Ejercicios Sab24

January 31, 2018

```
In [1]: import numpy as np
```

$$y = [0, 1, 2, 3, 4]$$

```
In [2]: mi_arreglo = [0, 1, 2, 3, 4]

In [3]: type(mi_arreglo)

Out[3]: list

In [4]: mi_arreglo_numpy = np.array(mi_arreglo)

In [5]: type(mi_arreglo_numpy)

Out[5]: numpy.ndarray

In [6]: help(np.array)

Help on built-in function array in module numpy.core.multiarray:

array(...)
    array(object, dtype=None, copy=True, order=None, subok=False, ndmin=0)

    Create an array.

    Parameters
    ----------
    object : array_like
        An array, any object exposing the array interface, an
        object whose __array__ method returns an array, or any
        (nested) sequence.
    dtype : data-type, optional
        The desired data-type for the array.  If not given, then
        the type will be determined as the minimum type required
        to hold the objects in the sequence.  This argument can only
        be used to 'upcast' the array.  For downcasting, use the
        .astype(t) method.
```

```
copy : bool, optional
    If true (default), then the object is copied.  Otherwise, a copy
    will only be made if __array__ returns a copy, if obj is a
    nested sequence, or if a copy is needed to satisfy any of the other
    requirements (`dtype`, `order`, etc.).
order : {'C', 'F', 'A'}, optional
    Specify the order of the array.  If order is 'C', then the array
    will be in C-contiguous order (last-index varies the fastest).
    If order is 'F', then the returned array will be in
    Fortran-contiguous order (first-index varies the fastest).
    If order is 'A' (default), then the returned array may be
    in any order (either C-, Fortran-contiguous, or even discontiguous),
    unless a copy is required, in which case it will be C-contiguous.
subok : bool, optional
    If True, then sub-classes will be passed-through, otherwise
    the returned array will be forced to be a base-class array (default).
ndmin : int, optional
    Specifies the minimum number of dimensions that the resulting
    array should have.  Ones will be pre-pended to the shape as
    needed to meet this requirement.

Returns
-------
out : ndarray
    An array object satisfying the specified requirements.

See Also
--------
empty, empty_like, zeros, zeros_like, ones, ones_like, fill

Examples
--------
>>> np.array([1, 2, 3])
array([1, 2, 3])

Upcasting:

>>> np.array([1, 2, 3.0])
array([ 1.,  2.,  3.])

More than one dimension:

>>> np.array([[1, 2], [3, 4]])
array([[1, 2],
       [3, 4]])

Minimum dimensions 2:
```

```
    >>> np.array([1, 2, 3], ndmin=2)
    array([[1, 2, 3]])

    Type provided:

    >>> np.array([1, 2, 3], dtype=complex)
    array([ 1.+0.j,  2.+0.j,  3.+0.j])

    Data-type consisting of more than one element:

    >>> x = np.array([(1,2),(3,4)],dtype=[('a','<i4'),('b','<i4')])
    >>> x['a']
    array([1, 3])

    Creating an array from sub-classes:

    >>> np.array(np.mat('1 2; 3 4'))
    array([[1, 2],
           [3, 4]])

    >>> np.array(np.mat('1 2; 3 4'), subok=True)
    matrix([[1, 2],
            [3, 4]])
```

```
In [7]: mi_arreglo_numpy * mi_arreglo_numpy

Out[7]: array([ 0,  1,  4,  9, 16])

In [9]: np.ndim(mi_arreglo_numpy)

Out[9]: 1

In [10]: mi_arreglo_numpy.size

Out[10]: 5
```

0.0.1 Matrices (Arreglos de dos dimensiones)

$$x = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

```
In [11]: mi_matriz_numpy = np.array([[1,2,3],
                                     [4,5,6],
                                     [7,8,9]])

In [15]: print(mi_matriz_numpy.ndim, mi_matriz_numpy.size)
```

```
2 9
```

