Programación para *Data Science*

Unidad 7: Análisis de datos en Python

Instrucciones de uso

A continuación se presentarán explicaciones y ejemplos de análisis de datos en Python. Recordad que podéis ir ejecutando los ejemplos para obtener sus resultados.

Introducción

En este módulo trabajaremos con librerías que ya hemos presentado en los módulos anteriores (NumPy, pandas y scikit-learn).

Este Notebook contiene ejemplos concretos de técnicas que pueden aplicarse para analizar los datos. Como en el módulo anterior, es importante destacar que se han seleccionado únicamente algunas técnicas pero, en la práctica, el conjunto de técnicas que se aplican para el análisis de datos es mucho más amplio. Además, para la mayoría de ejemplos usaremos las configuraciones por defecto incorporadas en las librerías, pero algunas de las funciones que probaremos tienen multitud de parámetros que podemos ajustar.

Primeros pasos

Para empezar, cargamos el conjunto de datos de flores de iris:

In [1]:

```
from sklearn import datasets

# Cargamos el dataset de iris:
iris = datasets.load_iris()
```

Análisis exploratorio de datos

En primer lugar, observaremos las características principales de los datos que utilizaremos en este Notebook. Conocer los datos con los que trabajaremos nos ayudará después en la creación de modelos y la validación de hipótesis.

Podemos echar un vistazo a la descripción del *dataset*:

In [2]:

```
print(iris.DESCR)
```

```
.. _iris_dataset:

Iris plants dataset
--------------------

**Data Set Characteristics:**

    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica

    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
```

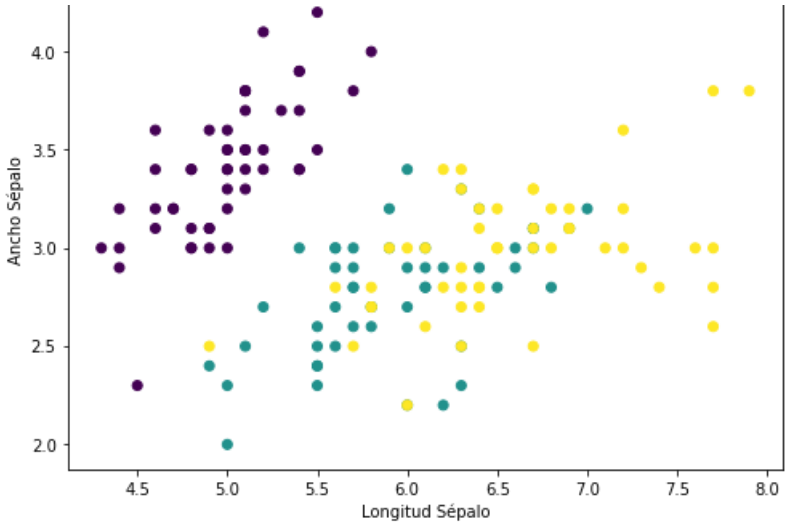
```
                Min  Max   Mean    SD   Class Correlation
    ============= ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20   0.76    0.9565  (high!)
    ============= ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

The famous Iris database, first used by Sir R.A. Fisher. The dataset is taken
from Fisher's paper. Note that it's the same as in R, but not as in the UCI
Machine Learning Repository, which has two wrong data points.

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field and
is referenced frequently to this day.  (See Duda & Hart, for example.)  The
data set contains 3 classes of 50 instances each, where each class refers to a
type of iris plant.  One class is linearly separable from the other 2; the
latter are NOT linearly separable from each other.

.. topic:: References

   - Fisher, R.A. "The use of multiple measurements in taxonomic problems"
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda, R.O., & Hart, P.E. (1973) Pattern Classification and Scene Analysis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Exposed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transactions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

En el conjunto Iris que acabamos de cargar, los datos están organizados de la siguiente forma: cada fila es una muestra y por cada muestra, las columnas (las características) son: longitud del sépalo, ancho del sépalo, longitud del pétalo y ancho del pétalo.

Representar visualmente los datos también nos permite realizar una primera aproximación a los mismos. Vamos a generar un *scatter plot* con los dos primeros atributos.

In [3]:

```
%matplotlib inline

# Importamos las librerías.
import matplotlib.pyplot as plt
from sklearn import datasets

# Importamos el dataset.
iris = datasets.load_iris()

# Seleccionamos solo los dos primeros atributos.
X = iris.data[:, :2]
Y = iris.target

# Creamos la figura.
plt.figure(1, figsize=(8, 6))
plt.clf()

# Coloreamos utilizando la categoría.
plt.scatter(X[:, 0], X[:, 1], c=Y)
plt.xlabel(u'Longitud Sépalo')
_ = plt.ylabel(u'Ancho Sépalo')
```



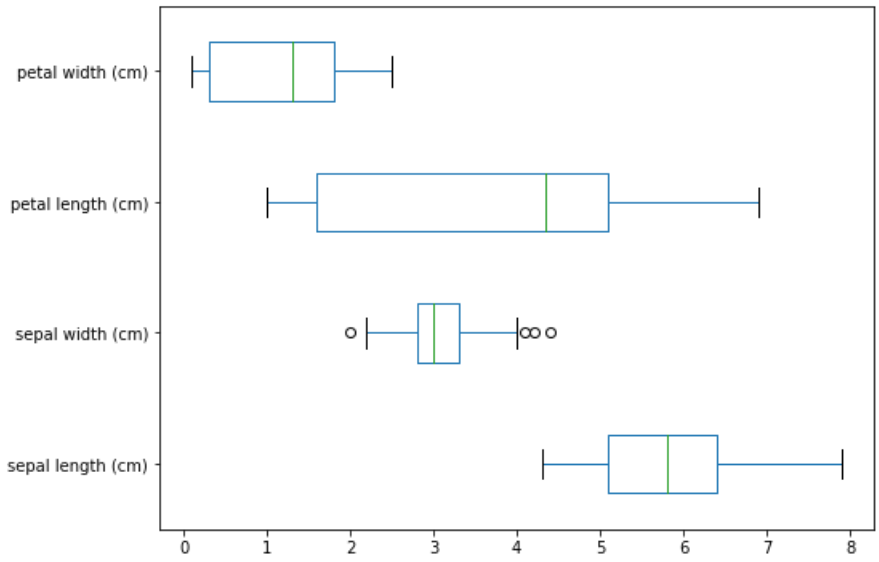
Después, creamos un *box plot* que resume los datos de todos los atributos disponibles.

In [4]:

```
# Cargamos los datos en un dataframe de pandas.
import pandas as pd
df = pd.DataFrame(iris.data, columns=iris.feature_names)
```

In [5]:

```
# Mostramos un box plot con los 4 atributos.
_ = df.plot.box(vert=False, figsize=(8, 6))
```



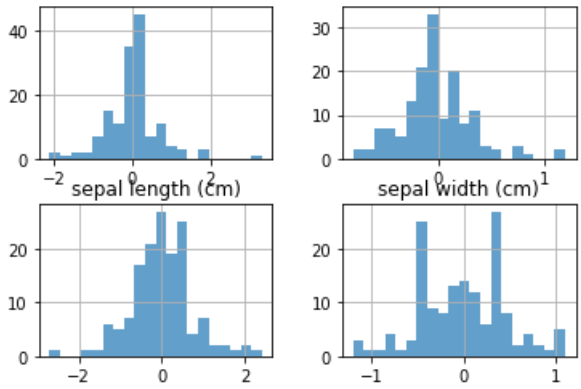
Finalmente, mostramos histogramas para los valores de cada atributo.

In [6]:

```
# Generamos los histogramas.
df.diff().hist(alpha=0.7, bins=20)
```

Out[6]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7fbc7c6887b8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbc7c63ac88>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7fbc7c5f8278>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7fbc7c5a8828>]],
      dtype=object)
```



Reducción de la dimensionalidad del dataset

Con el fin de manejar **la dimensionalidad** y evitar problemas como el sobre-ajuste, se utilizan métodos como el Análisis del Componente Principal (**Principal Component Analysis** o **PCA**). El PCA es un método que se utiliza para reducir el número de variables de los datos mediante la transformación y extracción de una muestra de datos relevante. De este modo se reduce la dimensión de los datos con el objetivo de retener la máxima información posible. En otras palabras, este método combina variables altamente correlacionadas para formar un número más reducido de un conjunto artificial de variables que se llaman "componentes principales" que explican la mayor parte de la variabilidad de los datos.

