



Python

GIORNO 5 - Pandas -

Raggruppamenti e operazioni con multipli DataFrame

Agenda:

- ❑ Pandas - Raggruppamenti
- ❑ Pandas - Indici avanzati (MultiIndex)
- ❑ Pandas - Merge, join e concat
- ❑ Pandas – Correlazione
- ❑ Programmazione assistita tramite AI



Perché?

- Come Data Analyst è fondamentale poter effettuare dei raggruppamenti di dati secondo determinate condizioni e conseguentemente aggregarli per gestire operazioni su larga scala.
- Spesso bisogna lavorare con dati separati in tabelle diverse, con la necessità di unirli in base a condizioni che possono variare in accordo con il contesto.
- Capita spesso di dover investigare pattern che possono causare o essere causati da altri pattern, e per arrivvarci è necessario prima individuare le correlazioni tra i dati.
- È davvero recente la possibilità di potersi servire di strumenti di Intelligenza Artificiale che ci possono coadiuvare nel lavoro.

Alla fine di questa lezione saprete:

- Utilizzare il metodo `.groupby()` per raggruppare i vostri dati ed effettuare operazioni specifiche sui raggruppamenti
- Gestire DataFrame con index a più livelli
- Utilizzare `.merge()`, `.join()` e `.concat()` per relazionare più DataFrame
- Cos'è e come calcolare la correlazione tra i dati
- Saper utilizzare diversi strumenti di AI per l'assistenza alla programmazione

Raggruppamenti

Effettuare dei raggruppamenti in base a fattori specifici ed essere in grado di effettuare operazioni su questi raggruppamenti è fondamentale per poter analizzare in modo esaustivo i dati.

In Pandas, le operazioni di raggruppamento sono gestite attraverso il metodo `.groupby()`

Nel dettaglio, il metodo `.groupby()` può effettuare diverse operazioni:

- **Divide:** Il DataFrame viene diviso in gruppi basati su un criterio specificato, che può essere una o più colonne del DataFrame o una funzione che opera sulle colonne.
- **Applica:** Una funzione viene applicata a ciascun gruppo separatamente. Questa funzione può essere una funzione di aggregazione (come `sum()`, `mean()`, `count()`, ecc.), una funzione personalizzata o una combinazione di funzioni.
- **Combina:** Infine, i risultati ottenuti da ciascun gruppo vengono combinati in un nuovo DataFrame o in un'altra struttura dati.

Raggruppamenti

Andiamo a vedere, nel dettaglio e con esempi, le varie operazioni:

Basandoci sul nostro DB AdventureWorksDW, leggiamo i dati dalla tabella *factresellersales* (al cui interno ci sono dei dati di vendita) e raggruppiamo questi dati per in base alla colonna *ProductKey*.

```
query = """SELECT *
           FROM factresellersales
         """
df = pd.read_sql(query, db_engine)
df.head()
✓ 1.9s Python
```

	SalesOrderNumber	SalesOrderLineNumber	OrderDate	DueDate	ShipDate	ProductKey	ResellerKey	PromotionKey	EmployeeKey	SalesTerritoryKey	OrderQuantity	UnitPrice	TotalProductCost	SalesAmount
0	SO43659	1	2017-07-01	2017-07-11	2017-07-08	349	676	1	285	5	1	2024.99	NaN	2024.99
1	SO43659	2	2017-07-01	2017-07-11	2017-07-08	350	676	1	285	5	3	2024.99	NaN	6074.97
2	SO43659	3	2017-07-01	2017-07-11	2017-07-08	351	676	1	285	5	1	2024.99	NaN	2024.99
3	SO43659	4	2017-07-01	2017-07-11	2017-07-08	344	676	1	285	5	1	2039.99	NaN	2039.99
4	SO43659	5	2017-07-01	2017-07-11	2017-07-08	345	676	1	285	5	1	2039.99	NaN	2039.99


```
grouped = df.groupby("ProductKey")
type(grouped)
✓ 0.0s Python
```

```
pandas.core.groupby.generic.DataFrameGroupBy
```

Raggruppamenti

Abbiamo un oggetto di tipo `pandas.core.groupby.generic.DataFrameGroupBy` su cui possiamo effettuare svariate operazioni.

Per accedere ad un gruppo specifico, ad esempio, usiamo il metodo `.get_group()`

grouped.get_group(349)

✓ 0.0s

Python

	SalesOrderNumber	SalesOrderLineNumber	OrderDate	DueDate	ShipDate	ProductKey	ResellerKey	PromotionKey	EmployeeKey	SalesTerritoryKey	OrderQuantity	UnitPrice	TotalProductCost	SalesAmount
0	SO43659	1	2017-07-01	2017-07-11	2017-07-08	349	676	1	285	5	1	2024.99	NaN	2024.99
24	SO43661	11	2017-07-02	2017-07-12	2017-07-09	349	442	1	288	6	4	2024.99	NaN	8099.96
69	SO43665	10	2017-07-04	2017-07-14	2017-07-11	349	146	1	289	1	1	2024.99	NaN	2024.99
113	SO43670	4	2017-07-08	2017-07-18	2017-07-15	349	504	1	281	3	1	2024.99	1898.09	2024.99
126	SO43672	2	2017-07-10	2017-07-20	2017-07-17	349	119	1	288	6	2	2024.99	3796.19	4049.98
...

In questo modo abbiamo recuperato gli oggetti che condividono lo stesso indice di raggruppamento, in questo caso il valore nella colonna `ProductKey`, e abbiamo filtrato per il gruppo la cui `ProductKey` è 349

Raggruppamenti

Ai raggruppamenti possiamo applicare dei metodi: vediamo per esempio come calcolare la media di *SalesAmount* per ogni *ProductKey*:

```
df.groupby("ProductKey")["SalesAmount"].mean()  
✓ 0.0s  
ProductKey  
212      56.935800  
213     92.780095  
214    119.202092  
215     60.224677  
216     94.880676  
...  
601     82.659280  
603    181.449574  
604    899.237551  
605   1184.882651  
606   988.962413  
Name: SalesAmount, Length: 334, dtype: float64
```

Se vogliamo scrivere tutto su una stessa riga, mettiamo tra parentesi tonde `()` il nome della colonna su cui effettuare il raggruppamento, tra parentesi quadre `[]` il nome della colonna su cui effettuare l'aggregazione (o applicare la funzione) e aggiungiamo poi il metodo specifico, in questo caso `.mean()`

Raggruppamenti

Possiamo anche raggruppare per un subset di colonne, specificando tra le parentesi tonde, invece che un solo nome, una lista di nomi. Per esempio:

```
df.groupby(["ProductKey", "OrderDate"])["SalesAmount"].mean()  
✓ 0.0s  
  
ProductKey  OrderDate  
212        2017-07-04    20.190000  
             2017-07-07    40.380000  
             2017-07-10    80.760000  
             2017-07-16    20.190000  
             2017-07-18    50.475000  
             ...  
606        2020-05-22   971.970000  
             2020-05-23   971.970000  
             2020-05-24   647.980000  
             2020-05-25  1619.950000  
             2020-05-28  1187.963333  
  
Name: SalesAmount, Length: 42508, dtype: float64
```

In questo modo calcoliamo la media (`.mean()`) del valore di vendita (`SalesAmount`) di ogni prodotto (`ProductKey`) per ogni giorno (`OrderDate`)

L'oggetto che otteniamo ha un indice a più livelli: vedremo cos'è e come gestirlo più avanti.

