«Talento Tech»

Data Analytics

con Python

Clase 08

Buenos Aires Ciudad

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Clase N° 08 | Integración de datos

Temario:

- Combinación de diferentes fuentes de datos en Python, Numpy y Pandas
- `merge()`
- `join()`

Objetivos de la clase:

- Repasar métodos de conjuntos.
- Aprender a combinar:
    - vectores y matrices en Numpy.
    - DataFrames en Pandas

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Integración de Datos en Python, NumPy y Pandas

En esta clase abordaremos cómo combinar datos de diferentes fuentes utilizando Python, junto a las bibliotecas NumPy y Pandas.

¿Qué es la integración de datos?

La integración de datos se refiere al **proceso de combinar datos provenientes de distintas fuentes para obtener una visión unificada y coherente**. Esto es particularmente relevante en proyectos de análisis donde los datos pueden estar dispersos en diversas bases de datos, archivos CSV, API, entre otros formatos. La capacidad de integrar datos correctamente garantiza que se cuente con la información estructurada de una manera más adecuada para facilitar los procesos posteriores.

Python: Operaciones sobre conjuntos y listas en Python (set operations)

Uno de los principales usos del tipo `set` es utilizarlo en operaciones del álgebra de conjuntos, como unión, intersección, diferencia y diferencia simétrica. Repasemos algunas de estas operaciones en Python.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Unión de conjuntos

La unión de dos conjuntos A y B es el conjunto `A ∪ B` que contiene todos los elementos de A y de B.

En Python se utiliza el operador `|` para realizar la unión de dos o más conjuntos.

```
a = {1, 2, 3, 4}
b = {2, 4, 6, 8}
a | b
# {1, 2, 3, 4, 6, 8}
```

Intersección de conjuntos

La intersección de dos conjuntos A y B es el conjunto `A ∩ B` que contiene todos los elementos comunes de A y B.
En Python se utiliza el operador `&` para realizar la intersección de dos o más conjuntos.

```
a = {1, 2, 3, 4}
b = {2, 4, 6, 8}
a & b
# {2, 4}
```

Diferencia de conjuntos en Python

La diferencia entre dos conjuntos A y B es el conjunto `A \ B` que contiene todos los elementos de A que no pertenecen a B.
En Python se utiliza el operador `-` para realizar la diferencia de dos o más conjuntos.

```
a = {1, 2, 3, 4}
b = {2, 4, 6, 8}
a - b
# {1, 3}
```

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Diferencia simétrica de conjuntos en Python

La diferencia simétrica entre dos conjuntos A y B es el conjunto que contiene los elementos de A y B que no son comunes.
En Python se utiliza el operador `^` para realizar la diferencia simétrica de dos o más conjuntos.

```
a = {1, 2, 3, 4}
b = {2, 4, 6, 8}
a ^ b
# {1, 3, 6, 8}
```

La función np.concatenate()

En el caso de que necesitemos concatenar dos o más vectores o matrices de NumPy se puede recurrir a la función `np.concatenate()`. Esta función que tiene la siguiente sintaxis básica:

```
np.concatenate((a1, a2, ...), axis=0)
```

- `a1, a2, ...`: son vectores o matrices de NumPy. Cada uno de los elementos de esta serie debe tener las mismas dimensiones.
- `axis`: eje a lo largo del cual se desea que se concatenan los vectores o matrices. Los posibles valores son
- `0`: las matrices se unen por filas (valor por defecto)
- `1`: las matrices se unen por columnas.

- `None`: se obtiene un vector.
- `out`: permite guardar el resultado en un objeto existente (debe respetar el tamaño de la estructura).

