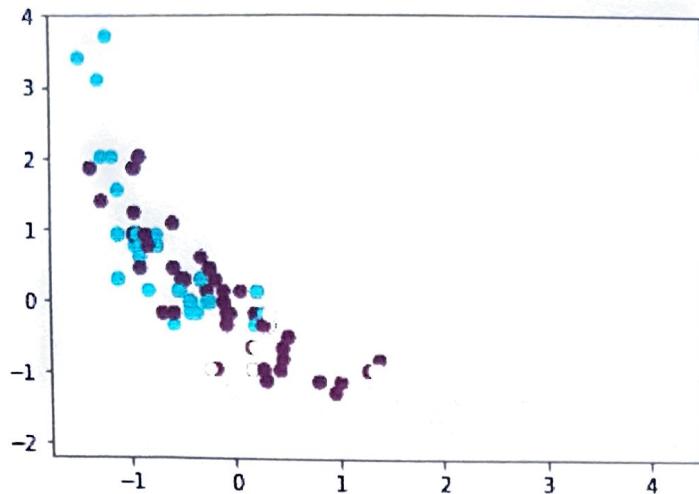
```
# Organizando os dados para o Plot
unstripped = previsoes.rdd.map(unstripData)
predList = unstripped.collect()
predPd = pd.DataFrame(predList)
```

In [48]:

```
plt.cla()
plt.scatter(predPd[3], predPd[4], c = predPd[0])
```

Out[48]:

<matplotlib.collections.PathCollection at 0x1d27f2dd6a0>



Spark SQL e Arquivos JSON

Neste site você pode **validar** a estrutura de um arquivo JSON: <http://jsonlint.com/> (<http://jsonlint.com/>)

In [49]:

```
# Importando o arquivo JSON
funcDF = spSession.read.json("data/funcionarios.json")
```

*Formato de arquivo json
similar ao dicionário
em python*

In [50]:

funcDF.show()

Ja reconhece formato da tabela

deptid	idade	nome	salario	sexo
1000	42	Gilmar Rezende	5000	m
2000	50	Matias Tavares	8500	m
1000	36	Paulo Miranda	9700	m
1000	41	Ana Paula Soares	9500	f
2000	34	Carolina Maia	6500	m

In [51]:

`funcDF.printSchema()`

*esquema
podem se mover*

```
root
|-- deptid: string (nullable = true)
|-- idade: string (nullable = true)
|-- nome: string (nullable = true)
|-- salario: string (nullable = true)
|-- sexo: string (nullable = true)
```

In [52]:

`type(funcDF)`

Out[52]:

`pyspark.sql.dataframe.DataFrame`

In [53]:

Operações com Dataframe Spark SQL - select()
`funcDF.select("nome").show()`

nome
Gilmar Rezende
Matias Tavares
Paulo Miranda
Ana Paula Soares
Carolina Maia

localizar linha onde idade é igual a 50

In [54]:

Operações com Dataframe Spark SQL - filter()
`funcDF.filter(funcDF["idade"] == 50).show()`

deptid	idade	nome	salario	sexo
2000	50	Matias Tavares	8500	m

In [55]:

```
# Operações com Dataframe Spark SQL - groupBy()
funcDF.groupBy("sexo").count().show()
```

sexo	count
m	4
f	1

agregando

In [56]:

```
# Operações com Dataframe Spark SQL - groupBy()
funcDF.groupBy("deptid").agg({"salario": "avg", "idade": "max"}).show()
```

deptid	max(idade)	avg(salario)
2000	50	7500.0
1000	42	8066.666666666667

In [57]:

```
# Registrando o dataframe como uma Temp Table
funcDF.registerTempTable("funcTB")
```

*tabela temporária
conseguida na memória*

In [58]:

```
# Executando queries SQL ANSI linguagem SQL ANSI
spSession.sql("select deptid, max(idade), avg(salario) from funcTB group by deptid").show()
```

deptid	max(idade)	avg(CAST(salario AS DOUBLE))
2000	50	7500.0
1000	42	8066.666666666667

Temp Tables

In [59]:

```
# Registrando o dataframe como temp Table
funcDF.createOrReplaceTempView("funcTB")
```

