Capacitación Python
Laboratorio 3

Nelson R. Salinas

Abril 10, 2018

La programación basada en objetos es, en gran medida, una de las principales ventajas de Python. En esta sesión se presentarán sus ventajas a través de algunos ejercicios con la librería "Pandas". También se realizarán algunos ejercicios para aplicar el concepto de ámbito y espacio de nombres.

1. El problema

Uno de los conjuntos de datos más famosos en ciencia de datos y aprendizaje de máquinas las medidas florales de tres especies del género *Iris*. Las medidas son longitud y amplitud de los sépalos y longitud y amplitud de los pétalos. Generalmente la pregunta asociada a este conjunto de datos es ¿pueden identificarse estas tres especies sólo sobre la base de las medidas de los sépalos y pétalos? Para responder esta pregunta es necesario explorar los datos y conocer sus propiedades.

2. Módulos y ámbito

Frecuentemente el programador necesita métodos que ya han sido implementados en Python. Para cargar en memoria dicho código se utiliza la función `import`.

Las maneras más comunes de incluir código externo están presentadas en el Código 1. Dos aspectos son clave: la completitud del proceso de importación y el *espacio de nombres* bajo en cual se van a incluir los elementos en el actual del código.

Por ejemplo, suponga que necesita un componente (clase DataFrame) del módulo Pandas. En el Código 1 se observan las cuatro formas de lograr dicho objetivo. Las opciones de las líneas 1, 2 y 3 incluyen todo el módulo (no sólo la clase DataFrame), mientras que la opción 4 sólo incluye dicha clase. La diferencia entre las opciones 1, 2 y 3 radica en el espacio de nombre resultante. Si se utilizan las opciones 1 y 2 cada vez se se requiera la clase DataFrame debe adjuntarse el nombre del módulo (`pandas.DataFrame` o `pd.DataFrame`), mientras que en la opción 3 no es necesario (simplemente `DataFrame`). Por otro lado, la opción 2 difiere de 1 porque utiliza un nombre alias para el módulo (`pd`). Para el presente ejercicio vamos a utilizar la opción de la línea 2.

Código 1: Uso de la función `import`.

```
1 import pandas
2 import pandas as pd
3 from pandas import *
4 from pandas import DataFrame
```

3. Instalación de la librería Pandas

Si en la primera sesión instaló Anaconda no es necesario intalar Pandas. Pero si por el contrario solo instaló Miniconda entonces es necesario instalar esta librería, para lo cual debe abrir la terminal asociada a Miniconda y ejecutar los comandos `conda install pandas`. Luego el sistema le pedirá confirmar la instalación.

4. Pandas

El módulo Pandas proporciona dos estructuras de datos bastante útiles para la manipulación de tablas de datos: DataFrame y Series. Ambas estructuras está implementado como una clase en Python (esto puede verificarse digitando `??pd.DataFrame` en ipython o Jupyter, que muestra el código fuente de dicha utilidad). La diferencia entre ambas clases radica en la dimensionalidad: DataFrame es una tabla bimensional, mientras que Series es una columna sencilla de datos. Afortunadamente Pandas proporciona funciones que realizan la creación de instancias de DataFrame de acuerdo a la naturaleza de los archivos en los cuales la información está guardada. Por ejemplo, si los datos están guardados como archivos csv se puede crear un DataFrame con la función `pd.read.csv`.

Código 2: Uso de la función `import`.

```
1 import pandas as pd
2
3 iris = pd.read_csv("iris-species/Iris.csv")
```

Si se ingresa el el nombre del DataFrame (`iris`) Python mostrará una cantidad determinada de filas (si la DataFrame es pequeña mostrará toda la tabla).

Estos son algunos de los atributos más útiles:

iris.columns Produce una lista con los nombres de las columnas de la tabla.

iris.shape Muestra el número de filas y columnas de la tabla.

iris.dtypes Indica qué tipo de datos son cada una de las columnas de la tabla. Nótese que las columnas de texto son del tipo “object”.

iris.size Muestra el número de celdas en la tabla.

Y algunos métodos:

iris.head(#) Imprime las primeras # filas.

iris.describe() Calcula los descriptores estadísticos básicos de las columnas numéricas: media, desviación estándar, cuartiles y valores mínimo y máximo.

iris.hist(bins=#) Produce un histograma de cada una de las columnas con # barras. Necesita la inclusión del módulo matplotlib.pyplot.

iris.groupby(columnas) Realiza agrupamiento de datos por valores de columna(s).

Casi todos los atributos y métodos de Dataframes también están disponibles para Series.