Selección escalar

```
In [13]: print(mi_matriz_numpy)

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


In [16]: mi_matriz_numpy[1,2]

Out[16]: 6

In [21]: mi_matriz_numpy[1:2]

Out[21]: array([[4, 5, 6]])
```

- a[comienzo:fin] # elementos desde el índice del comienzo hasta el índice fin-1
- a[comienzo:] # del número en comienzo hasta el fin
- a[:fin] # desde el principio hasta fin-1
- a[:] # todo el arreglo
- a[comienzo:fin:paso] # elementos desde el índice del comienzo hasta el índice fin-1, por paso

```
In [22]: mi_matriz_numpy

Out[22]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [23]: mi_matriz_numpy * 2

Out[23]: array([[ 2,  4,  6],
                [ 8, 10, 12],
                [14, 16, 18]])

In [24]: mi_matriz_numpy + 100

Out[24]: array([[101, 102, 103],
                [104, 105, 106],
                [107, 108, 109]])

In [27]: mi_matriz_numpy + mi_matriz_numpy

Out[27]: array([[ 2,  4,  6],
                [ 8, 10, 12],
                [14, 16, 18]])

In [35]: mi_matriz_numpy
```

```
Out[35]: array([[1, 2, 3],
                [4, 5, 6],
                [7, 8, 9]])

In [37]: mi_matriz_numpy.flatten()

Out[37]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

In [68]: %timeit  mi_matriz_numpy.flatten().ndim

The slowest run took 103.69 times longer than the fastest. This could mean that an intermediat
1000000 loops, best of 3: 982 ns per loop


In [71]: %timeit np.ndim(mi_matriz_numpy.flatten())

The slowest run took 19.89 times longer than the fastest. This could mean that an intermediate
100000 loops, best of 3: 1.25 µs per loop


In [72]: mi_matriz_numpy.mean()

Out[72]: 5.0

In [73]: mi_matriz_numpy.std()

Out[73]: 2.5819888974716112

In [74]: import pandas as pd

In [82]: mi_serie = pd.Series([1, 2, 3, 4, 5],
                              index = ['a', 'b', 'c','d', 'e'])


         mi_serie

Out[82]: a    1
         b    2
         c    3
         d    4
         e    5
         dtype: int64

In [83]: mi_serie[['a', 'd']]

Out[83]: a    1
         d    4
         dtype: int64

In [87]: mi_serie[mi_serie>2]
```

```
Out[87]: c    3
         d    4
         e    5
         dtype: int64

In [106]: mi_df = pd.DataFrame({
              'municipio': ['mty', 'spgg', 'sta_c', 'garcia'],
              'temp_celcius_jueves': [35, 36, 37, 38],
              'temp_celcius_viernes': [36, 37, 39, 38]
          })

          mi_df['temp_celcius_jueves']

Out[106]: 0    35
          1    36
          2    37
          3    38
          Name: temp_celcius_jueves, dtype: int64

In [124]: mi_df[mi_df['temp_celcius_jueves'] < 38]['temp_celcius_jueves']

Out[124]: 0    35
          1    36
          2    37
          Name: temp_celcius_jueves, dtype: int64

In [123]: mi_df[[mi_df['temp_celcius_jueves'] < 38] and
               mi_df['temp_celcius_viernes'] < 38]

Out[123]:   municipio  temp_celcius_jueves  temp_celcius_viernes
          0       mty                   35                    36
          1      spgg                   36                    37

In [149]: miarreglo2 = pd.DataFrame({
              'municipio': ['mpio8', 'municipio 53'],
              'temp_celcius_jueves': [35, 45],
              'temp_celcius_viernes': [36, 42]
          })

          miarreglo2

Out[149]:       municipio  temp_celcius_jueves  temp_celcius_viernes
          0         mpio8                   35                    36
          1  municipio 53                   45                    42

In [150]: mi_df = mi_df.append(miarreglo2, ignore_index=True)
          mi_df
```

