Guia Pandas.

Introducció a *pandas*

Què és pandas?

pandas és un paquet de Python per a la manipulació de dades en format tabular (files i columnes), organitzades en estructures anomenades *DataFrame*. Intuïtivament, un DataFrame s'assembla a un full d'Excel, però amb operacions vectoritzades i integració completa amb l'ecosistema científic de Python.

Per a què s'utilitza *pandas*?

És útil al llarg de tot el flux d'anàlisi de dades. Amb *pandas* pots:

- Importar datasets des de CSV, Excel, bases de dades, JSON, etc.
- Netejar dades (tratar valors nuls, duplicats, tipus incorrectes...).
- Endreçar i transformar (filtrar, ordenar, seleccionar columnes, pivotar/reshapejar).
- Agregar i resumir (mitjanes, comptatges, correlacions, estadístics bàsics).
- Treballar amb sèries temporals i dades de text.
- Visualitzar ràpidament (integració amb Matplotlib/Altair/Seaborn, etc.).

Beneficis clau del paquet *pandas*

- **Dissenyat per a Python:** integra molt bé amb NumPy, SciPy, scikit-learn, etc.
- **Poc verbós per operació:** moltes operacions es resolen en poques línies clares.
- **Vista intuïtiva de les dades:** els DataFrames faciliten entendre i explorar.
- **Joc d'eines extens:** EDA, tractament de nuls, estadística bàsica, reshape, merge/join, i més.
- **Escala raonable:** pot gestionar milions de files depenent de la màquina (memòria).

Com instal·lar *pandas*?

Pots instal·lar *pandas* amb `pip` o amb `conda` (si treballes amb entorns conda).

Instal·lació amb pip

```
pip install pandas
```

Instal·lació amb conda

```
conda install pandas
# (opcional) des del canal conda-forge:
# conda install -c conda-forge pandas
```

Verificar la instal·lació

```
import pandas as pd
print(pd.__version__)  # mostra la versió instal·lada
```

Primer exemple ràpid

Carreguem un CSV i fem algunes operacions bàsiques:

```
import pandas as pd

# Llegeix un CSV (substitueix 'fitxer.csv' pel teu arxiu)
df = pd.read_csv("fitxer.csv")

# Veure les primeres files
print(df.head())

# Seleccionar columnes i filtrar files
subset = df[["col1", "col2"]]
filtrat = df[df["col_num"] > 0]

# Estadístics ràpids
print(df.describe())
```

```
# Agrupar i agregar
agrupat = df.groupby("categoria")["import"].mean()
print(agrupat)

# Desa a CSV
df.to_csv("sortida.csv", index=False)
```

Enllaços útils

- Documentació oficial de pandas
- User Guide (temes per capítols)
- API Reference

Importar dades amb pandas

Importar el paquet pandas

Per començar a treballar amb **pandas**, primer s'ha d'importar el paquet. Normalment, s'utilitza l'àlies pd:

```
import pandas as pd
```

Importar fitxers CSV

La funció `read_csv()` permet llegir fitxers separats per comes (*comma-separated values*):

```
df = pd.read_csv("diabetes.csv")
```

Aquesta operació carrega el fitxer CSV `diabetes.csv` i genera un objecte `DataFrame` anomenat `df`. A partir d'aquí, ja es poden fer operacions d'anàlisi i manipulació de dades.

Importar fitxers de text

Per llegir fitxers de text, s'utilitza també `read_csv()` especificant el separador amb el paràmetre `sep`.

Els separadors més comuns són:

- Coma: `sep=","`
- Espai en blanc: `sep="\s"`
- Tabulador: `sep="\t"`
- Dos punts: `sep=":"`

```
df = pd.read_csv("diabetes.txt", sep="\s")
```

Importar fitxers Excel (una sola fulla)

Per llegir fitxers Excel (.xls o .xlsx), utilitza la funció `read_excel()` indicant la ruta del fitxer:

```
df = pd.read_excel("diabetes.xlsx")
```

Pots especificar altres arguments opcionals:

- `header`: fila que es farà servir com a encapçalament (per defecte és 0).
- `names`: llista amb noms de columna si el fitxer no en té.
- `index_col`: columna que s'utilitzarà com a índex (per defecte, cap).