Raggruppamenti

Possiamo anche applicare **più di un'aggregazione alla volta**. In questo caso, abbiamo bisogno di fare riferimento al metodo `.agg()`, che applica una o più funzioni di aggregazione agli oggetti di ciascun gruppo, definendo come parametro una lista di aggregazioni:

```
df.groupby("ProductKey")["SalesAmount"].agg(["mean", "median"])
✓ 0.0s
```

ProductKey	mean	median
212	56.935800	40.38
213	92.780095	80.76
214	119.202092	104.95
215	60.224677	60.57
216	94.880676	80.76
...
601	82.659280	64.78
603	181.449574	145.78
604	899.237551	647.98
605	1184.882651	971.97
606	988.962413	647.98

Raggruppamenti

Possiamo anche combinare più colonne per il raggruppamento ed effettuare più aggregazioni contemporaneamente:

```
df.groupby(["ProductKey", "OrderDate"])["SalesAmount"].agg(["mean", "median"])
```

✓ 0.1s

ProductKey	OrderDate	mean	median
212	2017-07-04	20.190000	20.190
	2017-07-07	40.380000	40.380
	2017-07-10	80.760000	80.760
	2017-07-16	20.190000	20.190
	2017-07-18	50.475000	50.475
606
	2020-05-22	971.970000	971.970
	2020-05-23	971.970000	971.970
	2020-05-24	647.980000	647.980
	2020-05-25	1619.950000	1619.950
	2020-05-28	1187.963333	1619.950

Raggruppamenti

Alcuni altri metodi degli oggetti `GroupBy` sono:

- `.apply()`: Applica una funzione personalizzata a ciascun gruppo.
- `.count()`: Calcola il numero di elementi non nulli in ciascun gruppo.
- `.first()`: Restituisce il primo elemento di ciascun gruppo.
- `.last()`: Restituisce l'ultimo elemento di ciascun gruppo.
- `.mean()`: Calcola la media di ciascun gruppo.
- `.median()`: Calcola la mediana di ciascun gruppo.
- `.min()`: Restituisce il valore minimo di ciascun gruppo.
- `.max()`: Restituisce il valore massimo di ciascun gruppo.
- `.sum()`: Calcola la somma di ciascun gruppo.
- `.describe()`: Genera statistiche descrittive per ciascun gruppo.
- `.size()`: Restituisce il numero di elementi in ciascun gruppo.
- `.cumsum()`: Restituisce la somma cumulativa degli elementi per ciascun gruppo.

Raggruppamenti

Abbiamo elencato solo alcuni dei metodi disponibili per gli oggetti `GroupBy`.

`.groupby()` è un metodo molto potente e versatile: per coglierne appieno tutte le sfaccettature e le potenzialità facciamo riferimento alla [documentazione ufficiale](#), che presenta una [sottosezione dedicata](#) appunto agli oggetti `GroupBy`:

<https://pandas.pydata.org/pandas-docs/stable/reference/groupby.html#>

Salvataggio su disco

Se abbiamo bisogno di salvare le analisi che abbiamo fatto, possiamo usare i metodi per convertire i DataFrame in file, quali ad esempio `.to_csv()` oppure `.to_excel()`

```
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/refs/heads/master/iris.csv"

iris = pd.read_csv(url)

iris_statistics = iris.describe()

# Salva il DataFrame in formato CSV
iris_statistics.to_csv("iris_describe.csv", index=None)

# Salva il DataFrame in formato Excel
iris_statistics.to_excel("iris_describe.xlsx", sheet_name="statistics")
```

Salvataggio su disco

```
iris_describe.csv
```

UNREGISTERED

iris_describe.csv

1 |sepal_length,sepal_width,petal_length,petal_width
2 150.0,150.0,150.0,150.0
3 5.84333333333334,3.05733333333337,3.7
58000000000005,1.199333333333336
4 0.8280661279778629,0.435866284936698,1.7
652982332594667,0.7622376689603465
5 4.3,2.0,1.0,0.1
6 5.1,2.8,1.6,0.3
7 5.8,3.0,4.35,1.3
8 6.4,3.3,5.1,1.8
9 7.9,4.4,6.9,2.5
10

The screenshot shows the Microsoft Excel ribbon with the 'Home' tab selected. The main area displays a table of statistical summary data for the 'iris' dataset. The columns are labeled A through F, and the rows are numbered 1 through 10. The data includes measures like count, mean, standard deviation, min, 25%, 50%, 75%, and max for four features: sepal_length, sepal_width, petal_length, and petal_width. The 'Clipboard' group on the ribbon has a 'Copy' icon highlighted. The status bar at the bottom shows 'Ready' and '140%'.

	A	B	C	D	E	F
1	sepal_length	sepal_width	petal_length	petal_width		
2	count	150	150	150	150	
3	mean	5,843333333	3,057333333	3,758	1,199333333	
4	std	0,828066128	0,435866285	1,765298233	0,762237669	
5	min	4,3	2	1	0,1	
6	25%	5,1	2,8	1,6	0,3	
7	50%	5,8	3	4,35	1,3	
8	75%	6,4	3,3	5,1	1,8	
9	max	7,9	4,4	6,9	2,5	
10						

MultilIndex

Raggruppando per più colonne otteniamo un tipo di DataFrame particolare:
un DataFrame MultiIndex.

Un DataFrame con un MultiIndex in Pandas è un DataFrame che ha un indice gerarchico costituito da più livelli. Questo tipo di indice permette di rappresentare dati multidimensionali in un formato tabellare.

In un DataFrame con MultiIndex, ogni riga è identificata da una combinazione di valori corrispondenti ai livelli dell'indice gerarchico. Questi livelli possono essere considerati come colonne aggiuntive di etichette di indice che forniscono una struttura gerarchica ai dati.