Out[150]:

	municipio	temp_celcius_jueves	temp_celcius_viernes
0	mty	35	36
1	spgg	36	37
2	sta_c	37	39
3	garcia	38	38
4	laredo	35	36
5	nvo_laredo	35	36
6	china	45	42
7	mpio4	35	36
8	municipio 5	45	42
9	mpio8	35	36
10	municipio 53	45	42

1 http://bit.ly/2sCuFc5

In [152]: url_dataset = 'https://gist.githubusercontent.com/israelzuniga/de79faf13d5fe48ea4f65

1.0.1 Datos

Los datos son características cualitativas o cuantitativas pertenecientes a un objeto, o un conjunto de objetos

1.0.2 Datos en bruto

"Raw data is a term for data collected on source which has not been subjected to processing or any other manipulation."

- Los datos en bruto llegan directamente de la fuente y no tienen la estructura necesaria para realizar análisis con ellos eficientemente.
- Requieren pre-procesamiento para ser utilizados.
- Por lo general suelen verse de la siguiente manera:

Video, audio, páginas web, también son fuentes de datos

1.0.3 Datos ordenados (Tidy data)

Notebook Tidy_data

- Las variables deben de ser entendibles para el humano

Codebook! Documento para poder entender la información de la tabla.

- Descripción de las características con sus unidades
- Instrucciones sobre las transformaciones que aplicamos a nuestros datos en bruto para trabajarlos

ESTO ES MUY IMPORTANTE
Existen historias de terror:
http://www.cc.com/video-clips/dcyvro/the-colbert-report-austerity-s-spreadsheet-error

1.1 Dataset para hoy - train.csv

1.1.1 Predecir la supervivencia en el Titanic

https://www.kaggle.com/c/titanic/data

El hundimiento del Titanic es uno de los naufragios más infames de la historia. El 15 de abril de 1912, durante su viaje inaugural, el Titanic se hundió después de chocar con un iceberg, matando de 1,502 a 2,224 pasajeros.

Una de las razones por las cuales se perdieron tantas vidas fue que no había suficientes botes salvavidas. Aunque hubo algún elemento de suerte involucrada en sobrevivir al hundimiento, algunos grupos de personas tenían más probabilidades de sobrevivir que otros, como las mujeres, los niños y personas de la clase alta.

Param hoy usaremos el dataset `train.csv` y completarás el análisis de qué tipo de personas eran propensos a sobrevivir . En un futuro, te pedimos aplicar las herramientas de aprendizaje automático para predecir que los pasajeros sobrevivieron a la tragedia.

1.1.2 Matplotlib

Las visualizaciones son una de las herramientas más poderosas a su disposición para explorar los datos y comunicar tus ideas. La biblioteca pandas incluye capacidades básicas para graficar con el paquete matplotlib.

```
In [226]: titanic = pd.read_csv(url_dataset)

In [227]: titanic.Age = titanic.Age.fillna(29)

In [228]: titanic.describe()
```

Out[228]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.560236	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	13.005010	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	22.000000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	29.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	35.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

```
In [160]: titanic.tail(12)
```

Out[160]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
879	880	1	1	Potter, Mrs. Thomas Jr (Lily Alexenia Wilson)	female	56.0	0	1	11767	83.1583	C50	C
880	881	1	2	Shelley, Mrs. William (Imanita Parrish Hall)	female	25.0	0	1	230433	26.0000	NaN	S
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958	NaN	S
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167	NaN	S
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000	NaN	S
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500	NaN	S
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250	NaN	Q
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000	NaN	S
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000	B42	S
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500	NaN	S
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000	C148	C
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500	NaN	Q

In [162]:
```
titanic.describe()
```

Out[162]:

	PassengerId	Survived	Pclass	Age	SibSp
count	891.000000	891.000000	891.000000	714.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008
std	257.353842	0.486592	0.836071	14.526497	1.102743

```
              min       1.000000    0.000000    1.000000    0.420000    0.000000
              25%     223.500000    0.000000    2.000000   20.125000    0.000000
              50%     446.000000    0.000000    3.000000   28.000000    0.000000
              75%     668.500000    1.000000    3.000000   38.000000    1.000000
              max     891.000000    1.000000    3.000000   80.000000    8.000000

                           Parch        Fare
              count  891.000000  891.000000
              mean     0.381594   32.204208
              std      0.806057   49.693429
              min      0.000000    0.000000
              25%      0.000000    7.910400
              50%      0.000000   14.454200
              75%      0.000000   31.000000
              max      6.000000  512.329200
```

```
In [163]: titanic.shape
```

```
Out[163]: (891, 12)
```

```
In [164]: len(titanic)
```

```
Out[164]: 891
```

```
In [165]: len(titanic.keys())
```

```
Out[165]: 12
```

```
In [168]: edad = titanic['Age']
```

```
In [171]: sum(edad.isnull())
```

```
Out[171]: 177
```

```
In [175]: np.mean(edad)
```

```
Out[175]: 29.69911764705882
```

```
In [176]: edad = titanic['Age'].fillna(29)
```

```
In [177]: sum(edad.isnull())
```

```
Out[177]: 0
```

```
In [178]: np.mean([1,2,3])
```

```
Out[178]: 2.0
```

```
In [183]: titanic.Pclass.unique()
```

```
Out[183]: array([3, 1, 2])
```

```
In [186]: titanic['Pclass'].unique()

Out[186]: array([3, 1, 2])

In [190]: titanic[['Pclass', 'Age']].head()

Out[190]:    Pclass   Age
          0       3  22.0
          1       1  38.0
          2       3  26.0
          3       1  35.0
          4       3  35.0
```

Podemos borrar columnas usando los comando del, pop(), drop() - del modifica nuestro dataframe borrando la Serie seleccionada. - pop() borra la Serie pero la regresa como un output - drop() regresa un dataframe sin la Serie, pero no modifica el dataframe original

```
In [192]: titanic.keys()

Out[192]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
                 'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
                dtype='object')

In [225]: titanic.head(1)

Out[225]:    PassengerId  Survived  Pclass                     Name  SibSp  Fare  \
          0            1         0       3  Braund, Mr. Owen Harris      1  7.25

            Embarked
          0        S

In [205]: # - pop() borra la Serie pero la regresa como un output

          cabin = titanic.pop('Cabin')

In [224]: # drop() regresa un dataframe sin la Serie, pero no modifica el
          # dataframe original

          titanic = titanic.drop('Sex', axis=1)

In [206]:

Out[206]: 0              NaN
          1              C85
          2              NaN
          3             C123
          4              NaN
          5              NaN
          6              E46
          7              NaN
```