4.1. Seleccion de datos

Una de las mayores ventajas de utilizar DataFrames es la facilidad con la cual se pueden seleccionar y filtrar datos. La selección de columnas se realiza con el nombre de la misma:

```
1 iris.Species
2 iris['Species']
```

Para seleccionar filas el proceso es un poco más complicado. Si se van a seleccionar varias filas al tiempo a manera de tajada (varias filas consecutivas) se puede utilizar la notación de slice clásica de Python:

```
1 iris[10:20]
```

Pero se se va seleccionar una fila específica es necesario utilizar el atributo `loc`:

```
1 iris.loc[12]
```

Este atributo no solo permite seleccionar fila, también columnas y celdas:

```
1 iris.loc[12,'Species'] \# celda
2 iris.loc[:,'Species'] \# columna Species
3 iris.loc[10:20, ['PetalWidthCm', 'Species']]
```

Otra posibilidad incluye seleccionar datos dadas condiciones de búsqueda. Por ejemplo, podemos estar interesados solo en los datos de la especie *Iris setosa*:

```
1 iris[iris.Species == 'Iris-setosa']
```

O los datos de *Iris setosa* cuyos pétalos son mayores a 5 cm:

```
1 iris[(iris.Species == 'Iris-setosa') & (iris.SepalLengthCm > 5)]
```

Nótese que en este último caso todas las condiciones están rodeadas de paréntesis, y el operador de conjunción lógica ahora es '&' envés de 'and'. Los otros operadores son '|' (disyunción, 'or') y '$\sim$' (negación, 'not').

```
1 iris[~(iris.Species == 'Iris-setosa') & (iris.SepalLengthCm > 5)]
2
3 iris[((iris.Species == 'Iris-virginica') | (iris.Species == Iris-
     versicolor)) & (iris.SepalLengthCm > 5)]
```

¿Cómo seleccionaría a los registros tanto de Iris-virginica con sépalos hasta 3 cm de ancho y los de Iris setosa con pétalos de 1 cm o más de ancho?

5. Exploración de datos

Varios atributos y métodos de los DataFrames de Pandas facilitan la exploración inicial de datos. Una de dichas tareas es la obtención del tipo de datos (numérico real, numérico entero, texto, etc.) y los rangos de valores de cada columna. Como se mencionó arriba el atributo `dtypes` nos muestra lo primero, mientras que el método `describe` muestra lo segundo. El método describe, sin embargo, sólo aplica a campos numérico. Para campos de texto se puede utilizar la función `unique` para mostrar los diferentes valores registrados.

```
1 iris.dtypes
2 iris.describe()
3 iris.Species.unique()
```

Adicionalmente, para facilitar la compresión de los datos en cuestión, se pueden realizar gráficas descriptivas. Por ejemplo, se puede realizar un histograma de cada una de las variables. Para realizar tales gráficos es necesario cargar en memoria el módulo "matplotlib.pyplot". Dado que el nombre de este módulo es bastante grande, usaremos un alias. Como se mencionó arriba, los histogramas pueden ser producidos a través del método `hist`. ¿Qué les demuestra el histograma de la longitud de pétalos?

```
1 import matplotlib.pyplot as plt
2 iris.PetalLengthCm.hist(bins=30)
3 plt.show()
```

Otro gráfico de utilidad son las gráficas de dispersión entre dos o tres variables. ¿Qué nos indica la gráfica de dispersión entre la longitud y amplitud de los pétalos? ¿Existe el mismo patrón con las dimensiones de los sépalos?

```
1 iris.plot(kind='scatter', x='PetalLengthCm', y='PetalWidthCm')
2 plt.show()
```

Sin embargo, para entender las variaciones de las variables entre las diferentes especies sería bastante útil modificar el gráfico para indicar la membresía de las especies. Para ellos inicializamos un objeto de gráfico con la primera ilustración y luego proyectamos los puntos de las siguientes ilustraciones con el parámetro `ax`. También es necesario cambiar el color de los punto con el parámetro `color` y añadir una leyenda con el parámetro `label`.

```
1 sp = iris[iris.Species == 'Iris-setosa'].plot(kind='scatter', x='
     PetalLengthCm', y='PetalWidthCm', color='green', label='setosa')
2 iris[iris.Species == 'Iris-versicolor'].plot(kind='scatter', x='
     PetalLengthCm', y='PetalWidthCm', color='blue', label='versicolor'
     , ax=sp)
3 iris[iris.Species == 'Iris-virginica'].plot(kind='scatter', x='
     PetalLengthCm', y='PetalWidthCm', color='red', label='virginica',
     ax=sp)
4 plt.show()
```