MultIndex

Possiamo accedere ai dati di un DataFrame MultiIndex usando `.loc[]`, come per un DataFrame con indice singolo, ma specificando tra le parentesi quadre una tuple contenente i due valori gerarchici in ordine:

```
multiindex = df.groupby(["ProductKey", "PromotionKey"])["OrderDate"].agg(["min", "max"])
multiindex
✓ 0.1s

      min      max
ProductKey PromotionKey
212          1  2017-07-04  2018-06-27
              1  2018-08-01  2019-06-29
                  2  2018-08-07  2019-06-11
                  3  2018-09-02  2019-06-25
                  8  2018-07-01  2018-07-30
...
...
605          1  2019-07-02  2020-05-29
              2  2019-10-06  2020-04-16
              3  2019-07-05  2020-04-06
606          1  2019-07-02  2020-05-28
              2  2019-07-04  2019-12-13

588 rows × 2 columns

multiindex.loc[(213, 8)]
✓ 0.0s

min    2018-07-01
max    2018-07-30
Name: (213, 8), dtype: object
```

MultilIndex

Dato che `.iloc[]` non lavora sulle etichette ma sulle posizioni, non sarà necessario passargli due valori in quanto ogni riga avrà il suo identificativo univoco: `.iloc[0]` restituirà i valori della prima riga, non importa i livelli degli indici.

```
multiindex = df.groupby(["ProductKey", "PromotionKey"])["OrderDate"].agg(["min", "max"])
multiindex
✓ 0.1s

      min      max
ProductKey PromotionKey
212          1 2017-07-04 2018-06-27
213          1 2018-08-01 2019-06-29
              2 2018-08-07 2019-06-11
              3 2018-09-02 2019-06-25
              8 2018-07-01 2018-07-30
...
605          1 2019-07-02 2020-05-29
              2 2019-10-06 2020-04-16
              3 2019-07-05 2020-04-06
606          1 2019-07-02 2020-05-28
              2 2019-07-04 2019-12-13

588 rows × 2 columns

multiindex.iloc[0]
✓ 0.0s

min    2017-07-04
max    2018-06-27
Name: (212, 1), dtype: object
```

MultilIndex

Se vogliamo appiattire gli indici possiamo usare il metodo `.reset_index()`, che riasseggerà i valori del `MultiIndex` in maniera dinamica a due nuove colonne generando un nuovo indice numerico.

```
multiindex = df.groupby(["ProductKey", "PromotionKey"])["OrderDate"].agg(["min", "max"])
multiindex
0.1s
```

		min	max
ProductKey	PromotionKey		
212	1	2017-07-04	2018-06-27
213	1	2018-08-01	2019-06-29
	2	2018-08-07	2019-06-11
	3	2018-09-02	2019-06-25
	8	2018-07-01	2018-07-30
...
605	1	2019-07-02	2020-05-29
	2	2019-10-06	2020-04-16
	3	2019-07-05	2020-04-06
606	1	2019-07-02	2020-05-28
	2	2019-07-04	2019-12-13

588 rows × 2 columns


```
multiindex.reset_index()
0.0s
```

	ProductKey	PromotionKey	min	max
0	212	1	2017-07-04	2018-06-27
1	213	1	2018-08-01	2019-06-29
2	213	2	2018-08-07	2019-06-11
3	213	3	2018-09-02	2019-06-25
4	213	8	2018-07-01	2018-07-30
...

Merge

Un altro aspetto fondamentale del lavoro con i dati tabulari è la gestione di più tabelle.

Pandas prevede dei metodi specifici per unire le tabelle, sia in direzione orizzontale che verticale.

Il metodo più simile al concetto di JOIN che abbiamo visto con SQL è `.merge()`, che combina le righe di due tabelle basandosi su una relazione definita tra di esse.

Merge

`pandas.merge()` accetta più parametri. I principali sono:

- `left=`: DataFrame o Series da unire
- `right=`: DataFrame o Series da unire
- `how=`: Tipo di unione da eseguire (left, right, inner, outer, cross). Default: "inner".
- `on=`: Nome o lista di nomi di colonne su cui effettuare l'unione
- `left_on=`: Se le colonne dei due DataFrame su cui vogliamo effettuare l'unione si chiamano in modo diverso, specifichiamo qui quelle del DataFrame di sinistra. Default: None
- `right_on=`: Se le colonne dei due DataFrame su cui vogliamo effettuare l'unione si chiamano in modo diverso, specifichiamo qui quelle del DataFrame di destra. Default: None

Per una visualizzazione completa dei parametri, riferiamoci alla documentazione ufficiale:

<https://pandas.pydata.org/docs/reference/api/pandas.merge.html>

Esiste anche il metodo `pandas.DataFrame.merge()` che è del tutto equivalente; l'unica differenza è che non c'è bisogno di specificare il parametro `left=`

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.merge.html>

Merge

Ormai conosciamo bene la tabella *factresellersales* del DB AdventureWorks e sappiamo che contiene delle chiavi numeriche, come ad esempio *ProductKey* e *PromotionKey*. Le chiavi numeriche in *ProductKey* sono esplicite testualmente dentro la tabella *dimproduct*:

```
query_dimproduct = """SELECT *
    FROM dimproduct
    """
dimproduct = pd.read_sql(query_dimproduct, db_engine)
dimproduct.head()
```

✓ 1.0s

	ProductKey	ProductAlternateKey	ProductSubcategoryKey	WeightUnitMeasureCode	SizeUnitMeasureCode	EnglishProductName
0	1	AR-5381		NaN	None	Adjustable Race
1	2	BA-8327		NaN	None	Bearing Ball
2	3	BE-2349		NaN	None	BB Ball Bearing
3	4	BE-2908		NaN	None	Headset Ball Bearings
4	5	BL-2036		NaN	None	Blade

Merge

Per unire le due tabelle, usiamo `.merge()` partendo da `factresellersales` e specificando i parametri che ci interessano:

```

query_dimproduct = """SELECT *
    FROM dimproduct
    """
dimproduct = pd.read_sql(query_dimproduct, db_engine)
query_factresellersales = """SELECT *
    FROM factresellersales
    """
factresellersales = pd.read_sql(query_factresellersales, db_engine)

factresellersales.merge(dimproduct, how="left", on="ProductKey")

