**Lab - Data Visualization de datos con matplotlib**

Se presentan los componentes y el estilo de los gráficos de barras, gráficos de líneas, gráficos de dispersión y gráficos circulares. Se aprendera la esencia de diseccionar matplotlib y usarlo a su voluntad. Una vez que haya aprendido cómo hacer todo esto, otras bibliotecas de visualización de Python son bastante fáciles de aprender.

Existen otras bibliotecas de visualización agradables, como:

* seaborn: <https://web.stanford.edu/~mwaskom/software/seaborn/>
* bokeh: <http://bokeh.pydata.org/en/latest/>

**In [1]:**

*# magic command to display matplotlib plots inline within the ipython notebook webpage*

%**matplotlib** inline

*# import necessary modules*

**import** **pandas** **as** **pd**, **numpy** **as** **np**, **matplotlib.pyplot** **as** **plt**

**import** **matplotlib.cm** **as** **cm**, **matplotlib.font\_manager** **as** **fm**

**load the data sets**

**In [3]:**

*# load the gps coordinate data, using the date as the full set's index*

*# the data files are encoded as utf-8: specify so to prevent matplotlib from choking on diacritics*

df = pd.read\_csv('summer-travel-gps-full.csv', encoding='utf-8', index\_col='date', parse\_dates=**True**)

rs = pd.read\_csv('summer-travel-gps-dbscan.csv', encoding='utf-8')

In [4]:

df.head()

**Out[4]:**

|  | **lat** | **lon** | **city** | **country** |
| --- | --- | --- | --- | --- |
| **date** |  |  |  |  |
| **2014-05-14 09:07:00** | 51.481292 | -0.451011 | West Drayton | United Kingdom |
| **2014-05-14 09:22:00** | 51.474005 | -0.450999 | Hounslow | United Kingdom |
| **2014-05-14 10:51:00** | 51.478199 | -0.446081 | Hounslow | United Kingdom |
| **2014-05-14 11:24:00** | 51.478199 | -0.446081 | Hounslow | United Kingdom |
| **2014-05-14 11:38:00** | 51.474146 | -0.451562 | Hounslow | United Kingdom |

In [5]:

rs.head()

Out[5]:

|  | **lat** | **lon** | **date** | **city** | **country** |
| --- | --- | --- | --- | --- | --- |
| **0** | 51.478199 | -0.446081 | 05/14/2014 10:51 | Hounslow | United Kingdom |
| **1** | 38.781775 | -9.137544 | 05/14/2014 15:11 | Lisbon | Portugal |
| **2** | 38.771939 | -9.128911 | 05/14/2014 15:41 | Lisbon | Portugal |
| **3** | 38.742987 | -9.147780 | 05/14/2014 16:11 | Lisbon | Portugal |
| **4** | 38.693780 | -9.206363 | 05/15/2014 14:19 | Lisbon | Portugal |

In [6]:

*# define the font styles*

title\_font = fm.FontProperties(family='Bitstream Vera Sans', style='normal', size=15, weight='normal', stretch='normal')

label\_font = fm.FontProperties(family='Bitstream Vera Sans', style='normal', size=12, weight='normal', stretch='normal')

ticks\_font = fm.FontProperties(family='Bitstream Vera Sans', style='normal', size=10, weight='normal', stretch='normal')

annotation\_font = fm.FontProperties(family='Bitstream Vera Sans', style='normal', size=10, weight='normal', stretch='normal')

axis\_bgcolor = '#f0f0f0'

**Nota:**

Documentación de la clase FontProperties

*class matplotlib.font\_manager.FontProperties(family=None, style=None, variant=None, weight=None, stretch=None, size=None, fname=None, math\_fontfamily=None)[source]*

*Bases: object*

Una clase para almacenar y manipular las propiedades de las fuentes.