Cómo concatenar vectores en NumPy

El uso más básico que nos permite la función `np.concatenate()` es el de **concatenar vectores**. Para hacerlo, creamos una tupla con los vectores y los pasamos como parámetro de la función. Esto nos devuelve un objeto `np.array` con los valores de todos los vectores.

```
import numpy as np
arr = np.concatenate(([1, 2, 3], [4, 5, 6]))
arr
array([1, 2, 3, 4, 5, 6])
```

Podemos concatenar más de un array de diferente longitud.

```
np.concatenate(([1, 2, 3], [4, 5, 6], [7, 8]))
array([1, 2, 3, 4, 5, 6, 7, 8])
```

Cómo concatenar matrices en NumPy

El uso más interesante de la función `np.concatenate()` es el de **concatenar matrices**. El método es el mismo. A menos que indiquemos lo contrario, la concatenación será por filas.

```
mat_1 = np.array([[1, 2, 3], [4, 5, 6]])
mat_2 = np.array([[1, 1, 1], [2, 2, 2]])
np.concatenate((mat_1, mat_2))
array([[1, 2, 3],
[4, 5, 6],
[1, 1, 1],
[2, 2, 2]])
```

Si deseamos una concatenación por columnas, deberemos asignar el valor 1 a la propiedad axis.

```
np.concatenate((mat_1, mat_2), axis=1)
array([[1, 2, 3, 1, 1, 1],
[4, 5, 6, 2, 2, 2]])
```

Si necesitamos obtener un vector a partir de las matrices, indicaremos el valor None en la propiedad axis. Opción que se muestra a continuación.

```
np.concatenate((mat_1, mat_2), axis=None)
array([1, 2, 3, 4, 5, 6, 1, 1, 1, 2, 2, 2])
```

Guardar los resultados en un objeto existente

Si necesitamos **almacenar el resultado en un objeto ya existente**, utilizaremos la propiedad out. Debemos tener en cuenta que el tamaño del objeto debe ser igual al del resultado esperado de la función, ya que en caso contrario se produciría un error. Veamos un ejemplo donde almacenamos el resultado de una concatenación en una matriz de ceros.

```
result = np.zeros((4,3))
np.concatenate((mat_1, mat_2), out=result)
result
array([[1., 2., 3.],
[4., 5., 6.],
[1., 1., 1.],
[2., 2., 2.]])
```

Concatenar matrices con vectores en NumPy

Cómo **NO concatenar** matrices con vectores en NumPy
Posiblemente, después de aprender como se tiene que utilizar la función `concatenate()` para unir dos matrices, lo primero que se puede intentar para combinar una matriz con un vector sea algo como lo siguiente

```
vec = np.array([10, 10, 10])
mat = np.array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
np.concatenate((mat, vec))

ValueError: all the input arrays must have same number of
dimensions, but the array at index 0 has 2 dimension(s) and
the array at index 1 has 1 dimension(s)
```

Esperaríamos obtener una nueva matriz de 4 por 3 con el vector en la última fila, pero obtenemos un error. Esto ocurrió porque las dimensiones de los dos objetos no coinciden, uno es una matriz de 3 por 3 y el otro es un vector.

Método para concatenar matrices con vectores en NumPy

Para solucionar el problema solamente deberemos convertir el vector en una matriz.

```
vec = np.array([10, 20, 30])
mat = np.array([[1, 2, 3],
[4, 5, 6],
[7, 8, 9]])
np.concatenate((mat, [vec]))
array([[ 1, 2, 3],
[ 4, 5, 6],
[ 7, 8, 9],
[10, 20, 30]])
```

En el caso de que necesitemos concatenar el vector como una columna, además de indicar a la función que se concatene por columnas (con el parámetro axis) hay que convertir el vector en una matriz columna mediante la transposición (T). Veamos un ejemplo:

```
np.concatenate((mat, np.array([vec]).T), axis=1)
array([[ 1, 2, 3, 10],
[ 4, 5, 6, 20],
[ 7, 8, 9, 30]])
```

Así obtenemos una matriz final de 3 por 4 donde el vector se insertó como una columna al final.

También es posible utilizar otros métodos para combinar datos en numpy. Te invitamos a explorar los métodos `hstack` y `vstack`.