```

3:1s

	SalesOrderNumber	SalesOrderLineNumber	OrderDate	DueDate	ShipDate	ProductKey	ResellerKey	PromotionKey	EmployeeKey	SalesTerritoryKey	...	ChineseDescription	ArabicDescription	HebrewDescription	ThaiDescription	GermanDescription	JapaneseDescription	TurkishDescription	Start
0	SO43659	1	2017-07-01	2017-07-11	2017-07-08	349	676	1	285	5	-	高档竞赛山地车。性能得到进一步增强，包括创新的 HL 车架、极其平稳的前悬架以及适用于所有地...	درية سباقات مخصصة للقيادة في... الجبال من أعلى ط...	אופני רמה גבוהה, מושגיה המהירה, אמינותה��檔競賽山地車。性能得到進一步增強，包括創新的 HL 車架、極其平穩的前懸...	硕爾德山地車 iskačivački vježbački สกู๊ปเดลจิจิร์ดาน...	-Top-Mountainbike für Profis. Mit innovativem ...	最高級の競技用マウ... シティンバイクです。 性能強化オプション... としては、革新的な... HL フレーム、...	"En üst model rekabetçi dağ bisikleti. Perform... En üst model rekabetçi dağ bisikleti. Perform...	2011
1	SO43659	2	2017-07-01	2017-07-11	2017-07-08	350	676	1	285	5	-	高档竞赛山地车。性能得到进一步增强，包括创新的 HL 车架、极其平稳的前悬架以及适用于所有地...	درية سباقات مخصصة للقيادة في... الجبال من أعلى ط...	אופני רמה גבוהה, מושגיה המהירה, אמינותה高檔競賽山地車。性能得到進一步增強，包括創新的 HL 車架、極其平穩的前懸...	硕爾德山地車 iskačivački vježbački สกู๊ปเดลจิจิร์ดาน...	-Top-Mountainbike für Profis. Mit innovativem ...	最高級の競技用マウ... シティンバイクです。 性能強化オプション... としては、革新的な... HL フレーム、...	"En üst model rekabetçi dağ bisikleti. Perform... En üst model rekabetçi dağ bisikleti. Perform...	2011

Merge

L'operazione restituisce un DataFrame composto da tutte le colonne di entrambi quelli inclusi nel merge. Possiamo limitare il campo andando a selezionare solo le colonne che ci interessano

```
merged = factresellersales.merge(dimproduct, how="left", on="ProductKey")[['SalesOrderNumber', 'SalesOrderLineNumber', 'OrderDate', 'ProductKey', 'EnglishProductName']]  
merged
```

✓ 4.8s

	SalesOrderNumber	SalesOrderLineNumber	OrderDate	ProductKey	EnglishProductName
0	SO43659	1	2017-07-01	349	Mountain-100 Black, 42
1	SO43659	2	2017-07-01	350	Mountain-100 Black, 44
2	SO43659	3	2017-07-01	351	Mountain-100 Black, 48
3	SO43659	4	2017-07-01	344	Mountain-100 Silver, 38
4	SO43659	5	2017-07-01	345	Mountain-100 Silver, 42
...
57846	SO69565	3	2020-05-31	565	Touring-3000 Blue, 54
57847	SO69565	4	2020-05-31	572	Touring-3000 Yellow, 62
57848	SO69565	5	2020-05-31	560	Touring-2000 Blue, 60
57849	SO69565	6	2020-05-31	555	Front Brakes
57850	SO69565	7	2020-05-31	484	Bike Wash - Dissolver

Join

Un altro metodo per unire più DataFrame è `.join()`

`pandas.DataFrame.join()` accetta più parametri. I principali sono:

- `other=`: DataFrame o Series da unire
- `on=`: Nome(i) di colonna o livello di indice nel chiamante su cui effettuare l'unione sull'indice in `other`, se nulla è specificato unisce indice su indice. Default: `None`
- `how=`: Tipo di unione da eseguire (`left`, `right`, `inner`, `outer`, `cross`). Default: "left".

Il DataFrame su cui stiamo utilizzando il metodo è considerato quello di sinistra, mentre quello che inseriamo nel parametro `other=` è considerato quello di destra.

Per una visualizzazione completa dei parametri, riferiamoci alla documentazione ufficiale:

<https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.join.html#pandas.DataFrame.join>

La differenza principale rispetto a `.merge()` sta proprio nel parametro `on=`: se `.merge()` lavora sulle colonne, `.join()` è pensato per lavorare sugli indici.

Join

Questa differenza ne implica altre, a cascata:

- **Ricerca nella tabella di destra:** `df1.join(df2)` effettua sempre la ricerca tramite l'indice di `df2`, ma `pd.merge(df1, df2)` può effettuare la ricerca su una o più colonne di `df2` (predefinito) o sull'indice di `df2` (con `right_index=True`)
- **Ricerca nella tabella di sinistra:** per impostazione predefinita, `df1.join(df2)` utilizza l'indice di `df1` e `pd.merge(df1, df2)` utilizza la/e colonna/e di `df1`. Questo può essere sovrascritto specificando `df1.join(df2, on=chiave_o_chiavi)` o `pd.merge(df1, df2, left_index=True)`
- **Left join vs inner join:** `.join()` esegue un join sinistro per impostazione predefinita (mantiene tutte le righe della tabella di sinistra), ma `merge()` esegue un join interno per impostazione predefinita (restituisce solo le righe corrispondenti delle due tabelle).

Join

Per unire le due tabelle *factresellersales* e *dimproduct* con un `.join()` invece che con un `merge()` dovremo quindi effettuare delle operazioni in più:

```
factresellersales = factresellersales.set_index("ProductKey")
dimproduct = dimproduct.set_index("ProductKey")
factresellersales.join(dimproduct)[['SalesOrderNumber', 'SalesOrderLineNumber', 'OrderDate', 'EnglishProductName']]
```

✓ 34s

ProductKey	SalesOrderNumber	SalesOrderLineNumber	OrderDate	EnglishProductName
349	SO43659	1	2017-07-01	Mountain-100 Black, 42
350	SO43659	2	2017-07-01	Mountain-100 Black, 44
351	SO43659	3	2017-07-01	Mountain-100 Black, 48
344	SO43659	4	2017-07-01	Mountain-100 Silver, 38
345	SO43659	5	2017-07-01	Mountain-100 Silver, 42
...
565	SO69565	3	2020-05-31	Touring-3000 Blue, 54
572	SO69565	4	2020-05-31	Touring-3000 Yellow, 62
560	SO69565	5	2020-05-31	Touring-2000 Blue, 60
555	SO69565	6	2020-05-31	Front Brakes
484	SO69565	7	2020-05-31	Bike Wash - Dissolver

Concat

Per unire due `DataFrame` in direzione “verticale”, cioè per aggiungere ad uno le righe di un altro, usiamo la funzione `.concat()`

Anche `pandas.concat(obj, ...)` accetta diversi parametri, ma quello che ci interessa è uno:

- `obj=`: Gli oggetti da concatenare

Per approfondire facciamo riferimento alla documentazione: <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html>

IMPORTANTE: `.concat()` non è un metodo dell’oggetto `DataFrame`, ma una funzione di Pandas. Non la invocheremo tramite, ad esempio, `DataFrame1.concat(DataFrame2)`, ma tramite `pd.concat([DataFrame1, DataFrame2])`; vuole cioè in input una `list` contenente i `DataFrame` da concatenare

Concat

Per esempio: nel DB AdventureWorksDW esiste anche una tabella *factinternetsales*, che contiene le informazioni sulle vendite effettuate via internet. Per avere una panoramica totale delle vendite ho bisogno di prendere in considerazione tutte le righe di *factresellersales* e *factinternetsales*:

```
query_factinternetsales = """SELECT *
    FROM factinternetsales
"""
factinternetsales = pd.read_sql(query_factinternetsales, db_engine)
pd.concat([factresellersales, factinternetsales])
✓ 3.6s
```

	SalesOrderNumber	SalesOrderLineNumber	OrderDate	DueDate	ShipDate	ResellerKey	PromotionKey	EmployeeKey	SalesTerritoryKey	OrderQuantity	CustomerKey	CurrencyKey	ExtendedAmount	UnitPriceDiscountPct	DiscountAmount	ProductStandardCost	TaxAmt	Freight
349	SO43659	1	2017-07-01	2017-07-11	2017-07-08	676.0	1	285.0	5	1 ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
350	SO43659	2	2017-07-01	2017-07-11	2017-07-08	676.0	1	285.0	5	3 ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
351	SO43659	3	2017-07-01	2017-07-11	2017-07-08	676.0	1	285.0	5	1 ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
344	SO43659	4	2017-07-01	2017-07-11	2017-07-08	676.0	1	285.0	5	1 ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
345	SO43659	5	2017-07-01	2017-07-11	2017-07-08	676.0	1	285.0	5	1 ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
...
60393	SO75122	1	2021-01-28	2021-02-07	None	NaN	1	NaN	6	1 ...	15868.0	100.0	21.98	0.0	0.0	8.22	1.76	0.55
60394	SO75122	2	2021-01-28	2021-02-07	None	NaN	1	NaN	6	1 ...	15868.0	100.0	8.99	0.0	0.0	6.92	0.72	0.22
60395	SO75123	1	2021-01-28	2021-02-07	None	NaN	1	NaN	6	1 ...	18759.0	100.0	21.98	0.0	0.0	8.22	1.76	0.55
60396	SO75123	2	2021-01-28	2021-02-07	None	NaN	1	NaN	6	1 ...	18759.0	100.0	159.00	0.0	0.0	59.47	12.72	3.98
60397	SO75123	3	2021-01-28	2021-02-07	None	NaN	1	NaN	6	1 ...	18759.0	100.0	8.99	0.0	0.0	6.92	0.72	0.22

Concat

Attenzione ai valori nulli! Vediamo comparire dei `None` e dei `NaN` in alcune colonne. Questo succede perché, mentre alcune colonne sono in comune tra le due tabelle, alcune sono presenti solo in una o nell'altra.

Nell'unire le righe Pandas istanzia tutte le colonne di cui può avere bisogno e, nel caso non trovi il valore per riempirle in una determinata riga, le riempie con dei valori vuoti.

I valori vuoti sono una problematica importante con cui ci confronteremo nella prossima lezione.

Per ora, per approfondire ulteriormente i metodi di unione di più `DataFrame`, andiamo a riferirci alla documentazione ufficiale che contiene una spiegazione esaustiva:

https://pandas.pydata.org/pandas-docs/stable/user_guide/merging.html

Somma cumulativa

Il metodo per eseguire una somma cumulativa è `.cumsum()`

```
import pandas as pd

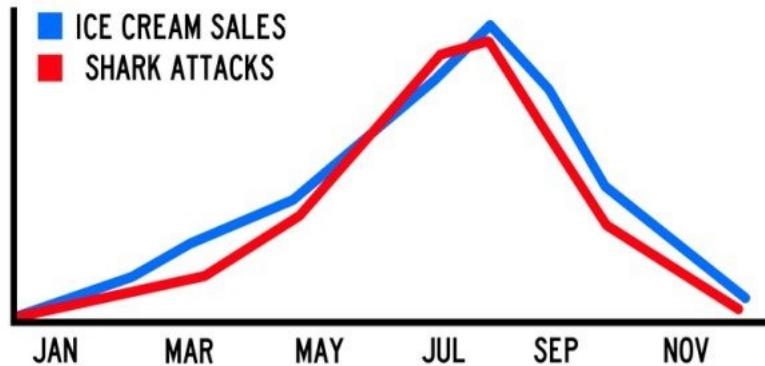
data = {
    "Giorno": ["Lunedì", "Martedì", "Mercoledì", "Giovedì", "Venerdì"],
    "Vendite": [100, 200, 150, 300, 250]
}
df = pd.DataFrame(data)

df.loc[:, "Vendite Cumulative"] = df.loc[:, "Vendite"].cumsum()

print(df)

      Giorno  Vendite  Vendite Cumulative
0     Lunedì      100              100
1    Martedì      200              300
2   Mercoledì     150              450
3   Giovedì       300              750
4   Venerdì       250             1000
```

Correlazione



Correlazione

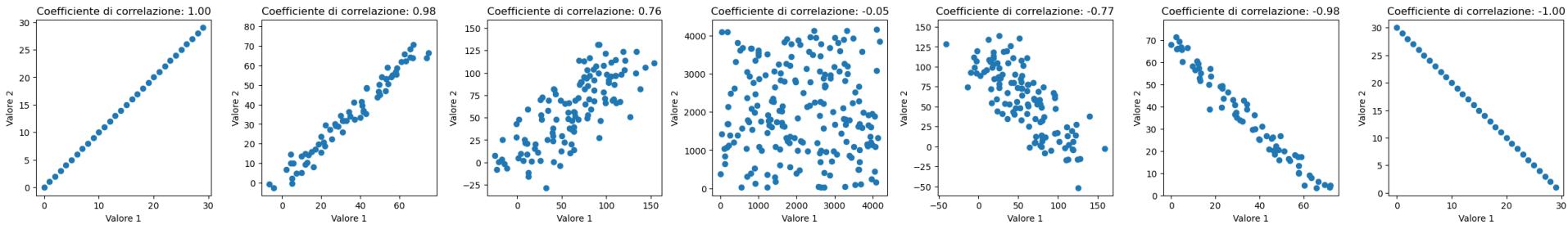
Può capitare spesso di voler sapere se alcune colonne sono **correlate** tra di loro.