*Vou verificar se a tabela já
existe nesse session**Se já existir ele recupera, Replace*

*Data Frame
Spark Session
SQL Context*

In [60]:

```
spSession.sql("select * from funcTB where salario = 9700").show()
```

deptid	idade	nome	salario	sexo
1000	36	Paulo Miranda	9700	m

In [61]:

outro alternativa para criar uma tabela temporária

```
# Criando Temp Table
```

```
sqlContext.registerDataFrameAsTable(funcDF, "funcTB2")
```

In [62]:

Pego um data frame e converte em tabela temp.

```
type(funcTB2)
```

```
NameError
```

Traceback (most recent call last)

```
<ipython-input-62-f4105f0fb9ac> in <module>
```

```
----> 1 type(funcTB2)
```

```
NameError: name 'funcTB2' is not defined
```

é apenas uma session no memória, só existe em tempo de execução

In [63]:

```
# Persistindo a Temp Table
```

```
funcTB3 = spSession.table("funcTB2")
```

converte para um objeto definido

In [64]:

```
type(funcTB3)
```

Out[64]:

```
pyspark.sql.dataframe.DataFrame
```

In [65]:

```
# Comparando o Dataframe com a tabela temporária criada
sorted(funcDF.collect()) == sorted(funcTB3.collect())
```

Out[65]:

Verifica se os objetos são iguais
Para depois checar se não houve perda

```
True
```

In [66]:

Aplicando o filtro

```
sqlContext.registerDataFrameAsTable(funcDF, "funcTB2")
funcTB3 = spSession.table("funcTB2")
funcTB3.filter("idade = '42'").first()
```

Out[66]:

Row(deptid='1000', idade='42', nome='Gilmar Rezende', salario='5000', sexo='m')

In [67]:

Drop Temp Table

```
sqlContext.dropTempTable("funcTB2")
```

temos 2 funções filters em RDD

em método do spark SQL

Banco de Dados Relacional

Extraindo Dados do MySQL. Primeiro precisamos baixar o driver JDBC. Haverá um driver JDBC para cada banco de dados que você conectar (Oracle, SQL Server, etc...)

1- Download do Driver JDBC para o MySQL: <http://dev.mysql.com/downloads/connector/j/>
[\(http://dev.mysql.com/downloads/connector/j/\)](http://dev.mysql.com/downloads/connector/j/)

2- Baixar o arquivo .zip

3- Descompactar o arquivo e copiar o arquivo **mysql-connector-java-8.0.16.jar** para a pasta **/opt/Spark/jars** ou para SO Windows em **C:\Spark\jars**

In [71]:

```
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
spSession = SparkSession.builder.master("local").appName("DSA-SparkSQL").getOrCreate()
sqlContext = SQLContext(sc)
```

*Senha ~~1981~~ MySQL
Descartes 1981*

Spark Connector: <https://docs.mongodb.com/spark-connector/current/> (<https://docs.mongodb.com/spark-connector/current/>)

Mongo Spark: <https://spark-packages.org/package/mongodb/mongo-spark> (<https://spark-packages.org/package/mongodb/mongo-spark>)

pyspark --packages org.mongodb.spark:mongo-spark-connector_2.12:2.4.0

In [1]:

```
# Imports
from pyspark.sql import SparkSession
```

*Atenção com o nome do ambiente
startar o mongo no prompt*

In [2]:

```
!java -version
```

```
java version "11.0.3" 2019-04-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.3+12-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.3+12-LTS, mixed mode)
```

In [3]:

```
!python --version
```

```
Python 3.7.3
```

In []:

In []:

Leitura

In [4]:

```
# Cria a sessão
my_spark = SparkSession \
    .builder \
    .appName("myApp") \
    .config("spark.mongodb.input.uri", "mongodb://localhost/test_db.test_collection") \
    .config("spark.mongodb.output.uri", "mongodb://localhost/test_db.test_collection") \
    .getOrCreate()
```

Apenas para que não linka

*Nome do app - pode ser o nome
que quiser*

conectar spark mongo DB

*Cria conexão
ou conect em uma
existente*

*Você pode salvar os dados
resultados no mesmo mongoDB
e servidor ou em outro*

In [5]:

`type(my_spark)`

Out[5]:

`pyspark.sql.session.SparkSession`

In [6]:

`print(my_spark)``<pyspark.sql.session.SparkSession object at 0x00000210155B59B0>`

In []:

In []:

In []:

*Carrega os dados do MongoDB**Para o Spark*

In [7]:

Carrega os dados do MongoDB no Spark

`dados = spark.read.format("com.mongodb.spark.sql.DefaultSource").load()`*Carrega os dados no memória*

In [8]:

`dados.printSchema()`

```

root
 |-- _id: struct (nullable = true)
 |   |-- oid: string (nullable = true)
 |-- item: string (nullable = true)
 |-- qty: double (nullable = true)
 |-- size: struct (nullable = true)
 |   |-- h: double (nullable = true)
 |   |-- w: double (nullable = true)
 |   |-- uom: string (nullable = true)
 |-- tags: array (nullable = true)
 |   |-- element: string (containsNull = true)

```

In [9]:

`dados.count()`*Quantos registros tem no banco*

Out[9]:

3

In [10]:

dados.head()

Out[10]:

```
Row(_id=Row(oid='5d0d21af50e9cbfc4385174c'), item='Camisa Polo', qty=25.0, size=Row(h=14.0, w=21.0, uom='cm'), tags=['branco', 'vermelho'])
```

In [11]:

dados.show()

ags	_id	item	qty	size	t
	[5d0d21af50e9cbfc...]	Camisa Polo	[25.0]	[14.0, 21.0, cm]	[branco, vermelho]
	[5d0d21af50e9cbfc...]	Vestido Bordado	[85.0]	[27.9, 35.5, cm]	[cinza]
	[5d0d21af50e9cbfc...]	Moleton	[45.0]	[19.0, 22.85, cm]	[verde, azul]

A partir daqui só usar as funções SQL bulk

Gravação

In [12]:

```
registro = spark.createDataFrame([(("Camisa T-Shirt", 50)], ["item", "qty"])
```

In [13]:

```
registro.write.format("com.mongodb.spark.sql.DefaultSource").mode("append").save()
```

- o que não estiver fornecendo parte será ignorado
- o id é criado automaticamente

In [14]:

dados.show()

ags	_id	item	qty	size	t
[5d0d21af50e9cbfc...]	Camisa Polo 25.0	[14.0, 21.0, cm]	14.0, 21.0, cm	[branco, vermelho]	
[5d0d21af50e9cbfc...]	Vestido Bordado 85.0	[27.9, 35.5, cm]	27.9, 35.5, cm	[cinza]	
[5d0d21af50e9cbfc...]	Moleton 45.0	[19.0, 22.85, cm]	19.0, 22.85, cm	[verde, azul]	
null	Camisa T-Shirt 50.0	50.0	null	50.0	n

Fim

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

Data Science Academy

Big Data Real-Time Analytics com Python e Spark

Capítulo 8

*** Atenção: ***

Utilize Java JDK 1.8 e Apache Spark 2.4.2

Java JDK 1.8:

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
(<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>)

Caso receba mensagem de erro "name 'sc' is not defined", interrompa o pyspark e apague o diretório metastore_db no mesmo diretório onde está este Jupyter notebook

Acesse <http://localhost:4040> (<http://localhost:4040>) sempre que quiser acompanhar a execução dos jobs

Agrupamento com Pares RDD (Pair RDD)

Tipo especial de RDD que armazena pares chave-valor. Útil quando é necessário armazenar dados que possuem uma chave e diversos valores (por exemplo, todas as transações de um cliente, geradas em tempo real).

`mapValues()`

`countByKey()`

`groupByKey()`

`reduceByKey()`

`aggregateByKey()`

In [1]:

```
# Importando arquivo csv e criando um RDD
carros = sc.textFile("data/carros.csv")
```

In [2]:

carros.take(20)

