Clase 7: Manejo de Datos en Python.

FM849 - Programación Científica para Proyectos de Inteligencia Artificial (IA)

Contenidos de hoy

- Introducción a Pandas en Python.
- Estructuras de Datos en Pandas (Series y DataFrames).
- Como cargar datos en Pandas.
- Como acceder a datos en Pandas.
- Como guardar datos en Pandas.

Paquete Pandas en Python

Pandas es una librería muy flexible que permite manejar y visualizar datos.

- Contiene estructuras y herramientas para manejar datos de manera conveniente.
- Tiene metodos para visualizar datos.

Por que Pandas ?

- Contiene estructuras y herramientas para manejar datos de manera conveniente.
- Es mucho mas flexible que otros programas (ej. Excel), por lo que nos permite hacer cosas mas complicadas y útiles para aplicaciones de IA.
- Es capaz de manejar grandes datasets con muchos ejemplos.

Empezando con Pandas

Primero importamos el paquete para hacer uso de todas sus funciones.

```
import pandas # importamos el paquete
data = pandas.read_csv("datos.csv") #usamos las funciones del paquete
```

Existe una manera mas conveniente de hacer esto.

```
import pandas as pd # importamos el paquete
data = pd.read_csv("datos.csv") #usamos las funciones del paquete
```

Así, evitamos escribir *pandas* en cada llamada a una función. También, comúnmente usaremos la librería *numpy* en conjunto con pandas.

```
import numpy as np
```

Antes de Pandas: Diccionarios.

El diccionario es una estructura de datos que almacena datos en forma **llave:valor**.

```
>>> country_capitals = {
  "Germany": "Berlin",   # item 1
  "Canada": "Ottawa",    # item 2
  "England": "London",   # item 3
  }

>>> country_capitals["Canada"] # buscamos la capital de Canada
```

Estructuras de Datos en Pandas.

Usando diccionarios podemos crear nuevas estructuras de datos. Por ejemplo, Series y DataFrames.

Dim	Nombre	Descripción
1D	pd.Serie	Arreglo de elementos 1D de un solo tipo
2D	pd.DataFrame	Arreglo 2D etiquetado mutable con columnas de tipo Serie.

Tabla 1: Tipos de estructuras de datos en Pandas.

Objetos de tipo Series

Un objeto de tipo *Series* es un arreglo 1-dimensional que contiene una secuencia de valores del mismo tipo. Este arreglo tiene asociado un arreglo de etiquetas, llamado *index*.

- Podemos crear un objeto *Series* usando un diccionario.

```
>>> dicc = {"a":3,"b":2,"c":1}
>>> obj = pd.Series(dicc)
>>> obj
```

Retorna

```
a 3
b 2
c 1
dtype: int64
```

Objetos de tipo Series

Un objeto de tipo *Series* es un arreglo 1-dimensional que contiene una secuencia de valores del mismo tipo. Este arreglo tiene asociado un arreglo de etiquetas, llamado *index*.

- Podemos crear un objeto *Series* usando una lista.

```
>>> obj = pd.Series([4, 7, -5, 3])
>>> obj
```

Retorna

```
0  4
1  7
2 -5
3  3
dtype: int64
```

Objetos de tipo DataFrame

Un objeto de tipo *DataFrame* representa una tabla rectangular de datos. Esta contiene una colección de columnas, donde cada una puede ser de un tipo diferente.

- Podemos crear un objeto *DataFrame* usando un diccionario donde cada *item* representa una columna.

```
data = {
"Name": ["Alice", "Bob", "Charlie"],
"Age": [25, 30, 35],
"City": ["Berlin", "Ottawa", "Londres"]
}
df = pd.DataFrame(data)
df
```

	Name	Age	City
0	Alice	25	Berlin
1	Bob	30	Ottowa
2	Charlie	35	Londres

Objetos de tipo DataFrame

Un objeto de tipo *DataFrame* representa una tabla rectangular de datos. Esta contiene una colección de columnas, donde cada una puede ser de un tipo diferente.

```
In[1]: data = {"state": ["Ohio", "Ohio", "Ohio", "Nevada", "Nevada"],
"year": [2000, 2001, 2002, 2002, 2003],
"pop": [1.5, 1.7, 3.6, 2.9, 3.2]}
In[2]: frame = pd.DataFrame(data)
```

```
Out[2]:
state year pop
0 Ohio 2000 1.5
1 Ohio 2001 1.7
2 Ohio 2002 3.6
3 Nevada 2002 2.9
4 Nevada 2003 3.2
```

DataFrames a partir de datasets

En los ejemplos anteriores generamos Series y DataFrames usando datos creados de manera manual. Sin embargo, en proyectos reales debemos cargar datos desde datasets o paginas webs. Esto se puede realizar con las siguientes funciones (Tabla 2).

Función	**Uso**	**Ejemplo**
read_csv	Cargar datos delimitados por comas desde un archivo local o una URL	*pd.read_csv("datos.csv")*
read_excel	Cargar datos tabulares desde un archivo Excel	*pd.read_excel("datos.xlsx")*
read_pickle	Cargar datos serializados en formato pickle	*pd.read_pickle("datos.pkl")*
read_json	Cargar datos desde archivos JSON	*pd.read_json("datos.json")*

Tabla 2: Funciones de Pandas para carga de datos

Básicos de Pandas

Ahora veremos que cosas basicas podemos hacer con un DataFrame. Volveremos a crear un pequeño dataset.

```
data = {
"Name": ["Alice", "Bob", "Charlie", "Kim"],
"Age": [45, 50, 35, 20],
"City": ["Berlin", "Ottawa", "Londres", "Hong Kong"]
}
df = pd.DataFrame(data)
df
```

	Name	Age	City
0	Alice	45	Berlin
1	Bob	50	Ottowa
2	Charlie	35	Londres
3	Kim	20	Hong Kong

Metodo Head

- *df*.*head*() para mirar las primeras filas.