Combinación de Diferentes Fuentes de Datos en Pandas

Cómo concatenar dos DataFrames

Verticalmente

En Pandas, dos DataFrames se pueden concatenar usando el método `concat()`. Supongamos que tenemos dos DataFrames:

```
import pandas as pd
# Creando DataFrame 1
df1 = pd.DataFrame({
'Nombre': ['John', 'Jack', 'Steve', 'Sarah'],
'Edad': [24, 32, 19, 29],
'Género': ['M', 'M', 'M', 'F']
})
```

```
   Nombre  Edad Género
0   John    24      M
1   Jack    32      M
2  Steve    19      M
3  Sarah    29      F
```

```
# Creando DataFrame 2
df2 = pd.DataFrame({
'Departamento': ['Marketing', 'Ventas', 'Recursos Humanos'],
'Empleados': [15, 12, 10],
})
```

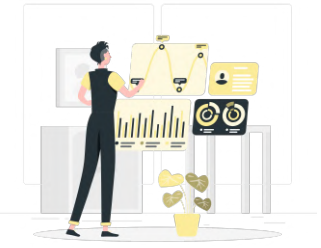
```
       Departamento  Empleados
0         Marketing         15
1            Ventas         12
2  Recursos Humanos         10
```

Podemos usar el método `concat()` para combinar los dos DataFrames verticalmente:

```
# Concatenando verticalmente
df3 = pd.concat([df1, df2], axis=0)
```

El parámetro `axis=0` indica que queremos concatenar los DataFrames apilándolos uno encima del otro (es decir, verticalmente). Después de la concatenación, obtenemos la siguiente salida:

```
  Nombre  Edad Género      Departamento  Empleados
0   John  24.0      M               NaN        NaN
1   Jack  32.0      M               NaN        NaN
2  Steve  19.0      M               NaN        NaN
3  Sarah  29.0      F               NaN        NaN
0    NaN   NaN    NaN         Marketing       15.0
1    NaN   NaN    NaN            Ventas       12.0
2    NaN   NaN    NaN  Recursos Humanos
10.0
```



Podemos ver que los dos DataFrames se concatenan como se desea, pero hay algunos valores `NaN` (nulos) en el nuevo DataFrame donde los nombres de las columnas no coinciden. Podemos eliminar esas filas si no se ajustan a nuestro análisis de datos, o podemos proporcionar algunos valores predeterminados para llenar los valores nulos.

Horizontalmente

También podemos concatenar dos DataFrames horizontalmente cambiando el axis en el método `concat()`:

```
# Concatenando horizontalmente
df4 = pd.concat([df1, df2], axis=1)
```

Ahora, obtenemos la siguiente salida:

```
  Nombre  Edad Género      Departamento  Empleados
0   John    24      M         Marketing       15.0
1   Jack    32      M            Ventas       12.0
2  Steve    19      M  Recursos Humanos       10.0
3  Sarah    29      F               NaN        NaN
```

Concatenar DataFrames con Columnas Diferentes

Si los dos DataFrames que se van a concatenar tienen columnas diferentes, Pandas identifica los nombres de las columnas que no coinciden y los agrega al nuevo DataFrame como columnas separadas. Veámoslo con un ejemplo:

```
# Crear DataFrame
df5 = pd.DataFrame({
'Nombre': ['John', 'Jack', 'Steve', 'Sarah'],
'Edad’: [24, 32, 19, 29],
'Cargo': ['Director', 'Asistente', 'Agente', 'Ejecutivo']
})
```

```
  Nombre  Edad      Cargo
0   John    24   Director
1   Jack    32  Asistente
2  Steve    19     Agente
3  Sarah    29  Ejecutivo
```

Ahora podemos concatenar df1 y df5:

```
# Concatenar df1 y df5
df6 = pd.concat([df1, df5], axis=1)
```

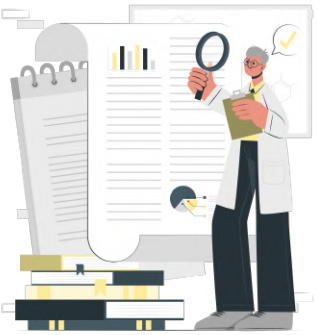
Buenos Aires aprende
Agencia de Habilidades para el Futuro

Después de la concatenación, obtenemos la siguiente salida:

```
  Nombre  Edad Género Nombre  Edad      Cargo
0   John    24      M   John    24   Director
1   Jack    32      M   Jack    32  Asistente
2  Steve    19      M  Steve    19     Agente
3  Sarah    29      F  Sarah    29  Ejecutivo
```

Como podemos ver, Pandas concatena los dos DataFrames agregando las columnas que no coinciden como columnas separadas en el nuevo DataFrame.

