Modulo 1

Introducción a la programación en Python

Centro de e-Learning
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Unidad 3: Procesamiento de datos en Python

Centro de e-Learning
UTN.BA

Organización del primer módulo

- **Clase 1:** Introducción al curso + Que es Python + Elección del editor de código + Definición de programa e instrucciones + Tipos de datos básicos

- **Clase 2:** Listas + Diccionarios + Tuplas + Estructuras de control condicionales y cíclicas + Funciones + Librería: NumPy

- **Clase 3:** Dataframes + Librería: Pandas + Concatenación y Merge de dataframes + Filtros + Limpieza de datos + Tipos de datos: fechas

- **Clase 4:** Visualización de datos + Principios generales del diseño analítico + Librería: Matplotlib + Gráficos mas populares

DataFrames

Los **DataFrames** son estructuras de datos pensadas para el almacenamiento de datasets (bases de datos).

- Contiene instancias (u observaciones) y atributos (o variables, pero no las de programacion) asociados con cada una de ellas.
- Similar a una matriz, pero con distintos tipos de datos en cada columna.

	nombre (chr)	edad (num)	ciudad (chr)
1	Mario	25	CBA
2	Rosa	26	CABA
3	Dana	23	CABA

Pandas

Pandas es una librería de Python ampliamente utilizada para el análisis y manipulación de datos. Se basa en las estructuras de datos de **NumPy**. Algunos puntos clave a considerar:

- **DataFrame**: es la estructura de datos principal en pandas y como dijimos anteriormente, sirven para almacenar y manipular datos en forma de tablas bidimensionales.
- **Series**: es una estructura de datos unidimensional que puede considerarse como una columna de un **DataFrame**.

```python
#cargamos la libreria y la renombramos a "pd" para usarla
import pandas as pd
lista_id = [1, 2, 3, 4, 5]
lista_texto = ['a', 'b', 'c', 'd', 'e']

#Transformamos ambas listas en un dataframe
df = pd.DataFrame({'id': lista_id, 'texto': lista_texto})
df
```

	id	texto
0	1	a
1	2	b
2	3	c
3	4	d
4	5	e

Pandas

Pandas al igual que el resto de las librerías, cuenta con **funciones** y **métodos** muy útiles, los principales pueden ser:

- **Creación de estructuras de datos:**
  pd.Series(data, index)
  pd.DataFrame(data, index, columns)
- **Lectura y escritura de datos:**
  pd.read_csv() ; pd.read_excel() ;pd.read_sql(),
  DataFrame.to_csv() ; DataFrame.to_Excel() ; DataFrame.to_sql()
- **Selección y filtrado de datos:**
  DataFrame[“columna”]
  DataFrame.loc[] ; DataFrame.iloc[]

Pandas

- **Operaciones en columnas:**

  DataFrame[“nueva_columna],

  DataFrame.drop(“columna”),

  DataFrame.rename(columns={“viejo_nombre”:”nuevo_nombre”})

- **Manejo de datos faltantes:**

  DataFrame.dropna()

  DataFrame.fillna(valor)

- **Operaciones estadísticas y de agregación:**

  DataFrame.describe()

  DataFrame.groupby(“columna”).mean()

Concatenacion (concat) de DataFrames

Esta función se utiliza para **concatenar** DataFrames a lo largo de un eje particular (por lo general, filas o columnas). Se pueden concatenar DataFrames **verticalmente** (a lo largo de las filas) y horizontalmente (a lo largo de las columnas).

In [4]:
```
# Creamos dos DataFrames de ejemplo
df1 = pd.DataFrame({'A': ['A0', 'A1'], 'B': ['B0', 'B1']})
df2 = pd.DataFrame({'A': ['A3', 'A4'], 'B': ['B3', 'B4']})

# Concatenamos verticalmente (a lo largo de las filas)
result_vertical = pd.concat([df1, df2])

print("Concatenación vertical:")
print(result_vertical)
```

```
Concatenación vertical:
    A   B
0  A0  B0
1  A1  B1
0  A3  B3
1  A4  B4
```

Concatenacion (concat) de DataFrames

Esta función se utiliza para **concatenar** DataFrames a lo largo de un eje particular (por lo general, filas o columnas). Se pueden concatenar DataFrames verticalmente (a lo largo de las filas) y **horizontalmente** (a lo largo de las columnas).