Las propiedades de las fuentes son las seis propiedades descritas en la W3C CSS, especificación de fuentes de nivel 1 y *math\_fontfamily* para las fuentes matemáticas:

1. **family**: Una lista de nombres de fuentes en orden decreciente de prioridad. Los elementos pueden incluir un nombre de familia de fuentes genérico, ya sea 'sans-serif', 'serif', 'cursive', 'fantasy' o 'monospace'. En ese caso, el tipo de letra real que se utilizará se buscará a partir del rcParam asociado durante el proceso de búsqueda en findfont. Por defecto: rcParams["font.family"] (por defecto: ['sans-serif'])
2. **style**: O bien 'normal', 'italic' or 'oblique'. Default: rcParams["font.style"] (default: 'normal')
3. **variant**: O bien 'normal' or 'small-caps'. Default: rcParams["font.variant"] (default: 'normal')
4. **stretch**: Un valor numérico en el rango 0-1000 o uno de 'ultra-condensed', 'extra-condensed', 'condensed', 'semi-condensed', 'normal', 'semi-expanded', 'expanded', 'extra-expanded' o 'ultra-expanded'. Default: rcParams["font.stretch"] (default: 'normal')
5. **weight**: Un valor numérico en el rango 0-1000 o uno de 'ultralight', 'light', 'normal', 'regular', 'book', 'medium', 'roman', 'semibold', 'demibold', 'demi', 'bold', 'heavy', 'extra bold', 'black'. Default: rcParams["font.weight"] (default: 'normal')
6. **size**: Un valor relativo de 'xx-small', 'x-small', 'small', 'medium', 'large', 'x-large', 'xx-large' o un tamaño de fuente absoluto, por ejemplo, 10. Por defecto: rcParams["font.size"] (por defecto: 10.0)
7. **math\_fontfamily**: La familia de fuentes utilizada para renderizar el texto matemático. Los valores soportados son: 'dejavusans', 'dejavuserif', 'cm', 'stix', 'stixsans' and 'custom'. Default: rcParams["mathtext.fontset"] (default: 'dejavusans')

**First up: Bar Charts**

In [7]:

*# get the most visited countries*

countries = df['country'].value\_counts()

countries

**Out[7]:**

Spain 646

Portugal 256

Germany 225

Czech Republic 130

Poland 104

Turkey 94

Greece 82

Montenegro 51

Macedonia (FYROM) 34

Kosovo 30

Croatia 30

Albania 29

Bosnia and Herzegovina 26

Serbia 15

United Kingdom 7

Name: country, dtype: int64

**Colores en Matplotlib**

Los colores en matplotlib son los siguientes:



**Atributo Alpha**

Matplotlib permite regular la transparencia de un gráfico con el atributo alpha. Por defecto, alpha=1.

Si se desea formar el gráfico más transparente, entonces alpha será menor que 1, como 0,5 o 0,25.

Si se desea que el gráfico sea menos transparente, entonces debes hacer que alfa sea mayor que 1. Esto solidifica el gráfico, haciéndolo menos transparente y más grueso y denso, por así decirlo.

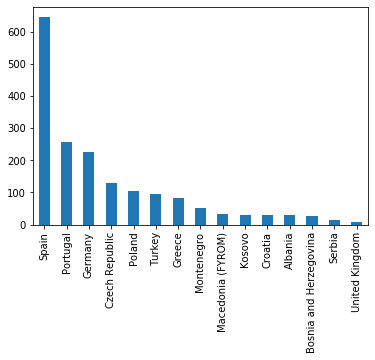
In [8]:

*# default matplotlib bar chart*

countries.plot(kind='bar')

**Out[8]:**

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fb9c76c9160>



In [10]:

# style the chart to make it look nicer

ax = countries.plot(kind='bar', figsize=[9, 6], width=0.6, alpha=0.6,

color='g', edgecolor='k', grid=False, ylim=[0, 700])

ax.set\_xticklabels(countries.index, rotation=45, rotation\_mode='anchor', ha='right', fontproperties=ticks\_font)

ax.yaxis.grid(True)

for label in ax.get\_yticklabels():

label.set\_fontproperties(ticks\_font)

ax.set\_facecolor(axis\_bgcolor)

ax.set\_title('Most Visited Countries', fontproperties=title\_font)

ax.set\_xlabel('')