Ignorando el índice al concatenar DataFrames horizontalmente



Cuando se concatenan dos DataFrames horizontalmente, el DataFrame resultante conserva los índices originales de los dos DataFrames. Esto puede causar problemas mientras se trabaja con este nuevo DataFrame. Por lo tanto, puede ser necesario en algunos casos ignorar el índice al concatenar horizontalmente. Podemos lograr esto configurando el parámetro `ignore_index` en `True` mientras concatenamos:

```
# Concatenar df1 y df2, ignorando el index
df7 = pd.concat([df1, df2], axis=1, ignore_index=True)
```

Después de la concatenación, obtenemos la siguiente salida:

```
       0   1  2                3     4
0   John  24  M        Marketing  15.0
1   Jack  32  M           Ventas  12.0
2  Steve  19  M  Recursos Humanos  10.0
3  Sarah  29  F              NaN   NaN
```

Ahora el nuevo DataFrame tiene un nuevo índice.

Buenos Aires aprende
Agencia de Habilidades para el Futuro

Métodos Principales para Integrar Datos en Pandas

`merge()`

La **función `merge()` de Pandas** permite combinar dos DataFrames basándose en una o más claves. Este método es similar a las operaciones de "join" que se realizan en bases de datos SQL. La flexibilidad de `merge()` permite realizar diferentes tipos de combinaciones como:

- **Inner Join:** Sólo se conservan las filas que tienen correspondencia en ambos DataFrames.
- **Outer Join:** Se conservan todas las filas de ambos DataFrames, llenando con `NaN` donde no haya coincidencias.
- **Left Join:** Se conservan todas las filas del DataFrame de la izquierda y sólo las filas coincidentes del de la derecha.
- **Right Join:** Se conservan todas las filas del DataFrame de la derecha y las coincidentes del de la izquierda.

Aquí un ejemplo práctico:

```
df1 = pd.DataFrame({'key': ['A', 'B', 'C'], 'value1': [1, 2,
3]})
df2 = pd.DataFrame({'key': ['B', 'C', 'D'], 'value2': [4, 5,
6]})

#df1
  key  value1
0   A       1
1   B       2
2   C       3

#df2
   key  value2
0   B       4
1   C       5
2   D       6

result = pd.merge(df1, df2, on='key', how='inner')
print(result)
```

Este código combinará df1 y df2 solo donde las claves coincidan, produciendo el siguiente resultado:

```
  key  value1  value2
0   B       2       4
1   C       3       5
```

`join()`

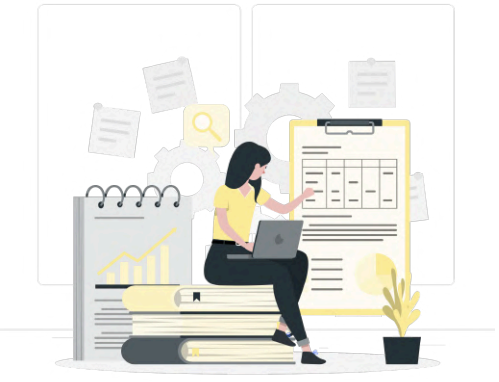
La **función** `join()` es otra forma de combinar DataFrames, especialmente útil cuando tenemos un DataFrame y queremos combinarlo con otro usando sus índices. Esta función proporciona una forma más sencilla de realizar un left join de manera predeterminada.
Por ejemplo, usando los mismos DataFrames, podemos establecer uno de ellos como índice y luego unir:

```
df1.set_index('key', inplace=True)
df2.set_index('key', inplace=True)
result = df1.join(df2, how='inner')
print(result)
```

El resultado será similar al obtenido con `merge()`, pero utilizando índices para realizar la combinación.

Reflexión final

La **integración de datos en Python, NumPy y Pandas** es una habilidad esencial en el análisis de datos. Mediante el uso de funciones como `concat()`, `merge()` y `join()`, podremos combinar datos de diversas fuentes y prepararlos para un análisis más profundo. La versatilidad de estas herramientas permite abordar una variedad de escenarios en los que se requiera manipular y analizar grandes volúmenes de información.