Out[2]:

```
[ 'MAKE', FUELTYPE, ASPIRE, DOORS, BODY, DRIVE, CYLINDERS, HP, RPM, MPG-CITY, MPG-HWY, PR
ICE',
'subaru,gas,std,two,hatchback,fwd,four,69,4900,31,36,5118',
'chevrolet,gas,std,two,hatchback,fwd,three,48,5100,47,53,5151',
'mazda,gas,std,two,hatchback,fwd,four,68,5000,30,31,5195',
'toyota,gas,std,two,hatchback,fwd,four,62,4800,35,39,5348',
mitsubishi,gas,std,two,hatchback,fwd,four,68,5500,37,41,5389',
'honda,gas,std,two,hatchback,fwd,four,60,5500,38,42,5399',
'nissan,gas,std,two,sedan,fwd,four,69,5200,31,37,5499',
'dodge,gas,std,two,hatchback,fwd,four,68,5500,37,41,5572',
'plymouth,gas,std,two,hatchback,fwd,four,68,5500,37,41,5572',
'mazda,gas,std,two,hatchback,fwd,four,68,5000,31,38,6095',
mitsubishi,gas,std,two,hatchback,fwd,four,68,5500,31,38,6189,
'dodge,gas,std,four,hatchback,fwd,four,68,5500,31,38,6229',
'plymouth,gas,std,four,hatchback,fwd,four,68,5500,31,38,6229',
'chevrolet,gas,std,two,hatchback,fwd,four,70,5400,38,43,6295',
'toyota,gas,std,two,hatchback,fwd,four,62,4800,31,38,6338',
'dodge,gas,std,two,hatchback,fwd,four,68,5500,31,38,6377',
'honda,gas,std,two,hatchback,fwd,four,58,4800,49,54,6479',
'toyota,gas,std,four,hatchback,fwd,four,62,4800,31,38,6488',
'honda,gas,std,two,hatchback,fwd,four,76,6000,30,34,6529 ]
```

In [3]:

```
# Criando uma Pair RDD
carrosPairRDD = carros.map(lambda x: (x.split(",") [0], x.split(",") [7]))
carrosPairRDD.take(20)
```

Out[3]:

```
[('MAKE', 'HP'),
('subaru', '69'),
('chevrolet', '48'),
('mazda', '68'),
('toyota', '62'),
('mitsubishi', '68'),
('honda', '60'),
('nissan', '69'),
('dodge', '68'),
('plymouth', '68'),
('mazda', '68'),
('mitsubishi', '68'),
('dodge', '68'),
('plymouth', '68'),
('chevrolet', '70'),
('toyota', '62'),
('dodge', '68'),
('honda', '58'),
('toyota', '62'),
('honda', '76')]
```

column 0

column 7

transformação = não é executado ainda

Ação = Executar

In [4]:

```
# Removendo o cabeçalho
header = carrosPairRDD.first()
carrosPairRDD2 = carrosPairRDD.filter(lambda line: line != header)
```

In [5]:

```
# Encontra o valor de HP por marca de carro e adiciona 1 a cada registro "Make/HP"
carrosPairRDD3 = carrosPairRDD2.mapValues(lambda x: (x, 1))
carrosPairRDD3.collect()
```

Out[5]:

```
[('subaru', ('69', 1)),
('chevrolet', ('48', 1)),
('mazda', ('68', 1)),
('toyota', ('62', 1)),
('mitsubishi', ('68', 1)),
('honda', ('60', 1)),
('nissan', ('69', 1)),
('dodge', ('68', 1)),
('plymouth', ('68', 1)),
('mazda', ('68', 1)),
('mitsubishi', ('68', 1)),
('dodge', ('68', 1)),
('plymouth', ('68', 1)),
('chevrolet', ('70', 1)),
('toyota', ('62', 1)),
('dodge', ('68', 1)),
('honda', ('58', 1)),
('toyota', ('62', 1)).
```

uma tupla
com os dados por HP

In [6]:

```
# Aplica redução por key (reduceByKey)
# E calcula o total de HP por fabricante e o total de automóveis por fabricante
fabricantes = carrosPairRDD3.reduceByKey(lambda x, y: (int(x[0]) + int(y[0])), x[1] + y[1])
fabricantes.collect()
```