In [5]:
```
# Creamos dos DataFrames de ejemplo
df1 = pd.DataFrame({'A': ['A0', 'A1'], 'B': ['B0', 'B1']})
df2 = pd.DataFrame({'A': ['A3', 'A4'], 'B': ['B3', 'B4']})

# Concatenar horizontalmente (a lo largo de las columnas)
result_horizontal = pd.concat([df1, df2], axis=1)

print("\nConcatenación horizontal:")
print(result_horizontal)
```

```
Concatenación horizontal:
    A   B   A   B
0  A0  B0  A3  B3
1  A1  B1  A4  B4
```

Combinación (merge) de DataFrames

Esta función se utiliza para **combinar** DataFrames basándose en columnas clave o índices. Es similar a la operación de **join** en SQL. Se pueden especificar columnas clave en ambos DataFrames y decidir como realizar la combinación (**interna**, externa, izquierda, derecha).

```
In [6]: # Creamos dos DataFrames de ejemplo
        df_left = pd.DataFrame({'key': ['K0', 'K1'], 'value_left': ['L0', 'L1']})
        df_right = pd.DataFrame({'key': ['K1', 'K2'], 'value_right': ['R1', 'R2']})

        # Realizamos una combinación basada en la columna 'key'
        result_inner = pd.merge(df_left, df_right, on='key', how='inner')

        print("Combinación interna:")
        print(result_inner)

        Combinación interna:
          key value_left value_right
        0  K1         L1          R1
```

Combinación (merge) de DataFrames

Esta función se utiliza para **combinar** DataFrames basándose en columnas clave o índices. Es similar a la operación de join en SQL. Se pueden especificar columnas clave en ambos DataFrames y decidir como realizar la combinación (interna, **externa**, izquierda, derecha).

```
In [7]: # Creamos dos DataFrames de ejemplo
        df_left = pd.DataFrame({'key': ['K0', 'K1'], 'value_left': ['L0', 'L1']})
        df_right = pd.DataFrame({'key': ['K1', 'K2'], 'value_right': ['R1', 'R2']})

        # Realizamos una combinación basada en la columna 'key'
        result_outer = pd.merge(df_left, df_right, on='key', how='outer')

        print("\nCombinación externa:")
        print(result_outer)

        Combinación externa:
          key value_left value_right
        0  K0         L0         NaN
        1  K1         L1          R1
        2  K2        NaN          R2
```

Combinación (merge) de DataFrames

Esta función se utiliza para **combinar** DataFrames basándose en columnas clave o índices. Es similar a la operación de join en SQL. Se pueden especificar columnas clave en ambos DataFrames y decidir como realizar la combinación (interna, externa, **izquierda**, derecha).

```
In [8]: # Creamos dos DataFrames de ejemplo
        df_left = pd.DataFrame({'key': ['K0', 'K1'], 'value_left': ['L0', 'L1']})
        df_right = pd.DataFrame({'key': ['K1', 'K2'], 'value_right': ['R1', 'R2']})

        # Realizamos una combinación basada en la columna 'key'
        result_left = pd.merge(df_left, df_right, on='key', how='left')

        print("\nCombinación izquierda:")
        print(result_left)

        Combinación izquierda:
          key value_left value_right
        0  K0         L0         NaN
        1  K1         L1          R1
```

Combinación (merge) de DataFrames

Esta función se utiliza para **combinar** DataFrames basándose en columnas clave o índices. Es similar a la operación de join en SQL. Se pueden especificar columnas clave en ambos DataFrames y decidir como realizar la combinación (interna, externa, izquierda, **derecha**).

```
In [9]: # Creamos dos DataFrames de ejemplo
        df_left = pd.DataFrame({'key': ['K0', 'K1'], 'value_left': ['L0', 'L1']})
        df_right = pd.DataFrame({'key': ['K1', 'K2'], 'value_right': ['R1', 'R2']})

        # Realizamos una combinación basada en la columna 'key'
        result_right = pd.merge(df_left, df_right, on='key', how='right')

        print("\nCombinación derecha:")
        print(result_right)

        Combinación derecha:
          key value_left value_right
        0  K1         L1          R1
        1  K2        NaN          R2
```