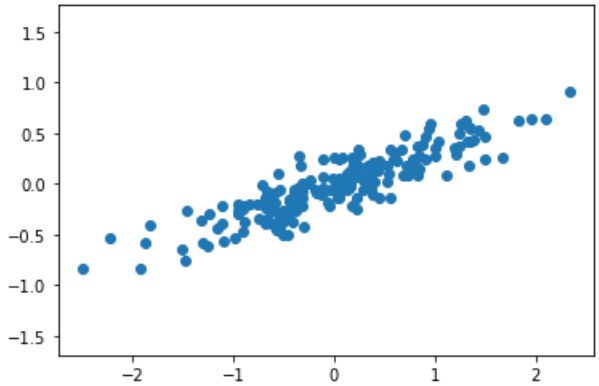
Por poner un ejemplo, consideraremos que queremos reducir la dimensionalidad del siguiente dataset de dos dimensiones:

In [2]:

```
# Importem les llibreries:
import matplotlib.pyplot as plt
import numpy as np

rng = np.random.RandomState(1)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal')
```

Out[2]:

```
(-2.7391278364515688,
 2.5801310701596343,
 -0.9477947579593763,
 1.0195904306706842)
```



A ojo, parece que hay una fuerte relación lineal entre las variables x (eje horizontal) e y (eje vertical). El PCA tiene el objetivo de encontrar la relación entre x e y de forma no supervisada. Esto lo consigue buscando una lista de los ejes principales de los datos y utilizando estos ejes para describir el conjunto de datos. Con el estimador de PCA de skearn, podemos calcularlo de la siguiente manera:

In [5]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca.fit(X)
```

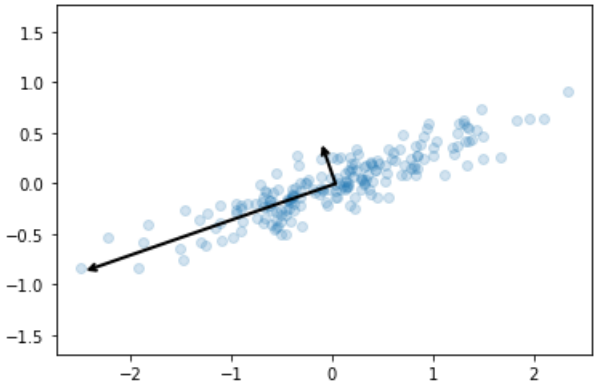
```
# Con la función fit() podemos calcular los "componentes principales" y "variación explicada" para cada uno:
print('Los componentes del PCA son :\n ' + str(pca.components_)+ '\n')

print('La variabilidad explicada por los componentes es:\n ' + str(pca.explained_variance_))
```

```
Los componentes del PCA son :
 [[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]

La variabilidad explicada por los componentes es:
 [0.7625315 0.0184779]
```

Pero qué significan estos números? A continuación visualizamos cómo a vectores sobre los datos de entrada, utilizando los **componentes** para definir la dirección del vector y la **varianza explicada** para definir la longitud cuadrada del vector:

In [6]:

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# muestra los datos
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```



Estos vectores representan los ejes principales de los datos, aquellos que **explican la mayor parte de la variabilidad de los datos**. En este caso vemos que el eje más relevante para resumir los datos es el más horizontal (apuntando a las 8 de un reloj), después lo complementa el eje más vertical (apuntando a las 11 de un reloj). Como son complementarios, su dirección es siempre ortogonal. Es por esta razón que tendremos un máximo de tantos componentes como dimensiones tenga el dataset con los que trabajamos. La longitud del vector indicacin la importancia que tiene este eje en la descripción de la distribución de los datos; más precisamente, es una medida de la varianza de los datos en el eje. En este caso, hemos partido de una dataset de 2 dimensiones, la x y la y. Si vulguesim reducir su dimensionalidad a una sola variable sabríamos que tenemos que transformar los datos projectatnt-las al eje más relevante, lo que corresponde con el componente principal, en este caso sabemos que sería el eje más horizontal.

Clasificación

Existen múltiples algoritmos de clasificación. Veamos un ejemplo de cómo usar un clasificador *k nearest neighbors* para predecir el tipo de especies de iris.

In [7]:

```
# Importamos el clasificador KNeighborsClassifier de la librería sklearn.
from sklearn.neighbors import KNeighborsClassifier
# Importamos NumPy.
import numpy as np
```

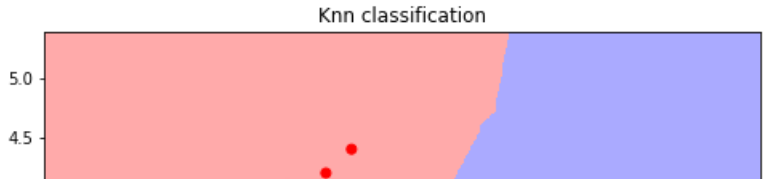
```
# Seleccionamos las dos primeras características (usaremos únicamente dos características para
# poder representar gráficamente los resultados en 2D).
X = iris.data[:, :2]
y = iris.target

# Separamos los datos (de manera aleatoria) en dos subconjuntos: el de aprendizaje y el de test.
indices = np.random.permutation(len(iris.data))
iris_X_train = X[indices[:-10]]
iris_y_train = y[indices[:-10]]
iris_X_test  = X[indices[-10:]]
iris_y_test  = y[indices[-10:]]

# Creamos el clasificador.
knn = KNeighborsClassifier()

# Entrenamos el clasificador.
knn.fit(iris_X_train, iris_y_train)

# Probamos el clasificador.
iris_y_test_predicted = knn.predict(iris_X_test)

# Mostramos los resultados de la predicción sobre el conjunto de test.
print("Clases reales: \t\t" + str(iris_y_test))
print("Clases predichas: \t" + str(iris_y_test_predicted))
print("Accuracy: \t\t" + str(knn.score(iris_X_test, iris_y_test)))
```

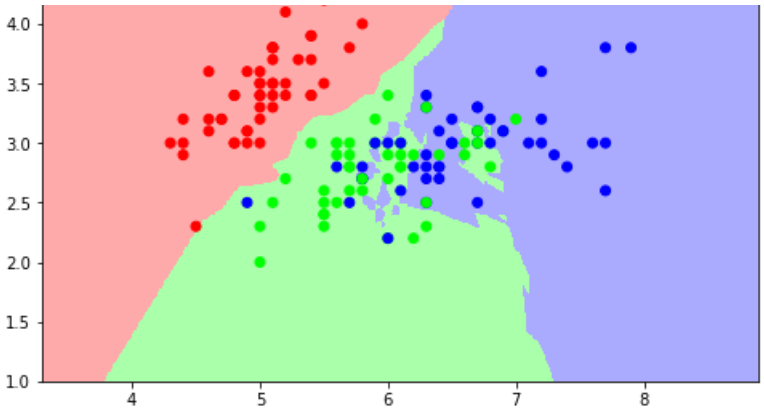
```
Clases reales:   [2 2 2 1 1 1 0 1 2 0]
Clases predichas:  [2 2 2 2 1 2 0 1 1 0]
Accuracy:   0.7
```

Podemos visualizar gráficamente el clasificador aprendido:

In [8]:

```
%matplotlib inline

# Importamos la librería.
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

# Creamos los mapas de colores que usaremos para la representación.
cmap_light = ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF'])
cmap_bold = ListedColormap(['#FF0000', '#00FF00', '#0000FF'])

# Calculamos los límites de la visualización.
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1

# Realizamos la predicción.
h = .01
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
Z = knn.predict(np.c_[xx.ravel(), yy.ravel()])

# Mostramos el resultado en una figura.
plt.figure(1, figsize=(8, 6))
Z = Z.reshape(xx.shape)
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Mostramos las muestras utilizadas en el aprendizaje.
plt.scatter(iris_X_train[:, 0], iris_X_train[:, 1], c=iris_y_train, cmap=cmap_bold)