Importar fitxers Excel (múltiples fulles)

Per llegir una fulla concreta d'un fitxer Excel amb diverses fulles, fes servir el paràmetre `sheet_name`. Pots passar-hi el nom de la fulla o el seu índex (recorda que Python utilitza **indexació 0-based**).

```
# Carregar la segona fulla (índex 1)
df = pd.read_excel("diabetes_multi.xlsx", sheet_name=1)
```

Importar fitxers JSON

La funció `read_json()` permet llegir fitxers en format JSON directament i convertir-los a `DataFrame`:

```
df = pd.read_json("diabetes.json")
```

Importar dades des d'una base de dades SQL

Per carregar dades d'una base de dades relacional, pots usar `pd.read_sql()` conjuntament

amb una connexió de base de dades. Per exemple, amb SQLite:

```
import sqlite3

# Connexió a una base de dades SQLite
conn = sqlite3.connect("my_database.db")

# Llegir dades d'una taula
df = pd.read_sql("SELECT * FROM my_table", conn)
```

Per a conjunts de dades grans, és recomanable utilitzar **SQLAlchemy** per optimitzar consultes i connexions.

Importar dades des d'una API (format JSON)

Si les dades provenen d'una API web que retorna JSON, pots llegir-les directament amb `pd.read_json()`:

```
url = "https://api.exemple.com/dades.json"
df = pd.read_json(url)
```

Referències

- Documentació oficial de pandas
- pandas.read_csv()
- pandas.read_excel()
- pandas.read_json()
- pandas.read_sql()

Exportar dades amb pandas

Igual que **pandas** pot importar dades de diversos tipus de fitxers, també permet *exportar* els `DataFrame` a diferents formats. Això és especialment útil quan les dades han estat transformades i s'han de desar localment o compartir amb altres eines.

Exportar un DataFrame a un fitxer CSV

Un `DataFrame` (per exemple, `df`) es desa com a fitxer CSV amb el mètode `.to_csv()`. Els paràmetres principals són:

- `filename`: el nom (i camí) del fitxer de sortida.
- `index`: indica si s'ha d'escriure l'índex (`True` per defecte).

```
df.to_csv("diabetes_out.csv", index=False)
```

Exportar un DataFrame a un fitxer JSON

Per desar un `DataFrame` com a fitxer JSON, crida el mètode `.to_json()`:

```
df.to_json("diabetes_out.json")
```

Nota: Un fitxer JSON desa l'objecte tabular com a parelles clau-valor. Això fa que es repeteixin els noms de les columnes per a cada registre.

Exportar un DataFrame a un fitxer de text

Igual que amb els CSV, pots fer servir `.to_csv()` però especificant un separador (`sep`) diferent i una extensió `.txt`.

```
df.to_csv("diabetes_out.txt", header=df.columns, index=None, sep=" ")
```

- `header`: indica si s'escriuen els noms de columna.
- `sep`: defineix el caràcter separador (per exemple, espai o tabulador).

Exportar un DataFrame a un fitxer Excel

Per desar un `DataFrame` en format Excel (`.xls` o `.xlsx`), utilitza el mètode `.to_excel()`:

```
df.to_excel("diabetes_out.xlsx", index=False)
```

També pots definir el nom de la fulla amb el paràmetre `sheet_name`, o exportar múltiples fulles utilitzant un objecte `ExcelWriter`.

```
with pd.ExcelWriter("diabetes_out_multi.xlsx") as writer:
    df1.to_excel(writer, sheet_name="Fulla1", index=False)
    df2.to_excel(writer, sheet_name="Fulla2", index=False)
```

Referències

- Documentació oficial de pandas
- pandas.DataFrame.to_csv()
- pandas.DataFrame.to_json()
- pandas.DataFrame.to_excel()

Visualitzar i entendre DataFrames amb pandas

Després de llegir dades tabulars en un `DataFrame`, el primer pas és donar-hi una ullada. **pandas** ofereix mètodes per veure una mostra de les dades o obtenir un resum estadístic general.