Out[6]:

```
[('chevrolet', (188, 3)),
 ('mazda', (1390, 16)),
 ('mitsubishi', (1353, 13)),
 ('nissan', (1846, 18)),
 ('dodge', (675, 8)),
 ('plymouth', (607, 7)),
 ('saab', (760, 6)),
 ('volvo', (1408, 11)),
 ('alfa-romero', (376, 3)),
 ('mercedes-benz', (1170, 8)),
 ('jaguar', (614, 3)),
 ('subaru', (1035, 12)),
 ('toyota', (2969, 32)),
 ('honda', (1043, 13)),
 ('isuzu', (168, 2)),
 ('volkswagen', (973, 12)),
 ('peugot', (1098, 11)),
 ('audi', (687, 6)),
 ('bmw', (1111, 8)),
 ('mercury', ('175', 1)),
 ('porsche', (764, 4))]
```

3 automóveis do chevrolet
com HP (cavalo de potêncio) somados
exibe informações

In [7]:

```
# Calculando a média de HP dividindo pela contagem total
fabricantes.mapValues(lambda x: int(x[0])/int(x[1])).collect()
```

Out[7]:

```
[('chevrolet', 62.66666666666664),
 ('mazda', 86.875),
 ('mitsubishi', 104.07692307692308),
 ('nissan', 102.55555555555556),
 ('dodge', 84.375),
 ('plymouth', 86.71428571428571),
 ('saab', 126.6666666666667),
 ('volvo', 128.0),
 ('alfa-romero', 125.3333333333333),
 ('mercedes-benz', 146.25),
 ('jaguar', 204.6666666666666),
 ('subaru', 86.25),
 ('toyota', 92.78125),
 ('honda', 80.23076923076923),
 ('isuzu', 84.0),
 ('volkswagen', 81.0833333333333),
 ('peugot', 99.818181818181),
 ('audi', 114.5),
 ('bmw', 138.875),
 ('mercury', 175.0),
 ('porsche', 191.0)]
```

média de HP

Variável só pode ser vista
Pelo Node, não pode ser
vista por outro node

Accumuladores e Broadcast

O Spark faz uma cópia do código que você escreveu para processar os dados e executa essas cópias, uma por node do cluster. Qualquer variável criada no código é local ao node. O Spark gera cópias dessas variáveis locais, uma em cada node, que agem de forma independente. Mas se precisamos que a mesma variável seja manipulada de forma única através de todo o cluster? Usamos Acumuladores e Broadcast.

Pode ser alterada

Variável Broadcast - read-only, é compartilhada em todo o cluster.

não pode alterar
Só pode ser lida

Variável Accumulator - é compartilhada em todo o cluster, mas pode ser atualizada em cada node do cluster.

In [8]:

```
# Inicializando variáveis Accumulator
sedanCount = sc.accumulator(0)
hatchbackCount = sc.accumulator(0)
```

Se você quiser uma única Variável
Sendo compartilhada por todos os
Nodes

In [9]:

```
# Inicializando variáveis Broadcast
sedanText = sc.broadcast("sedan")
hatchbackText = sc.broadcast("hatchback")
```

In [10]:

```
def splitLines(line):
    global sedanCount
    global hatchbackCount

    # Usa a variável Broadcast para comparar e adiciona a contagem ao accumulator
    if sedanText.value in line:
        sedanCount +=1
    if hatchbackText.value in line:
        hatchbackCount +=1

    return line.split(",")
```

Pode ser visto por todo o programa

Usa a variável Broadcast para comparar e adiciona a contagem ao accumulator

accumulator, está edicionando, somando

In [11]:

```
# Map()
splitData = carros.map(splitLines)
```

RDD

Função

In [12]:

```
# Ação para executar a transformação (Lazy evaluation)
splitData.count()
print(sedanCount, hatchbackCount)
```

executa a transformação
apenas com o .count()

92 67

Partições

Sempre que criamos RDD's, esses objetos são divididos em partições e essas partições são distribuídas através dos nodes do cluster. Por default, os RDD's são sempre particionados. Essas partições precisam ser configuradas quando se trabalha com grandes clusters.

In [13]:

```
fabricantes.getNumPartitions()
```

Out[13]:

2

In [14]:

```
# Especificando o Número de Partições
collData = sc.parallelize([3,5,4,7,4], 3)
collData.cache() → memoria
collData.count()
```

5 elementos

3 partições

Out[14]:

5

In [15]:

```
collData.getNumPartitions()
```

Out[15]:

3

In [16]:

```
print (sc.defaultParallelism)
```

8

Por poderás ciar 8 part. cada

Fim

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