	Name	Age	City
0	Alice	45	Berlin
1	Bob	50	Ottawa
2	Charlie	35	London
3	Kim	20	Hong Kong

`df.head(2)` →

	Name	Age	City
0	Alice	45	Berlin
1	Bob	50	Ottawa

Metodo tail

- *df*.*tail*() para mirar las ultimas filas.

	Name	Age	City
0	Alice	45	Berlin
1	Bob	50	Ottawa
2	Charlie	35	London
3	Kim	20	Hong Kong

`df.tail(2)` →

	Name	Age	City
2	Charlie	35	London
3	Kim	20	Hong Kong

Atributo shape

- *df .shape* devuelve la forma del DataFrame (filas,columnas).

	Name	Age	City
0	Alice	45	Berlin
1	Bob	50	Ottawa
2	Charlie	35	London
3	Kim	20	Hong Kong

`df.shape` →

(4,3)

Funcion len

- *len*(*df*) devuelve el largo del DataFrame medido en numero de filas.

	Name	Age	City
0	Alice	45	Berlin
1	Bob	50	Ottawa
2	Charlie	35	London
3	Kim	20	Hong Kong

4

Atributo columns

- *df.columns* devuelve los nombres de todas las columnas del DataFrame.

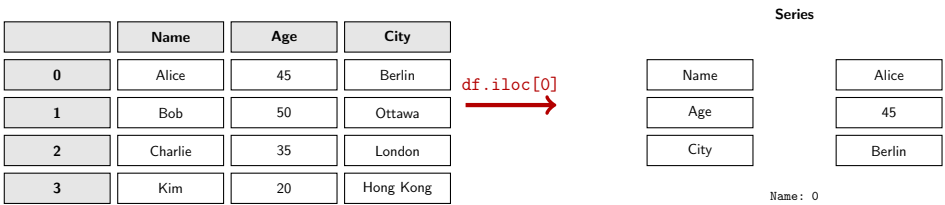
	Name	Age	City
0	Alice	45	Berlin
1	Bob	50	Ottawa
2	Charlie	35	London
3	Kim	20	Hong Kong

`df.columns` →

["Name",.^ge",City"]

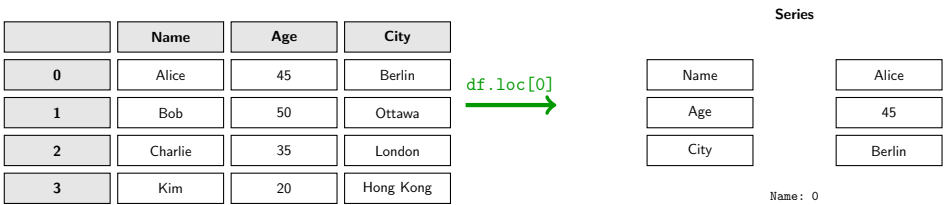
Metodo iloc

- $df.iloc(k)$ devuelve una Serie generada con la fila en la posición k.



Metodo loc

- $df.loc(k)$ devuelve una Serie generada con la fila indexada por el valor k.



Como guardar datos con Pandas ?

Lo ultimo que vamos a ver es como guardar un objeto de tipo DataFrame, podemos guardar este usando las siguientes funciones (Tabla 3).

Función	**Uso**	**Ejemplo**
to_csv	Guardar un DataFrame en un archivo delimitado por comas (CSV)	*df.to_csv("datos.csv")*
to_excel	Guardar un DataFrame en un archivo Excel	*df.to_excel("datos.xlsx")*
to_pickle	Serializar y guardar un DataFrame en formato pickle	*df.to_pickle("datos.pkl")*
to_json	Guardar un DataFrame en un archivo JSON	*df.to_json("datos.json")*

Tabla 3: Funciones de Pandas para guardar datos

Resumen de DataFrames en Pandas.

- $df.read_csv()$ para cargar un dataset desde un archivo csv o URL.
- $df.head()$ para mirar las primeras filas.
- $df.tail()$ para mirar las ultimas filas.
- $df.shape$ para obtener las dimensiones de la tabla.
- $len(df)$ para obtener el numero de filas en la tabla.
- $df.columns$ para obtener los nombres de las columnas en la tabla.
- $df.iloc[k]$ para obtener la fila ubicada en la posición k.
- $df.loc[index]$ para obtener la fila asignada al indice $index$.
- $df.to_csv()$ para guardar el DataFrame en un archivo csv.

Pandas tiene mas funciones! (veremos esto mañana...).

Resumen de la clase.

- Diccionarios de Python.
- pd.Series de Python.
- pd.DataFrames de Python.
- Funciones asociadas a la clase pd.DataFrame.

Vamos a ver un ejemplo practico en Python Collab

Exploremos un dataset: https://colab.research.google.com/drive/1X7xoTisfFrGZFfrQusp=sharing.

Figura 1: Pokemones de primera generación.

Referencias:

- Wes McKinney. (2022). Python for Data Analysis. Third Edition.