Visualitzar dades amb `.head()` i `.tail()`

Pots veure les primeres o últimes files d'un `DataFrame` amb els mètodes `.head()` i `.tail()`. L'argument n indica quantes files mostrar (per defecte 5).

```
df.head()          # Primeres 5 files
df.tail(n=10)      # Últimes 10 files
```

Entendre les dades amb `.describe()`

El mètode `.describe()` mostra estadístiques resum de les columnes numèriques: nombre de valors, mitjana, desviació estàndard, mínim, màxim i quartils.

```
df.describe()
```

També pots personalitzar els percentils mostrats amb el paràmetre `percentiles`:

```
df.describe(percentiles=[0.3, 0.5, 0.7])
```

O bé, limitar el resum a certs tipus de dades amb `include` o `exclude`:

```
df.describe(include=[int])     # Només columnes enteres
df.describe(exclude=[int])     # Exclou columnes enteres
```

El mètode `.T` (transposar) pot ajudar a visualitzar millor les estadístiques:

```
df.describe().T
```

Entendre les dades amb `.info()`

`.info()` és una manera ràpida de veure tipus de dades, valors nuls i ús de memòria.

```
df.info(show_counts=True, memory_usage=True, verbose=True)
```

- **show_counts=True**: mostra el nombre de valors no nuls per columna.
- **memory_usage=True**: mostra l'ús total de memòria.
- **verbose=True**: imprimeix informació completa.

Entendre la mida del DataFrame amb `.shape`

L'atribut `.shape` retorna una tupla amb el nombre de files i columnes:

```
df.shape        # (files, columnes)
df.shape[0]     # Nombre de files
df.shape[1]     # Nombre de columnes
```

Consultar les columnes del DataFrame

L'atribut `.columns` retorna els noms de les columnes com un objecte `Index`:

```
df.columns
```

Pots convertir-ho fàcilment en una llista amb la funció `list()`:

```
list(df.columns)
```

Comprovar valors nuls amb `.isnull()`

Per comprovar si hi ha valors nuls, es pot fer servir `.isnull()`. Abans, creem una còpia del DataFrame i afegim alguns valors nuls a la columna `Pregnancies`:

```
df2 = df.copy()
df2.loc[2:5, "Pregnancies"] = None
df2.head(7)
```

El mètode `.isnull()` retorna `True` si un element és nul:

```
df2.isnull().head(7)
```

Sovint és més útil saber quants nuls hi ha per columna. Combinem `.isnull()` amb `.sum()`:

```
df2.isnull().sum()
```

Això retorna una sèrie amb el nombre de valors nuls per columna.

Referències

- Documentació oficial de pandas
- pandas.DataFrame.head()
- pandas.DataFrame.describe()
- pandas.DataFrame.info()
- pandas.DataFrame.isnull()

Ordenar, seleccionar i extreure dades amb pandas

El paquet **pandas** ofereix diverses formes de *classificar*, *filtrar* i *extreure* dades dins d'un `DataFrame`. A continuació veurem les més habituals.

Ordenar dades

Per ordenar un `DataFrame` per una columna concreta:

```
df.sort_values(by="Age", ascending=False, inplace=True)  # Ordena per Age (d
```

També es poden ordenar per diverses columnes:

```
df.sort_values(by=["Age", "Glucose"], ascending=[False, True], inplace=True)
```

Reiniciar l'índex

Si filtres o ordenes un `DataFrame`, l'índex pot quedar desalineat. Pots restablir-lo amb `.reset_index()`:

```
df.reset_index(drop=True, inplace=True)  # Reinicia índex i elimina el vell
```

Filtrar dades segons condicions

Per extreure files basades en una condició:

```
df[df["BloodPressure"] > 100]  # Files on BloodPressure és > 100
```

Aïllar columnes

Una sola columna es pot obtenir amb claudàtors `[ ]`:

```
df["Outcome"]
```

El resultat és un objecte `Series` (una columna individual d'un DataFrame).

Aïllar diverses columnes

Pots passar una llista de noms de columnes dins dels claudàtors dobles `[[ ]]`:

```
df[["Pregnancies", "Outcome"]]
```

Aïllar files

Una sola fila

```
df[df.index == 1]   # Retorna la fila amb índex 1
```

Un conjunt de files

```
df[df.index.isin(range(2, 10))]  # Retorna files amb índex del 2 al 9
```

Accedir a files amb `.loc[]` i `.iloc[]`

`.loc[]` selecciona per etiqueta (*label*), mentre que `.iloc[]` selecciona per posició (*integer location*).