La **correlazione** è un concetto statistico che misura la relazione e il grado di dipendenza tra due variabili. In pratica, indica quanto e come due variabili cambiano insieme. La correlazione viene spesso espressa attraverso un valore chiamato coefficiente di correlazione, che varia tra -1 e 1.

- Correlazione **positiva o diretta** (valori tra 0 e 1): quando una variabile aumenta, anche l'altra aumenta.
Esempio: Più alta è la temperatura → più gelati vengono venduti.
- Correlazione **negativa o inversa** (valori tra -1 e 0): quando una variabile aumenta, l'altra diminuisce.
Esempio: Aumentare l'uso dello smartphone → ridurre la durata della batteria.
- **Nessuna** correlazione (valore 0): non c'è una relazione evidente tra le due variabili.
Esempio: Il numero di scarpe che possiedi e la temperatura esterna.

Correlazione

Ecco degli esempi di due valori e i loro coefficienti di correlazione:



È importante conoscere la correlazione tra dei set di dati (tipicamente tra **colonne di una tabella**) perché se i dati hanno una correlazione (diretta o inversa) può esserci una **causalità**, cioè una delle due influenza l'altra, o magari c'è un terzo fattore che influenzano entrambi.

L'esistenza di **correlazione non implica automaticamente causalità**, ma è un indizio molto forte per andare a investigare se è effettivamente così o meno.

Correlazione

Leggiamo il dataset `mpg` che contiene informazioni su caratteristiche di diverse auto:

```
url = "https://raw.githubusercontent.com/mwaskom/seaborn-data/refs/heads/master/mpg.csv"
auto = pd.read_csv(url)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year	origin	name	
0	18.0	8	307.0	130.0	3504	12.0	70	usa	chevrolet chevelle malibu	
1	15.0	8	350.0	165.0	3693	11.5	70	usa	buick skylark 320	
2	18.0	8	318.0	150.0	3436	11.0	70	usa	plymouth satellite	
3	16.0	8	304.0	150.0	3433	12.0	70	usa	amc rebel sst	
4	17.0	8	302.0	140.0	3449	10.5	70	usa	ford torino	
...	
393	27.0	4	140.0	86.0	2790	15.6	82	usa	ford mustang gl	
394	44.0	4	97.0	52.0	2130	24.6	82	europe	vw pickup	
395	32.0	4	135.0	84.0	2295	11.6	82	usa	dodge rampage	
396	28.0	4	120.0	79.0	2625	18.6	82	usa	ford ranger	
397	31.0	4	119.0	82.0	2720	19.4	82	usa	chevy s-10	

Correlazione

I DataFrame hanno un metodo per calcolare automaticamente la correlazione tra tutte le colonne (numeriche), ovvero `.corr()`

```
auto.corr(numeric_only=True)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000

Correlazione

I DataFrame hanno un metodo per calcolare automaticamente la correlazione tra tutte le colonne (numeriche), ovvero `.corr()`

```
auto.corr(numeric_only=True)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000

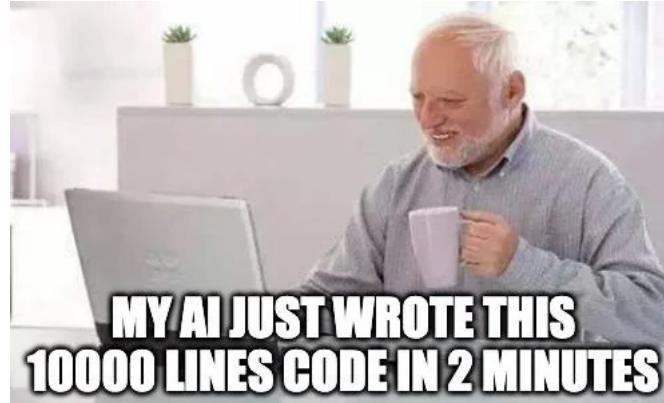
Correlazione

I DataFrame hanno un metodo per calcolare automaticamente la correlazione tra tutte le colonne (numeriche), ovvero `.corr()`

```
auto.corr(numeric_only=True)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model_year
mpg	1.000000	-0.775396	-0.804203	-0.778427	-0.831741	0.420289	0.579267
cylinders	-0.775396	1.000000	0.950721	0.842983	0.896017	-0.505419	-0.348746
displacement	-0.804203	0.950721	1.000000	0.897257	0.932824	-0.543684	-0.370164
horsepower	-0.778427	0.842983	0.897257	1.000000	0.864538	-0.689196	-0.416361
weight	-0.831741	0.896017	0.932824	0.864538	1.000000	-0.417457	-0.306564
acceleration	0.420289	-0.505419	-0.543684	-0.689196	-0.417457	1.000000	0.288137
model_year	0.579267	-0.348746	-0.370164	-0.416361	-0.306564	0.288137	1.000000

Programmazione assistita con AI



Programmazione assistita con AI

Per scrivere codice, in particolare riguardo algoritmi che hanno parti ripetitive o possibilmente tediose, ma non solo, è possibile servirsi di Intelligenza Artificiale Generativa (Generative AI).

L'IA Generativa è capace per l'appunto di generare dei risultati di varia natura (testuale, audio, grafica) a partire dalla richiesta da parte di un utente che viene espressa in linguaggio naturale (chiamata *prompt*). A seconda dei dati su cui sono addestrate, le IA Generative possono specializzarsi in diversi ambiti, ed esistono anche quelle addestrate su codice Python, che possono aiutarci a scrivere codice e rendere il nostro lavoro più snello e più veloce, e persino se siamo bloccati su qualcosa in particolare e non conosciamo ancora i metodi per risolverlo.

Una volta scritto il *prompt* desiderato, il codice risultante può essere utilizzato così com'è, o modificato in base alle necessità; se il risultato non fosse quello desiderato, bisognerà modificare il *prompt*, magari rendendolo più dettagliato (ricordiamo che, differentemente dai linguaggi di programmazione che hanno una sintassi rigida e un risultato prevedibile, l'uso del linguaggio naturale significa anche perdere la assoluta prevedibilità del risultato).