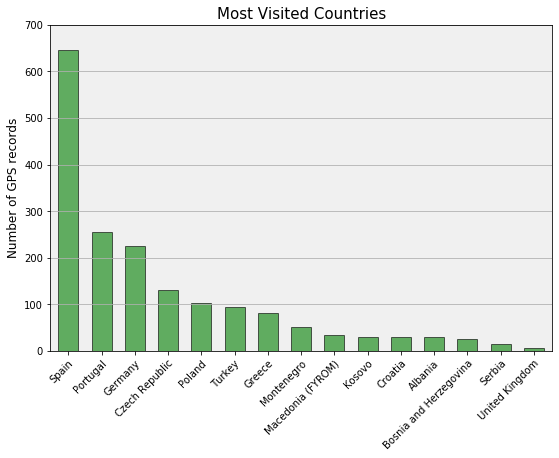
ax.set\_ylabel('Number of GPS records', fontproperties=label\_font)

plt.show()

**out**

findfont: Font family ['Bitstream Vera Sans'] not found. Falling back to DejaVu Sans.

findfont: Font family ['Bitstream Vera Sans'] not found. Falling back to DejaVu Sans.



Ahora intente: vuelva a crear el gráfico anterior, pero haga que las barras sean naranjas con bordes granate.

In [13]:

*# make a re-usable function to draw nice bar charts*

**def** bar\_chart(data, title='', xlabel='', ylabel='', color='k', ylim=**None**):

ax = data.plot(kind='bar', figsize=[9, 6], width=0.6, alpha=0.6,

color=color, edgecolor='k', grid=**False**, ylim=ylim)

ax.set\_xticklabels(data.index, rotation=45, rotation\_mode='anchor', ha='right', fontproperties=ticks\_font)

ax.yaxis.grid(**True**)

**for** label **in** ax.get\_yticklabels():

label.set\_fontproperties(ticks\_font)

ax.set\_facecolor(axis\_bgcolor)

ax.set\_title(title, fontproperties=title\_font)

ax.set\_xlabel(xlabel, fontproperties=label\_font)

ax.set\_ylabel(ylabel, fontproperties=label\_font)

plt.show()

In [14]:

*# let's re-scale the data and use our new function*

log\_countries = np.log(df['country'].value\_counts())

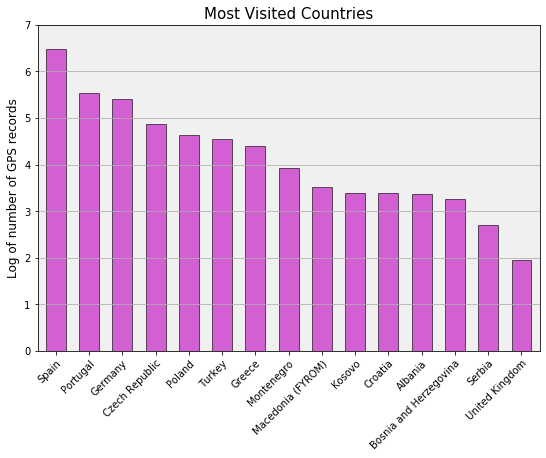
bar\_chart(data=log\_countries,

title='Most Visited Countries',

ylabel='Log of number of GPS records',

color='m',

ylim=[0,7])



In [15]:

*# do the same thing, but now for the most visited cities instead of countries*

cities = df['city'].value\_counts().head(13)

cities

Out[15]:

Barcelona 632

Lisbon 158

Tübingen 131

Prague 97

Istanbul 94

Athens 67

Krakow 58

Porto 58

Freiburg 54

Kotor 51

Ohrid 34

Prizren 30

Berat 27

Name: city, dtype: int64

**In [16]:**

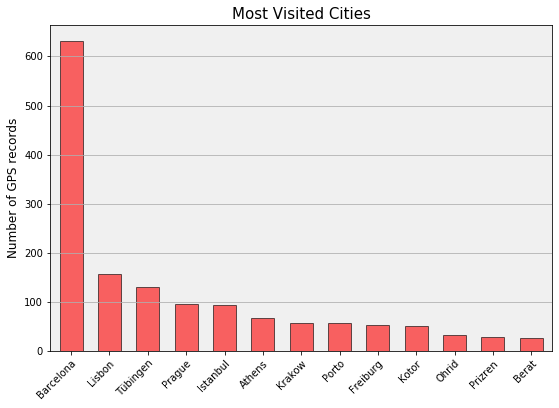
title = 'Most Visited Cities'

ylabel = 'Number of GPS records'

color = 'r'

bar\_chart(cities, title=title, ylabel=ylabel, color=color)

**out**



Ahora intente: use nuestra función bar\_chart() para crear un gráfico como el de arriba, pero asígnele una etiqueta en el eje x y use los valores de registro de las ciudades, como hicimos antes para los países.

**Next: scatter plots as simple maps**

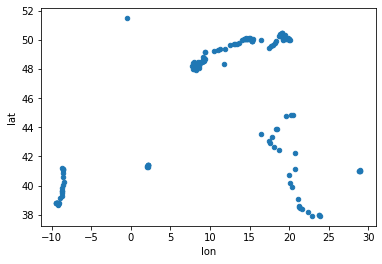
**In [17]:**

*# scatter plot the longitude and latitude data, with matplotlib defaults*

rs.plot(kind='scatter', x='lon', y='lat')

**Out[17]:**

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fb9c4e179b0>



In [18]:

*# get a single representative point from the data set for each of the most visited cities*

most\_visited\_cities\_index = df['city'].value\_counts().head(8).index

most\_visited\_cities = pd.DataFrame(df[df['city'].isin(most\_visited\_cities\_index)])

most\_visited\_cities.drop\_duplicates(subset=['city'], keep='first', inplace=**True**)

most\_visited\_cities

Out[18]:

|  | **lat** | **lon** | **city** | **country** |
| --- | --- | --- | --- | --- |
| **date** |  |  |  |  |
| **2014-05-14 15:11:00** | 38.781775 | -9.137544 | Lisbon | Portugal |
| **2014-05-17 15:13:00** | 41.145266 | -8.610441 | Porto | Portugal |
| **2014-05-19 00:04:00** | 41.377119 | 2.151632 | Barcelona | Spain |
| **2014-05-30 17:59:00** | 50.097330 | 19.895394 | Krakow | Poland |
| **2014-06-02 12:10:00** | 50.087729 | 14.660600 | Prague | Czech Republic |
| **2014-06-06 19:06:00** | 48.518270 | 9.058023 | Tübingen | Germany |
| **2014-07-05 05:32:00** | 37.967669 | 23.729203 | Athens | Greece |
| **2014-07-06 20:42:00** | 40.982352 | 28.815756 | Istanbul | Turkey |

In [20]:

fig, ax = plt.subplots()

fig.set\_size\_inches(10, 6)

rs\_scatter = ax.scatter(x=rs['lon'], y=rs['lat'], c='m', edgecolor='k', alpha=.4, s=150)

*# set font of tick labels*

**for** label **in** ax.get\_xticklabels():

label.set\_fontproperties(ticks\_font)

**for** label **in** ax.get\_yticklabels():

label.set\_fontproperties(ticks\_font)

*# set title, axis labels, background color*

ax.set\_title('Most Visited Cities', fontproperties=title\_font)

ax.set\_xlabel('Longitude', fontproperties=label\_font)

ax.set\_ylabel('Latitude', fontproperties=label\_font)

ax.set\_facecolor(axis\_bgcolor)