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("Knn classification")

plt.show()
```





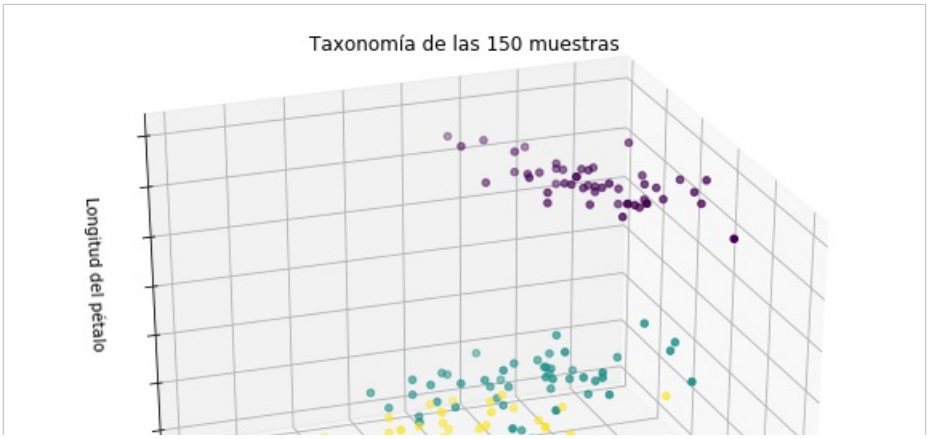
Clústering

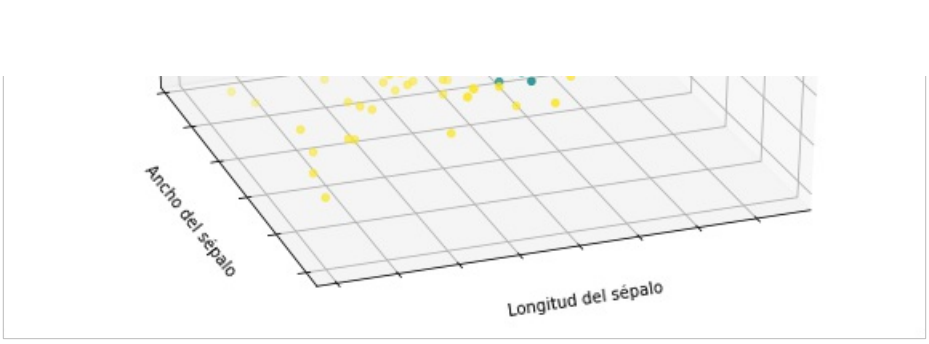
Como con los algoritmos de clasificación, actualmente existen multitud de algoritmos de clústering. Veámos un ejemplo de utilización del algoritmo *k-means*.

En primer lugar, generamos una visualización del conjunto de muestras. A continuación tenéis un código de ejemplo en el que representamos la taxonomía de las diferentes muestras (coloreamos por clase de Iris) dependiendo de la longitud del sépalo (columna 0), ancho del sépalo (columna 1) y longitud del pétalo (columna 2):

In [9]:

```
%matplotlib inline

# Cargamos las librerías necesarias.
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import datasets

# Cargamos el dataset.
iris = datasets.load_iris()
# Datos de la muestra
X_iris = iris.data
# Categorías de la muestra (tres tipos de iris)
Y_iris = iris.target

# Creamos una figura.
fig = plt.figure(1, figsize=(8, 6))
# De tipo 3D
ax = Axes3D(fig, elev=-150, azim=110)
# Y representamos los diferentes puntos, coloreando por tipo de Iris
ax.scatter(X_iris[:,[0]], X_iris[:,[1]], X_iris[:,[2]], c=Y_iris)