Primer, modifiquem l'índex de `df2` per veure la diferència:

```
df2.index = range(1, 769)
```

Exemples bàsics

```
df2.loc[1]     # Fila amb etiqueta 1
df2.iloc[1]    # Fila amb posició 1 (la segona fila)
```

Seleccionar múltiples files

```
df2.loc[100:110]   # Files amb etiquetes de 100 a 110
df2.iloc[100:110]  # Files per posició 100-109
```

Seleccionar files i columnes

```
df2.loc[100:110, ["Pregnancies", "Glucose", "BloodPressure"]]
df2.iloc[100:110, :3]
```

Seleccionar rangs amplis o finals

```
df2.loc[760:, ["Pregnancies", "Glucose", "BloodPressure"]]
df2.iloc[760:, :3]
```

Modificar valors existents

També pots canviar valors directament usant l'operador =:

```
df2.loc[df["Age"] == 81, ["Age"]] = 80
```

Filtrat condicional

pandas permet filtrar files segons condicions sobre valors de columnes.

Exemple: pressió arterial igual a 122

```
df[df.BloodPressure == 122]
```

Exemple: resultat igual a 1

```
df[df.Outcome == 1]
```

Exemple: pressió arterial major que 100

```
df.loc[df["BloodPressure"] > 100, ["Pregnancies", "Glucose", "BloodPressure"
```

Referències

- Documentació oficial de pandas
- pandas.DataFrame.sort_values()
- pandas.DataFrame.loc[]
- pandas.DataFrame.iloc[]
- pandas.DataFrame.reset_index()

Neteja de dades amb pandas

La **neteja de dades** és una de les tasques més habituals en ciència de dades. Amb **pandas** pots pre-processar dades per a qualsevol ús, com ara l'entrenament de models de *machine learning* o d'aprenentatge profund.

En aquest exemple farem servir el `DataFrame` `df2` amb quatre valors nuls per il·lustrar diferents tècniques. Recorda que pots comprovar quants valors nuls hi ha amb:

```
df2.isnull().sum()
```

Tècnica #1: eliminar valors nuls

Una manera de tractar dades mancants és simplement **eliminar-les**. Això és útil quan hi ha moltes dades i perdre unes poques files no afecta l'anàlisi.

```
df3 = df2.copy()
df3 = df3.dropna()
df3.shape
```

L'argument `axis` permet especificar si vols eliminar files (`axis=0`, per defecte) o columnes (`axis=1`). També pots utilitzar `inplace=True` per modificar el `DataFrame` directament.

```
df3 = df2.copy()
df3.dropna(inplace=True, axis=1)
df3.head()
```

Pots eliminar tant files com columnes que tinguin valors nuls amb `how='all'`:

```
df3 = df2.copy()
df3.dropna(inplace=True, how="all")
```

Tècnica #2: substituir valors nuls

En lloc d'eliminar-los, pots **substituir els valors nuls** per un valor estadístic (mitjana, mediana, moda) o un valor fix.

```
df3 = df2.copy()

# Calcular la mitjana de Pregnancies
mean_value = df3["Pregnancies"].mean()

# Substituir valors nuls per la mitjana
df3 = df3.fillna(mean_value)
```

Aquesta estratègia és útil quan vols mantenir totes les files i reduir la pèrdua d'informació.

Eliminar duplicats

Vegem com eliminar files duplicades. Primer, afegim duplicats concatenant el `DataFrame` `df2` amb si mateix:

```
df3 = pd.concat([df2, df2])
df3.shape
```

Per eliminar les files duplicades, utilitza el mètode `.drop_duplicates()`:

```
df3 = df3.drop_duplicates()
df3.shape
```

Canviar noms de columnes

És habitual haver de canviar els noms de les columnes per fer-los més clars o coherents.

```
df3.rename(columns={"DiabetesPedigreeFunction": "DPF"}, inplace=True)
df3.head()
```

També pots assignar **directament una llista de noms** a totes les columnes:

```
df3.columns = [
    "Glucose", "BloodPressure", "SkinThickness",
    "Insulin", "BMI", "DPF", "Age", "Outcome", "STF"
]
df3.head()
```

Checklist de neteja de dades

Algunes tasques comunes de neteja que pots fer amb pandas:

- Comprovar valors nuls: `df.isnull().sum()`
- Eliminar valors nuls: `df.dropna()`
- Substituir valors nuls: `df.fillna()`
- Eliminar duplicats: `df.drop_duplicates()`
- Renombrar columnes: `df.rename()` o `df.columns = []`
- Canviar tipus de dades: `df.astype()`
- Detectar valors fora de rang amb condicions (ex: `df[df["col"] > límit]`)

Referències

- Documentació oficial de pandas
- pandas.DataFrame.dropna()
- pandas.DataFrame.fillna()
- pandas.DataFrame.drop_duplicates()
- pandas.DataFrame.rename()