Filtros

En Pandas, los filtros permiten seleccionar y extraer subconjuntos específicos de datos que cumplen ciertos criterios. Hay varias formas de realizar filtros en pandas:

- Filtros por condición

```
In [11]: data = {'Nombre': ['Alice', 'Bob', 'Charlie', 'David'],
                 'Edad': [25, 30, 22, 35],
                 'Puntuación': [90, 85, 88, 95]}
         df = pd.DataFrame(data)    # Creamos un DataFrame de ejemplo
         # Filtramos las filas donde la edad es mayor que 25
         filtro = df['Edad'] > 25
         result = df[filtro]
         print(result)

           Nombre  Edad  Puntuación
         1    Bob    30          85
         3  David    35          95
```

Filtros

En Pandas, los filtros permiten seleccionar y extraer subconjuntos específicos de datos que cumplen ciertos criterios. Hay varias formas de realizar filtros en pandas:

- Filtros por Múltiples condiciones:

```
In [12]: data = {'Nombre': ['Alice', 'Bob', 'Charlie', 'David'],
                 'Edad': [25, 30, 22, 35],
                 'Puntuación': [90, 85, 88, 95]}
         df = pd.DataFrame(data)    # Creamos un DataFrame de ejemplo
         # Filtramos las filas donde la edad es mayor que 25 y puntuación mayor que 90
         filtro_multiple = (df['Edad'] > 25) & (df['Puntuación'] > 90)
         result_multiple = df[filtro_multiple]
         print(result_multiple)

           Nombre  Edad  Puntuación
         3  David    35          95
```

Filtros

En Pandas, los filtros permiten seleccionar y extraer subconjuntos específicos de datos que cumplen ciertos criterios. Hay varias formas de realizar filtros en pandas:

- Filtros por Valores de Columna:

```
In [13]: data = {'Nombre': ['Alice', 'Bob', 'Charlie', 'David'],
                 'Edad': [25, 30, 22, 35],
                 'Puntuación': [90, 85, 88, 95]}
         df = pd.DataFrame(data)    # Creamos un DataFrame de ejemplo
         # Filtramos las filas donde el nombre sea 'Charlie' o 'David'
         nombres_a_filtrar = ['Charlie', 'David']
         filtro_nombres = df['Nombre'].isin(nombres_a_filtrar)
         result_nombres = df[filtro_nombres]
         print(result_nombres)

             Nombre  Edad  Puntuación
         2  Charlie    22          88
         3    David    35          95
```

Filtros

En Pandas, los filtros permiten seleccionar y extraer subconjuntos específicos de datos que cumplen ciertos criterios. Hay varias formas de realizar filtros en pandas:

- Filtros por índice:

```
In [14]: data = {'Nombre': ['Alice', 'Bob', 'Charlie', 'David'],
                 'Edad': [25, 30, 22, 35],
                 'Puntuación': [90, 85, 88, 95]}
         df = pd.DataFrame(data)    # Creamos un DataFrame de ejemplo
         # Filtramos las filas por índice
         indices_a_filtrar = [0, 2]
         result_indices = df.loc[indices_a_filtrar]
         print(result_indices)

             Nombre  Edad  Puntuación
         0    Alice    25          90
         2  Charlie    22          88
```

Modificación de datos en un DataFrame

Al igual que los filtros, también tenemos varias alternativas para modificar los datos de un DataFrame. Por ejemplo:

- Modificar un valor especifico:

```
In [15]: data = {'Nombre': ['Alice', 'Bob', 'Charlie', 'David'],
                 'Edad': [25, 30, 22, 35],
                 'Puntuación': [90, 85, 88, 95]}
         df = pd.DataFrame(data)    # Creamos un DataFrame de ejemplo

         # Modificamos la puntuación de Bob
         df.at[1, 'Puntuación'] = 92
         print(df)

          Nombre  Edad  Puntuación
         0  Alice    25          90
         1    Bob    30          92
         2  Charlie    22          88
         3  David    35          95
```

Modificación de datos en un DataFrame

Al igual que los filtros, también tenemos varias alternativas para modificar los datos de un DataFrame. Por ejemplo:

- Modificar una columna completa:

```
In [16]: data = {'Nombre': ['Alice', 'Bob', 'Charlie', 'David'],
                 'Edad': [25, 30, 22, 35],
                 'Puntuación': [90, 85, 88, 95]}
         df = pd.DataFrame(data)    # Creamos un DataFrame de ejemplo

         df['Edad'] = df['Edad'] + 1
         print(df)

             Nombre  Edad  Puntuación
         0    Alice    26          90
         1      Bob    31          85
         2  Charlie    23          88
         3    David    36          95
```

Modificación de datos en un DataFrame

Al igual que los filtros, también tenemos varias alternativas para modificar los datos de un DataFrame. Por ejemplo:

- Condiciones y Modificaciones:

```
In [17]: data = {'Nombre': ['Alice', 'Bob', 'Charlie', 'David'],
                 'Edad': [25, 30, 22, 35],
                 'Puntuación': [90, 85, 88, 95]}
         df = pd.DataFrame(data)    # Creamos un DataFrame de ejemplo

         # Incrementar la puntuación en 10 para aquellos con edad mayor a 25
         df.loc[df['Edad'] > 25, 'Puntuación'] += 10
         print(df)

             Nombre  Edad  Puntuación
         0    Alice    25          90
         1      Bob    30          95
         2  Charlie    22          88
         3    David    35         105
```

Manejo de valores faltantes

¿Por qué pueden faltar valores en una base?

- **MCAR** (**missing completely at random**)

P(missing) para todas las instancias es la misma y no depende de las medidas de otras variables. Ej: se perdio la respuesta para una encuesta.

- **MAR** (**missing at random**)

P(missing) depende de la informacion observada. Ej: missing en nota del recuperatorio (depende de valor observado para nota del primer parcial).

- **MNAR** (**missing not at random**)

P(missing) esta relacionada con los valores perdidos. Ej: ¿Cuánto gana? (si es muy alto quizás no responden)

Manejo de valores faltantes

Entonces, ¿Qué podemos hacer si faltan datos en nuestra base de datos?

"Obviously, the best way to treat missing data is not to have them". Si no, hay distintas alternativas y su aplicación dependera del contexto:

1. **Completar** con las medidas de tendencia central:
- **Media**: generalmente la mas utilizada por su simplicidad.
- **Mediana**: buena opcion para algunas variables de analisis, por ejemplo: ingresos de una persona/familia.
- **Moda**: si existe un valor en concreto altamente representativo que se repita.

2. **Interpolar**
3. **Ignorar las filas** con missing values (si nuestro dataset tiene muchos valores faltantes en distintas columnas, esta alternativa puede hacer que se pierda mucha informacion).
4. Convertirlos en un valor de la variable categórica

Tipos de datos: Fechas

En pandas, las **fechas** y **tiempos** se manejan a través del tipo de dato **datetime64**. Este tipo de dato proporciona una funcionalidad robusta para trabajar con fechas y horas. Algunos de los métodos mas útiles son:

- **Creación de fechas:** podemos hacerlo con pd.Timestamp o pd.to_datetime()
- **Creación de rangos de fechas:** pandas proporciona la función pd.date_range()
- **Acceso a componentes de fecha**: podemos obtener el año, mes día, etc.
  fecha.year
  fecha.month
  fecha.day

Tipos de datos: Fechas - Ejemplo

In [19]:
```
fecha = pd.Timestamp('2022-01-20')  #Creamos un objeto de fecha
print("la fecha es:", fecha)

fecha_str = '2022-01-20'
fecha_convertida = pd.to_datetime(fecha_str) #Convertimos una cadena a una fecha
print("La fecha convertida es:", fecha_convertida)
```

```
la fecha es: 2022-01-20 00:00:00
La fecha convertida es: 2022-01-20 00:00:00
```

In [20]:
```
# Accededemos a las componentes de fecha
print("El año de fecha es:", fecha.year)
print("El mes de fecha es:", fecha.month)
print("El día de fecha es:", fecha.day)
```

```
El año de fecha es: 2022
El mes de fecha es: 1
El día de fecha es: 20
```

In [21]:
```
# Sumamos 3 días a la fecha
nueva_fecha = fecha + pd.Timedelta(days=3)
print("La nueva fecha con 3 dias extra es:", nueva_fecha)
```

```
La nueva fecha con 3 dias extra es: 2022-01-23 00:00:00
```