*# annotate each of the most visited cities on the map*

bbox\_style = {'boxstyle':'round', 'color':'k', 'fc':'w', 'alpha':0.8}

arrowprops\_style = {'arrowstyle':'->', 'connectionstyle':'arc3,rad=0.5', 'color':'k', 'alpha':0.8}

annotate\_cities = **lambda** row: ax.annotate(row['city'],

xy=(row['lon'], row['lat']),

xytext=(row['lon'] + 1, row['lat'] + 1),

fontproperties=annotation\_font,

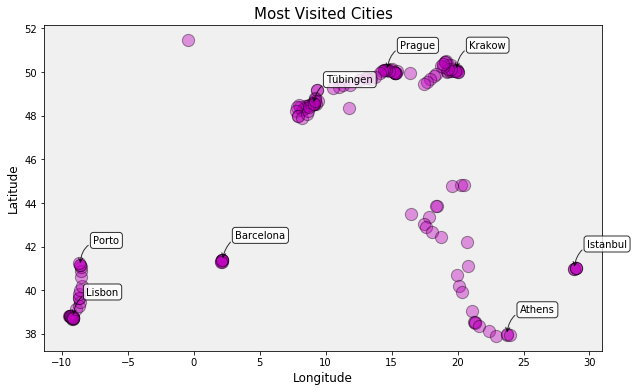
bbox=bbox\_style,

xycoords='data',

arrowprops=arrowprops\_style)

most\_visited\_cities.apply(annotate\_cities, axis=1)

plt.show()



Ahora intente: vuelva a crear el diagrama de dispersión anterior, pero dé a los puntos un tamaño y un alfa diferentes. Experimente moviendo las etiquetas de anotación y cambiando el estilo de flecha.

**Line charts, with annotation**

In [21]:

*# get the count of records by date*

countdata = df.groupby(df.index.date).size()

countdata.head()

**Out[21]:**

2014-05-14 36

2014-05-15 61

2014-05-16 57

2014-05-17 64

2014-05-18 50

dtype: int64

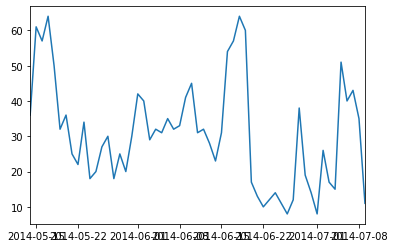
In [22]:

*# default matplotlib line chart*

countdata.plot(kind='line')

Out[22]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fb9c4d6ddd8>



In [24]:

*# create a nicer line plot*

fig, ax = plt.subplots()

ax = countdata.plot(kind='line',

figsize=[10, 5],

linewidth='3',

alpha=0.5,

marker='o',

color='c')

*# add annotations for explanation*

ax.annotate('Left the EU',

xy=('2014-06-20', 60),

fontproperties=annotation\_font,

bbox=dict(boxstyle='round', color='k', fc='w', alpha=0.7),

xycoords='data')

ax.annotate('Had WiFi',

xy=('2014-06-23', 40),

fontproperties=annotation\_font,

bbox=dict(boxstyle='round', color='k', fc='w', alpha=0.7),

xycoords='data')

ax.annotate('Return to EU',

xy=('2014-07-01', 53.5),

fontproperties=annotation\_font,

bbox=dict(boxstyle='round', color='k', fc='w', alpha=0.7),

xycoords='data')

*# set the x-ticks/labels for every nth row of the data: 1 tick mark per horizontal inch*

n = len(countdata) / int(fig.get\_size\_inches()[0]) *#interval between tick marks*

xtick\_data = countdata.iloc[range(0, len(countdata), 7)] *#rows spaced apart by that interval (ie, weekly)*

ax.xaxis.set\_ticks(xtick\_data.index) *#set the tick labels as the dates of those rows*

ax.set\_xlim(['2014-05-13', '2014-07-10'])

ax.xaxis.grid(**True**)

ax.yaxis.grid(**True**)

*# set tick labels, axis labels, and title*

ax.set\_xticklabels(xtick\_data.index, rotation=35, rotation\_mode='anchor', ha='right', fontproperties=ticks\_font)

**for** label **in** ax.get\_yticklabels():

label.set\_fontproperties(ticks\_font)