Buenos Aires aprende
Agencia de Habilidades para el Futuro

Materiales y recursos adicionales

- Documentación oficial de Python: Set
- Documentación oficial de Numpy: numpy.concatenate
- Documentación oficial de Pandas:
  - pandas.concat
  - pandas.merge
  - pandas.DataFrame.join

Próximos pasos

- Repaso y profundización: Estadística descriptiva.
  - Medidas de tendencia central.
  - Medidas de dispersión.

Ejercicios prácticos:

SYNTHDATA
SINTETIZA CONOCIMIENTO

Actividad 1: Unificar de Datos de Ventas e Inversiones en Marketing

Contexto

En SynthData, el proyecto de esta semana durante tu pasantía es ayudar en el análisis de datos de ventas de dos sucursales de una empresa. Silvia, tu mentor y Project Manager, ha recopilado datos sobre las ventas de productos de 2 locales de una empresa cliente.

Ambos DataFrames necesitan ser combinados para crear un único conjunto que facilite la visualización del rendimiento de ventas en la empresa.

Objetivos

El objetivo de esta actividad es practicar la combinación de DataFrames. Al final de esta tarea, deberías poder ver cómo se alinean los datos de ambas sucursales y cómo se podrían manejar datos faltantes.

Ejercicio práctico

- Obtener los dos DataFrames que contienen los datos de ventas.
- Concatenar los DataFrames (decidir si correponde hacerlo horizontal o verticalmente), verificar y describir el resultado. Sugerir cómo manejar los datos faltantes, si los hubiera.
- Asegurarse de realizar las operaciones previas necesarias para obtener la siguiente estructura de columnas: `Sucursal, Producto, Ventas, Mes`

Sets de datos

ventas (Google Sheets)

- Norte: columnas Sucursal, Producto, Ingresos, Mes.
- Sur: columnas Sede, Producto, Ventas, Mes.

¿Por qué importa esto en SynthData?

La integración de datos es fundamental en el análisis de datos económicos. Combinar diferentes fuentes de datos permitirá a SynthData obtener una visión completa del rendimiento de productos y mejorar las estrategias comerciales basadas en análisis exhaustivos. Así, tu trabajo se convierte en una pieza clave para ayudar a la toma de decisiones informadas.

Actividad 2: Análisis de Inversiones en Gestión de Personal

Contexto

Matías, el Data Analyst en SynthData, te ha asignado la tarea de combinar datos de empleados de dos fuentes diferentes. Estás trabajando en una base de datos que almacena los datos personales de los empleados de la empresa, y otra que contiene su situación laboral.

Esta nueva estructura de datos es fundamental para entender la situación actual de contratación y mejorar la gestión de recursos humanos.

Objetivos

El objetivo de esta actividad es familiarizarte con el uso de pd.merge() para realizar combinaciones de datos basadas en claves. Al final de esta tarea, deberías entender cómo unir diferentes DataFrames que tienen columnas comunes.

Ejercicio práctico

- Obtener los DataFrames.
- Utilizá `pd.merge()` para combinar ambos DataFrames basados en la columna que se considere más apropiada, con el join que corresponda. Dejar documentada la justificación de ambas elecciones.

Sets de datos

- data_empleados.csv: columnas ID, Nombre, Teléfono, Email, Estudios.
- situacion_empleados.csv: columnas ID, Tipo de Contrato, Salario, Fecha de Incorporación, Departamento, Cargo

¿Por qué importa esto en SynthData?

La capacidad de integrar datos de empleados permite a SynthData evaluar la situación del personal actual frente a los gastos en RRHH. Esto no solo optimiza la planificación de recursos, sino que también permite detectar oportunidades de crecimiento y áreas que requieren atención. Así, tu tarea apoya la alineación de los recursos humanos con las metas empresariales.

⛔ **Estos ejercicios son una simulación de cómo se podría resolver el problema en este contexto específico. Las soluciones encontradas no aplican de ninguna manera a todos los casos.**
Recordá que las soluciones dependen de los sets de datos, el contexto y los requerimientos específicos de los stakeholders y las organizaciones.

Buenos Aires aprende

Agencia de Habilidades para el Futuro

Buenos Aires Ciudad