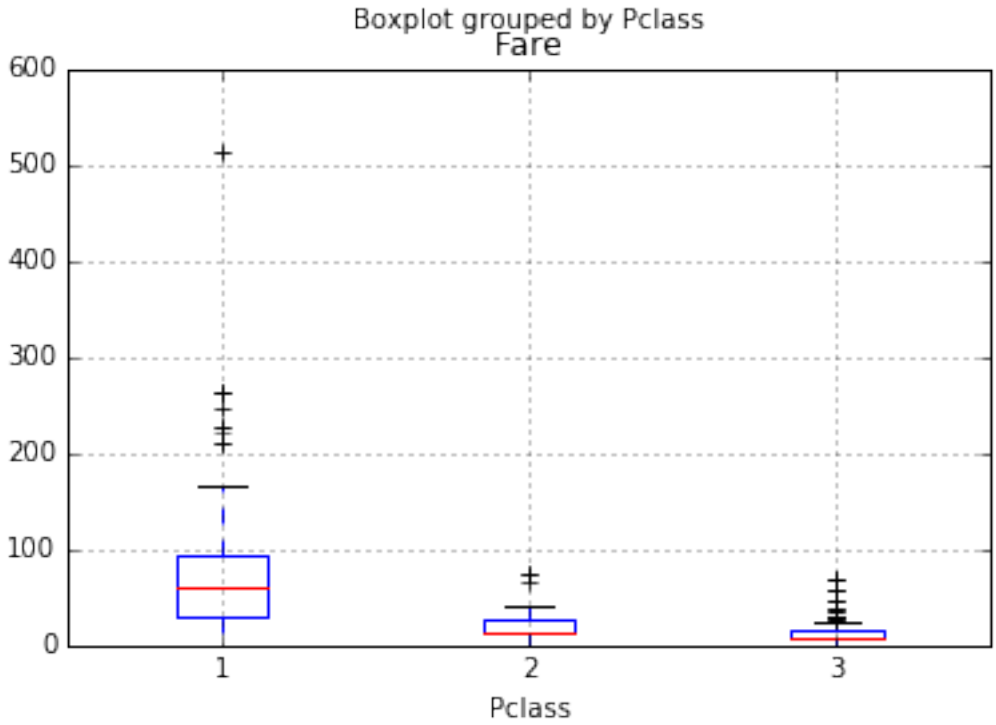
```
8              NaN
9              NaN
10              G6
11            C103
12             NaN
13             NaN
14             NaN
15             NaN
16             NaN
17             NaN
18             NaN
19             NaN
20             NaN
21             D56
22             NaN
23              A6
24             NaN
25             NaN
26             NaN
27     C23 C25 C27
28             NaN
29             NaN
          ...
861            NaN
862            D17
863            NaN
864            NaN
865            NaN
866            NaN
867            A24
868            NaN
869            NaN
870            NaN
871            D35
872    B51 B53 B55
873            NaN
874            NaN
875            NaN
876            NaN
877            NaN
878            NaN
879            C50
880            NaN
881            NaN
882            NaN
883            NaN
884            NaN
885            NaN
```

```
886            NaN
887            B42
888            NaN
889           C148
890            NaN
Name: Cabin, dtype: object
```

```
In [241]: import matplotlib as plt
          %matplotlib inline
```

```
In [246]: titanic.boxplot(column='Fare', by='Pclass', return_type='axes')
```

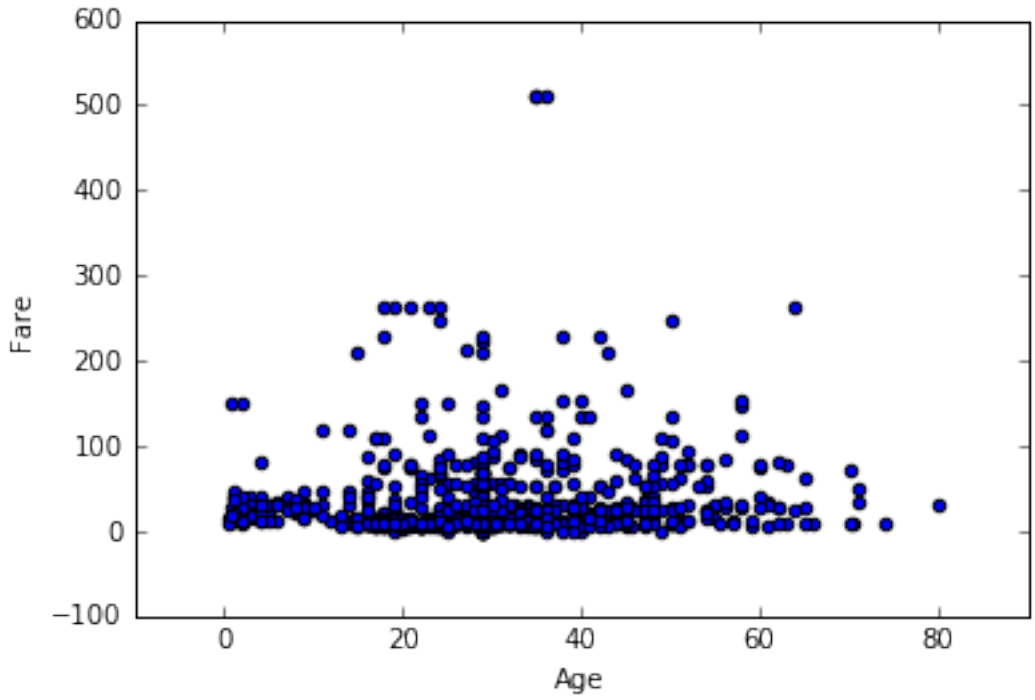
```
Out[246]: OrderedDict([('Fare', <matplotlib.axes._subplots.AxesSubplot at 0x113162dd8>)])
```




```
In [247]: #scatterplot
```

```
          titanic.plot(kind='scatter',
                       x = 'Age',
                       y = 'Fare')
```

```
Out[247]: <matplotlib.axes._subplots.AxesSubplot at 0x112fa0668>
```




```
In [270]: mi_tabla_cruzada = pd.crosstab(index=titanic.Pclass,
                                          columns=titanic.Survived,
                                          margins=True)
          mi_tabla_cruzada
```