# Leyendas y títulos
ax.set_title(u"Taxonomía de las 150 muestras")
ax.set_xlabel(u"Longitud del sépalo")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel(u"Ancho del sépalo")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel(u"Longitud del pétalo")
_ = ax.w_zaxis.set_ticklabels([])
```

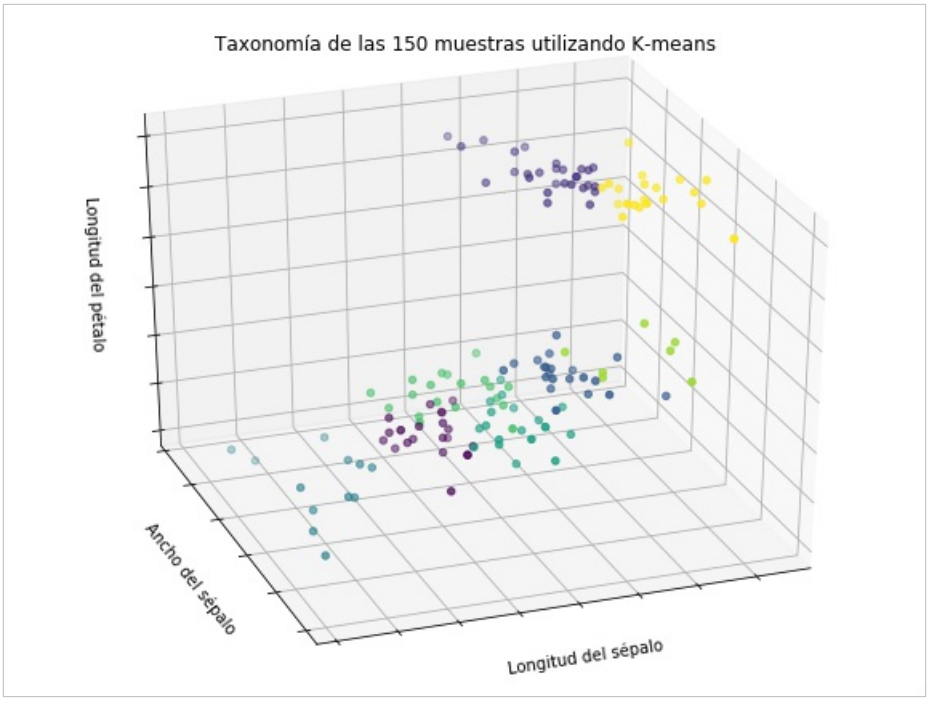




Ahora vamos a hacer el siguiente experimento: utilizando el algoritmo de clústering *k-means*, vamos a colorear utilizando los grupos que calcule el algoritmo y no las clases que ya conocemos:

In [10]:

```
%matplotlib inline

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import cluster, datasets

iris = datasets.load_iris()
X_iris = iris.data

# Cargamos el algoritmo K-means y hacemos fit a nuestros datos:
k_means = cluster.KMeans()
k_means.fit(X_iris)

fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
ax.scatter(X_iris[:,[0]], X_iris[:,[1]], X_iris[:,[2]], c=k_means.labels_)
ax.set_title(u"Taxonomía de las 150 muestras utilizando K-means")
ax.set_xlabel(u"Longitud del sépalo")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel(u"Ancho del sépalo")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel(u"Longitud del pétalo")
_ = ax.w_zaxis.set_ticklabels([])
```



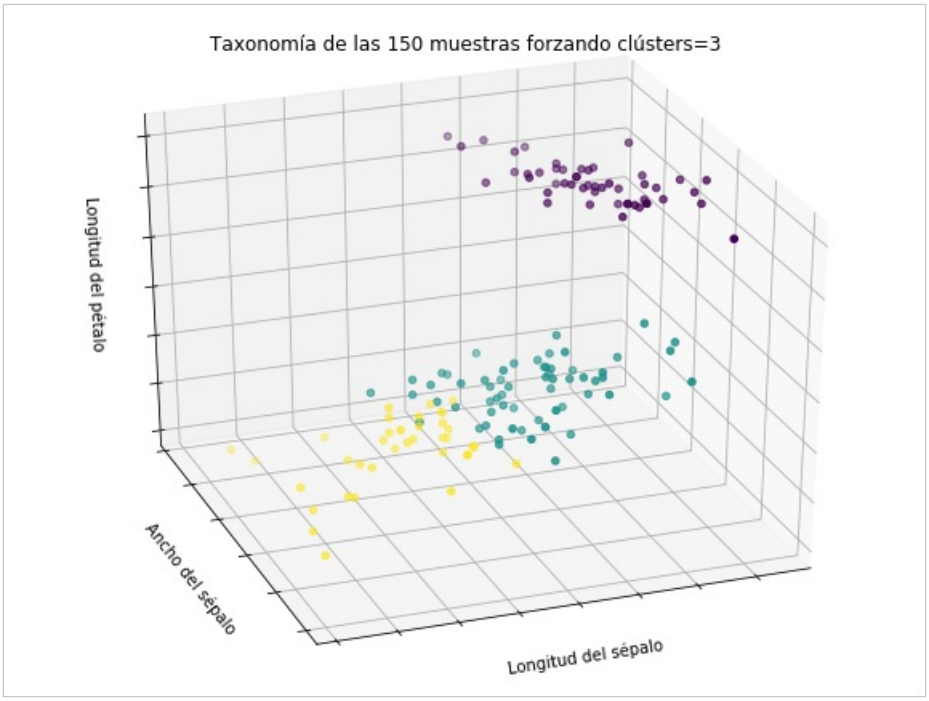
¡Notad que en **ningún momento se utiliza la clase de la muestra** (`iris.target`) para entrenar el algoritmo ni para evaluarlo! Ahora estamos utilizando un algoritmo de clústering, que agrupará las muestras en función de las características de las mismas. El resultado del algoritmo es el grupo al que pertenece cada muestra (pero el algoritmo no intenta predecir la clase de la muestra). Los nombres de los grupos generados son arbitrarios (en este caso, valores enteros del 0 al número de grupos - 1).

Vamos ahora a forzar que el número de clústers sea igual a 3 y vamos a representar el resultado:

Vamos ahora a forzar que el número de clusters sea igual a 3 y vamos a representar el resultado:

In [11]:

```
%matplotlib inline

import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from sklearn import cluster, datasets

iris = datasets.load_iris()
X_iris = iris.data

# Cargamos el algoritmo K-means y hacemos fit a nuestros datos
# esta vez forzando el número de clústers a tres:
k_means = cluster.KMeans(n_clusters=3)
k_means.fit(X_iris)

fig = plt.figure(1, figsize=(8, 6))
ax = Axes3D(fig, elev=-150, azim=110)
ax.scatter(X_iris[:,[0]], X_iris[:,[1]], X_iris[:,[2]], c=k_means.labels_)
ax.set_title(u"Taxonomía de las 150 muestras forzando clústers=3")
ax.set_xlabel(u"Longitud del sépalo")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel(u"Ancho del sépalo")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel(u"Longitud del pétalo")
_ = ax.w_zaxis.set_ticklabels([])
```



Recordad que utilizando un algoritmo de clústering no aprendemos a qué clase pertenece cada muestra sino que simplemente agrupamos las muestras en grupos (clústers).

Validación del modelo

Debemos evitar evaluar los modelos con los mismos datos que se han utilizado para el aprendizaje. En el ejemplo de clasificación, hemos separado los datos de manera aleatoria en dos conjuntos, uno para el aprendizaje y uno para el test. Está técnica se conoce como *holdout*. En el ejemplo de la clasificación hemos usado numpy para crear los dos conjuntos. En la unidad 6 vimos cómo realizar este mismo proceso usando las funciones sobre *dataframes* que ofrece la libería pandas. Ahora veremos cómo podemos hacerlo usando sklearn:

In [12]:

```
# Importamos la función 'train_test_split'.
from sklearn.model_selection import train_test_split
```

```
# Separamos las muestras utilizando un 20 % para test y el resto para aprendizaje.
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.2, random_s
tate=0)

print("Total de muestras: " + str(len(iris.data)))
print("Aprendizaje: " + str(len(X_train)), "(" + str(float(len(X_train))/len(iris.data)*100) + "%)"
)
print("Test: " + str(len(X_test)), "(" + str(float(len(X_test))/len(iris.data)*100) + "%)")
```

```
Total de muestras: 150
Aprendizaje: 120 (80.0%)
Test: 30 (20.0%)
```

También podemos usar otras técnicas para evaluar los modelos, por ejemplo, *kfold* o *Leave One Out*:

In [13]:

```
# Importamos la función KFold.
from sklearn.model_selection import KFold
# Importamos NumPy.
import numpy as np

# Particionamos un conjunto de nueve muestras usando 3-Fold y mostramos el resultado.
X = np.array(range(9))
kf = KFold(n_splits=3)
for train, test in kf.split(X):
    print("%s %s" % (X[train], X[test]))
```

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

In [14]:

```
# Importamos la función LeaveOneOut.
from sklearn.model_selection import LeaveOneOut
# Importamos NumPy.
import numpy as np

# Particionamos un conjunto de nueve muestras usando LeaveOneOut y mostramos el resultado.
X = np.array(range(9))
loo = LeaveOneOut()
for train, test in loo.split(X):
    print("%s %s" % (X[train], X[test]))
```

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