ChatGPT

- Uno dei possibili tool esterni da utilizzare è ChatGPT: <https://chatgpt.com/>
 - Si tratta di un tool generico che può generare risposte rispetto a moltissimi altri ambiti e in altri formati

The screenshot shows the ChatGPT interface. On the left, a user message is displayed in a grey box: "Scrivimi il codice per ordinare un DataFrame in maniera nidificata rispetto a due colonne". Below it, a response message says: "Ecco un esempio di codice Python che ordina un DataFrame `pandas` in maniera nidificata rispetto a due colonne:". At the bottom left is a button labeled "Ordinamento Dataframe". On the right, a code editor window titled "Ordinamento Dataframe" contains the following Python code:

```
1 import pandas as pd
2
3 # Creazione di un DataFrame di esempio
4 dati = {
5     'Nome': ['Alice', 'Bob', 'Charlie', 'David', 'Eve'],
6     'Età': [25, 30, 25, 35, 30],
7     'Punteggio': [85, 90, 75, 95, 90]
8 }
9 df = pd.DataFrame(dati)
10
11 # Ordinamento nidificato prima per 'Età' (crescente) e poi per 'Punteggio' (decrescente)
12 df_ordinato = df.sort_values(by=['Età', 'Punteggio'], ascending=[True, False])
13
14 # Stampa del DataFrame ordinato
15 print(df_ordinato)
16
```

Programmazione assistita con AI

Esistono altri tool equivalenti a ChatGPT, che hanno capacità generiche, quali:

- Claude AI: <https://claude.ai/>
- DeepSeek: <https://www.deepseek.com/>
- Vitruvian: <https://vitruvian.asc27.com/>

Oppure tool specifici per la programmazione, quali:

- GitHub Copilot: <https://github.com/features/copilot>
- Codeium: <https://www.codeium.com/>
- Tabnine: <https://www.tabnine.com/>

Esistono anche tool AI già integrati, sia in Anaconda che in Colaboratory

- Anaconda Assistant in Jupyter Lab
- Google Codey in Colaboratory

Claude AI

Claude  Nested DataFrame Sorting by Two Columns ▾[Share](#)

 Scrivimi il codice Python per ordinare un DataFrame in maniera nidificata rispetto a due colonne.

Ecco il codice Python per ordinare un DataFrame in maniera nidificata rispetto a due colonne:

```
python
import pandas as pd

# Supponiamo di avere un DataFrame chiamato df
# df = pd.read_csv('tuo_file.csv') # Oppure creane uno di esempio

# Per ordinare il DataFrame in modo nidificato rispetto alle colonne 'colonna1' e
df_ordinato = df.sort_values(by=['colonna1', 'colonna2'])

# Se vuoi ordinare la prima colonna in modo ascendente e la seconda in modo discendente
df_ordinato_misto = df.sort_values(by=['colonna1', 'colonna2'], ascending=[True, False])

# Per visualizzare il risultato
print(df_ordinato)
```

Vitruvian



VITRUVIAN-1

+ New Chat

Today

Codice Fibonacci Python

Codice Fibonacci Python

This month

Greeting

Reasoning
Done

Ecco il codice Python per calcolare i primi dieci numeri di Fibonacci:

```
def fibonacci(n):
    fib_sequence = [0, 1] # Inizializza il primo e il secondo numero di Fibonacci
    for _ in range(2, n): # Cicla per generare i successivi n-2 numeri
        next_fib = fib_sequence[-1] + fib_sequence[-2]
        fib_sequence.append(next_fib)
    return fib_sequence

# Ottieni i primi 10 numeri di Fibonacci (da F(0) a F(9))
print(fibonacci(10))
```

Spiegazione del codice:

1. La funzione `fibonacci(n)` genera i primi `n` numeri di Fibonacci.
2. Inizializza la lista `fib_sequence` con i primi due numeri di Fibonacci `[0, 1]`.
3. Il ciclo viene eseguito `n-2` volte (poiché sono già stati inizializzati due numeri) per generare i successivi numeri.
4. Ogni iterazione calcola il numero di Fibonacci successivo sommando gli ultimi due elementi, quindi aggiungendolo alla lista.

Codey (Colaboratory)

Genera Scrivi il codice che seleziona i valori pari, maggiori di 50 e che abbiano indice "a" nella variabile "valori", usando l'attributo loc | [Chiudi](#)

< 1 of 1 > [Usa il codice con cautela](#)

```
1 # prompt: Scrivi il codice che seleziona i valori pari, maggiori di 50 e che abbiano indice "a" nella variabile "valori"
2
3 import pandas as pd
4 valori = pd.Series([100, 150, 25, 40, 31, 78, 102, 123, 55],
5                     index=list("aaabbbaaa"))
6
7 # Seleziona i valori pari, maggiori di 50 e con indice "a"
8 valori_selezionati = valori.loc[(valori % 2 == 0) & (valori > 50) & (valori.index == 'a')]
9 valori_selezionati
10
```

0

a	100
a	150
a	102

dtype: int64

Codey (Colaboratory)

Genera Scrivi il codice di un semplice gioco dove bisogna indovinare un numero ↑ ↓ ⚡ 🔍 🌐 🛡️ 🌐 🌐 🌐 🌐 🌐 Chiudi

< 1 of 1 > 👍 👎 [Usa il codice con cautela](#)

29 s ▶

```
1 # prompt: Scrivi il codice di un semplice gioco dove bisogna indovinare un numero
2
3 import random
4
5 def number_guessing_game():
6     """A simple number guessing game."""
7
8     secret_number = random.randint(1, 100)
9     guesses_left = 7
10
11    print("Benvenuto al gioco dell'indovinello!")
12    print("Ho pensato a un numero tra 1 e 100.")
13
14    while guesses_left > 0:
15        try:
16            guess = int(input(f"Hai {guesses_left} tentativi. Indovina il numero: "))
```

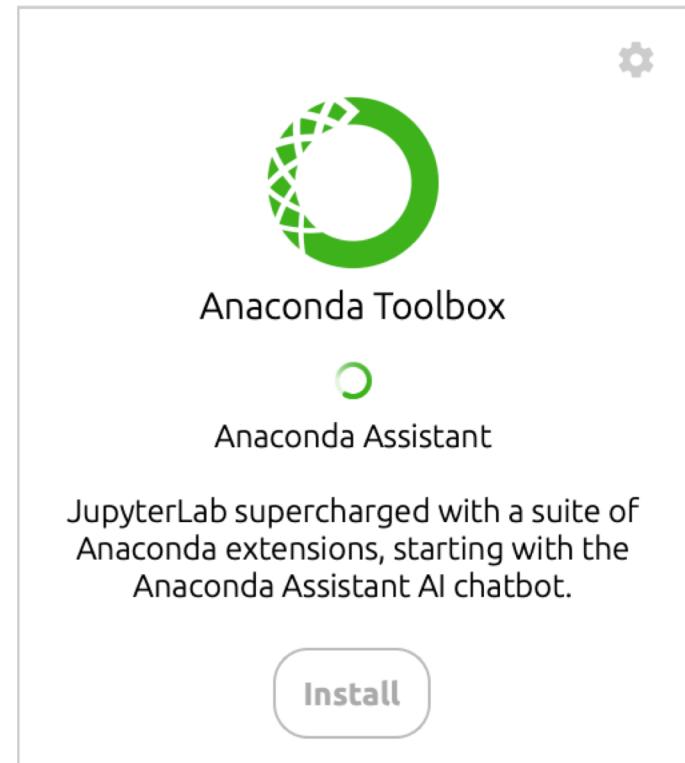
Codey (Colaboratory)

```
28
29     guesses_left -= 1
30
31 print(f"Hai esaurito i tentativi. Il numero era {secret_number}.")
32
33 if __name__ == "__main__":
34     number_guessing_game()
35
```

```
→ Benvenuto al gioco dell'indovinello!
Ho pensato a un numero tra 1 e 100.
Hai 7 tentativi. Indovina il numero: 50
Troppo basso!
Hai 6 tentativi. Indovina il numero: 75
Troppo basso!
Hai 5 tentativi. Indovina il numero: 87
Troppo basso!
Hai 4 tentativi. Indovina il numero: 92
Troppo basso!
Hai 3 tentativi. Indovina il numero: 96
Troppo basso!
Hai 2 tentativi. Indovina il numero: 98
Congratulazioni! Hai indovinato il numero 98 in 5 tentativi.
```