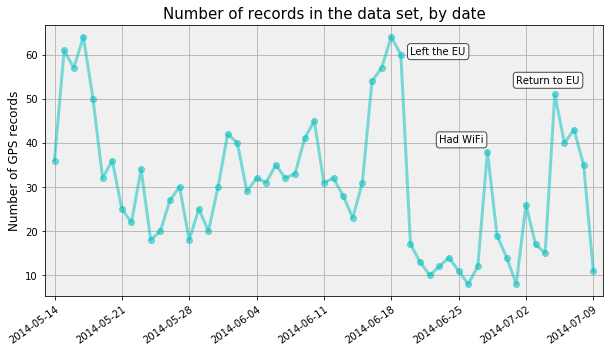
ax.set\_title('Number of records in the data set, by date', fontproperties=title\_font)

ax.set\_xlabel('', fontproperties=label\_font)

ax.set\_ylabel('Number of GPS records', fontproperties=label\_font)

ax.set\_facecolor(axis\_bgcolor)

plt.show()



Ahora intente: vuelva a crear el diagrama de arriba con diferentes anchos de línea, colores y marcadores de puntos. Juega moviendo las etiquetas de anotación. Haga las marcas verticales con 3 días de diferencia en lugar de 5.

**Pie charts**

**In [25]:**

cities = df['city'].value\_counts()

print(cities.count())

**out**

113

**[]**

cities.head()

Out[25]:

Barcelona 632

Lisbon 158

Tübingen 131

Prague 97

Istanbul 94

Name: city, dtype: int64

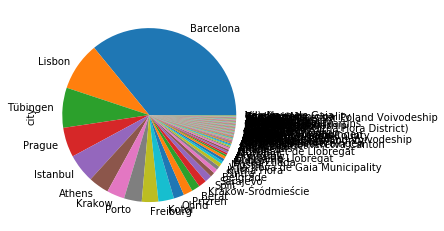
**In [26]:**

*# draw a pie chart nightmare*

cities.plot(kind='pie')

Out[26]:

<matplotlib.axes.\_subplots.AxesSubplot at 0x7fb9c4beae10>



In [27]:

*# function to produce more beautiful pie charts with matplotlib*

**def** pie\_chart(fractions, *#values for the wedges*

labels, *#labels for the wedges*

title = '', *#title of the pie chart*

cm\_name = 'Pastel1', *#name of the matplotlib colormap to use*

autopct = **lambda** x: str(round(x, 1)) + '%', *#format the value text on each pie wedge*

labeldistance = 1.05, *#where to place wedge labels in relation to pie wedges*

shadow = **True**, *#shadow around the pie*

startangle = 90, *#rotate 90 degrees to start the top of the data set on the top of the pie*

edgecolor = 'w', *#color of pie wedge edges*

width = 8, *#width of the figure in inches*

height = 8, *#height of the figure in inches*

grouping\_threshold = **None**, *#group all wedges below this value into one 'all others' wedge*

grouping\_label = **None**): *#what to label the grouped wedge*

*# if the user passed a threshold value, group all fractions lower than it into one 'misc' pie wedge*

**if** **not** grouping\_threshold==**None**:

*# if user didn't pass a label, apply a default text*

**if** grouping\_label == **None**:

grouping\_label = 'Others'

*# select the rows greater than the cut-off value*

row\_mask = fractions > grouping\_threshold

meets\_threshold = fractions[row\_mask]

*# group all other rows below the cut-off value*

all\_others = pd.Series(fractions[~row\_mask].sum())

all\_others.index = [grouping\_label]

*# append the grouped row to the bottom of the rows to display*

fractions = meets\_threshold.append(all\_others)

labels = fractions.index

*# get the color map then pull 1 color from it for each pie wedge we'll draw*

color\_map = cm.get\_cmap(cm\_name)

num\_of\_colors = len(fractions)

colors = color\_map([x/float(num\_of\_colors) **for** x **in** range(num\_of\_colors)])

*# create the figure and an axis to plot on*

fig, ax = plt.subplots(figsize=[width, height])

*# plot the pie*

wedges = ax.pie(fractions,

labels = labels,

labeldistance = labeldistance,

autopct = autopct,

colors = colors,

shadow = shadow,

startangle = startangle)