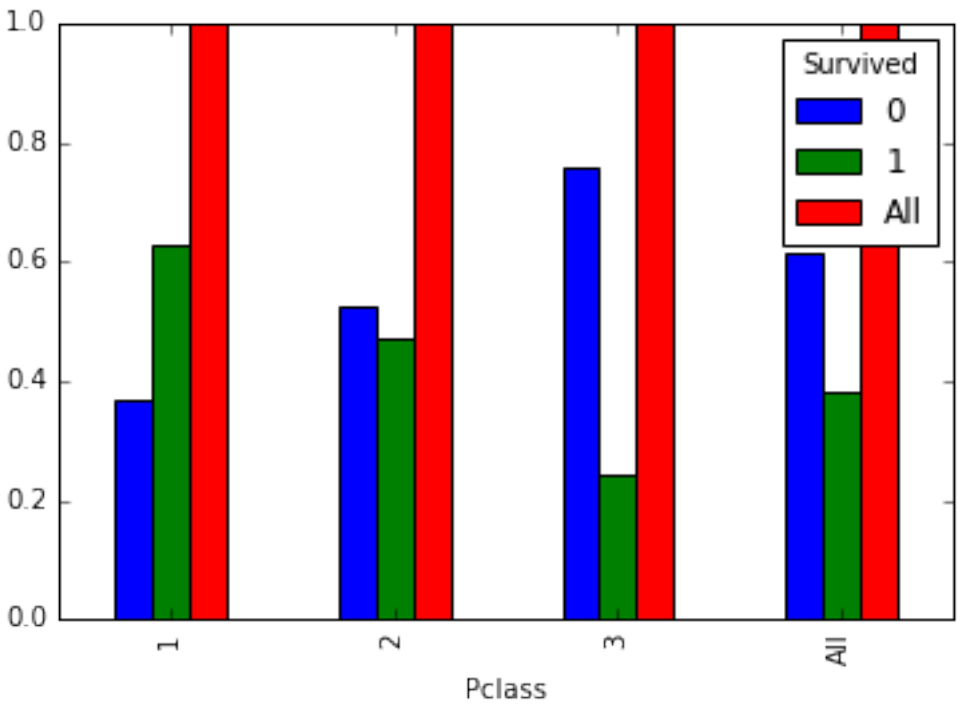
```
Out[270]: Survived    0    1  All
          Pclass
          1          80  136  216
          2          97   87  184
          3         372  119  491
          All       549  342  891
```

```
In [261]: mi_tabla_cruzada  = mi_tabla_cruzada.div(mi_tabla_cruzada['All'],
                                                   axis=0)
```

```
Out[261]: Survived         0         1  All
          Pclass
          1         0.370370  0.629630  1.0
          2         0.527174  0.472826  1.0
          3         0.757637  0.242363  1.0
          All       0.616162  0.383838  1.0
```

```
In [262]: mi_tabla_cruzada.plot(kind='bar')
```

```
Out[262]: <matplotlib.axes._subplots.AxesSubplot at 0x1136cb278>
```



In [256]:

Out[256]: Int64Index([0, 1], dtype='int64', name='Survived')