Anaconda Assistant (Anaconda/Jupyter Lab)

Per poter utilizzare Anaconda Assistant in Jupyter Lab
è necessario installare il kit Anaconda Toolbox dal
Navigator di Anaconda



Anaconda Assistant (Anaconda/Jupyter Lab)

Sandbox.ipynb Share Notebook Python [conda env:base]

```
[16]: import pandas as pd  
df = pd.DataFrame({"label": ["a", "b", "c", "a", "a", "b"],  
                    "value": [123, 456, 789, 210, 543, 876]})
```

Assistant

Ti aiuterò a ordinare e filtrare il DataFrame utilizzando pandas.

```
[13]: # Assumendo che df sia già definito  
# 1. Ordina il DataFrame per la colonna "label"  
# 2. Filtra per valori >= 500 nella colonna "value"  
  
df_filtered = df.sort_values(by='label').query('value >= 500')  
  
# In alternativa, si può scrivere anche così:  
# df_filtered = df.sort_values(by='label')[df['value'] >= 500]  
  
# Visualizza il risultato  
print(df_filtered)
```

label	value
a	543
b	876
c	789

Spiegazione del codice:

1. `sort_values(by='label')` : ordina il DataFrame in base alla colonna "label" in ordine crescente

Anaconda Assistant 4.1.0

Ordina il DataFrame "df" per la colonna "label" e poi filtra la colonna "value" per i valori maggiori o uguali a 500

```
# Assumendo che df sia già definito  
# 1. Ordina il DataFrame per la colonna "label"  
# 2. Filtra per valori >= 500 nella colonna "value"  
  
df_filtered = df.sort_values(by='label').query('value >= 500')  
  
# In alternativa, si può scrivere anche così:  
# df_filtered = df.sort_values(by='label')[df['value'] >= 500]  
  
# Visualizza il risultato  
print(df_filtered)
```

Spiegazione del codice:

1. `sort_values(by='label')` : ordina il DataFrame in base alla colonna "label" in ordine crescente

Anaconda Assistant (Anaconda/Jupyter Lab)

The screenshot shows a Jupyter Notebook interface with a file named "Sandbox.ipynb". The code cell contains a Python script for a simple game. The script imports the random module and defines a play_game function. The function prompts the user for input, handles invalid inputs, generates a random choice for the computer, prints both player and computer choices, and determines the winner.

```
import random

def play_game():
    choices = ["sasso", "carta", "forbice"]

    while True:
        # Input del giocatore
        player = input("\nScegli sasso, carta o forbice (o 'q' per uscire): ")

        if player == 'q':
            print("Grazie per aver giocato!")
            break

        if player not in choices:
            print("Scelta non valida! Riprova.")
            continue

        # Scelta computer
        computer = random.choice(choices)

        # Mostra le scelte
        print(f"\nTu hai scelto: {player}")
        print(f"Il computer ha scelto: {computer}")

        # Determina il vincitore
        if player == computer:
```

The screenshot shows the Anaconda Assistant 4.1.0 interface. A message box asks the user to write the code for a Rock, Paper, Scissors game. Below it, the assistant provides a template for the code, which is identical to the one shown in the Jupyter Notebook.

Scrivi il codice per carta, sasso, forbice (using no context from Notebook)

Let me help you create a simple Rock, Paper, Scissors game in Python.

```
import random

def play_game():
    choices = ["sasso", "carta", "forbice"]

    while True:
        # Input del giocatore
        player = input("\nScegli sasso, carta o forbice")

        if player == 'q':
            print("Grazie per aver giocato!")
            break

        if player not in choices:
            print("Scelta non valida! Riprova.")
```

Anaconda Assistant (Anaconda/Jupyter Lab)

```
continue

# Scelta computer
computer = random.choice(choices)

# Mostra le scelte
print(f"\nTu hai scelto: {player}")
print(f"Il computer ha scelto: {computer}")

# Determina il vincitore
if player == computer:
    print("Pareggio!")
elif (player == "sasso" and computer == "forbice") or \
    (player == "carta" and computer == "sasso") or \
    (player == "forbice" and computer == "carta"):
    print("Hai vinto!")
else:
    print("Il computer ha vinto!")

# Avvia il gioco
print("Benvenuto a Sasso, Carta, Forbice!")
play_game()
```

Benvenuto a Sasso, Carta, Forbice!

Scegli sasso, carta o forbice (o 'q' per uscire): sasso

Tu hai scelto: sasso
Il computer ha scelto: forbice
Hai vinto!

Scegli sasso, carta o forbice (o 'q' per uscire): q
Grazie per aver giocato!

```
# Avvia il gioco
print("Benvenuto a Sasso, Carta, Forbice!")
play_game()
```



Questo codice:

1. Importa il modulo `random` per la scelta casuale del computer
2. Definisce una funzione `play_game()` che contiene la logica del gioco
3. Usa un loop while per permettere partite multiple
4. Gestisce l'input del giocatore e verifica che sia valido
5. Genera una scelta casuale per il computer
6. Determina il vincitore secondo le regole del gioco
7. Permette di uscire digitando 'q'

Per eseguire il gioco, basta copiare e incollare il codice in un notebook Jupyter e eseguirlo. Il gioco continuerà finché il giocatore non decide di uscire digitando 'q'.



Reply to this chat...



Chat 24% full

Attach to chat



GRAZIE
EPCODE