*# change the edgecolor for each wedge*

**for** wedge **in** wedges[0]:

wedge.set\_edgecolor(edgecolor)

*# set the title and show the plot*

ax.set\_title(title, fontproperties=title\_font)

plt.show()

**In [28]:**

cities = df['city'].value\_counts()

*# test out our function*

pie\_chart(fractions = cities,

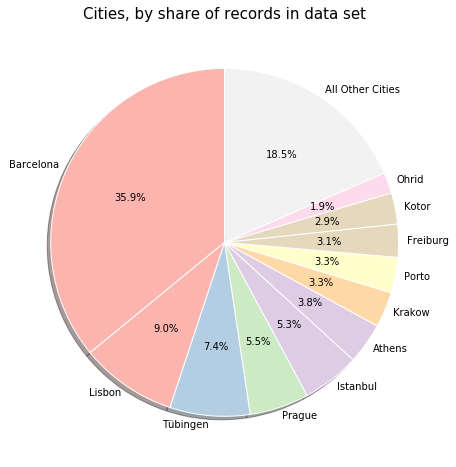
labels = cities.index,

title = 'Cities, by share of records in data set',

grouping\_threshold = 30,

grouping\_label = 'All Other Cities')

**out**



In [29]:

countries = df['country'].value\_counts()

*# convert the pie wedge percentage into its absolute value (instead of percent share)*

**def** my\_autopct(pct):

total = sum(countries)

**return** int(round((pct \* total) / 100.))

*# test out our function*

pie\_chart(fractions = countries,

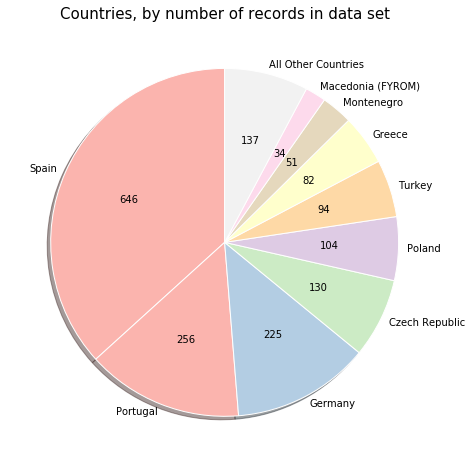
labels = countries.index,

title = 'Countries, by number of records in data set',

autopct=my\_autopct,

grouping\_threshold = 30,

grouping\_label = 'All Other Countries')



**Uso de Scatter Matrix**

Este ejemplo carga desde un archivo CSV datos con entradas mixtas numéricas y categóricas, y traza algunas cantidades, por separado para mujeres y hombres, gracias a la herramienta de trazado integrada de pandas (que utiliza matplotlib detrás de la escena).

Ver <http://pandas.pydata.org/pandas-docs/stable/visualization.html>

[]

data = pd.read\_csv('brain\_size.csv', sep=';', na\_values='.')

# Box plots of different columns for each gender

groupby\_gender = data.groupby('Gender')

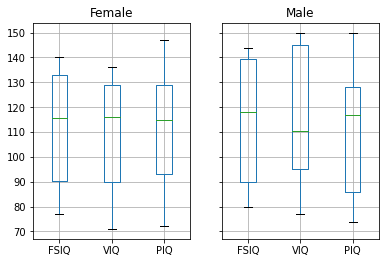
groupby\_gender.boxplot(column=['FSIQ', 'VIQ', 'PIQ'])

**out**

Female AxesSubplot(0.1,0.15;0.363636x0.75)

Male AxesSubplot(0.536364,0.15;0.363636x0.75)

dtype: object



[]

from pandas.plotting import scatter\_matrix

import matplotlib.pyplot as plt

# Scatter matrices para diferentes columnas

scatter\_matrix(data[['Weight', 'Height', 'MRI\_Count']])

scatter\_matrix(data[['PIQ', 'VIQ', 'FSIQ']])

plt.show()

out

