In Python tutto è un oggetto!

Oggetti di tipo intero, float, stringa, null, bool e tuple

Stringhe

Stampiamo l'oggetto "Hello World"

In [1]:
```
print("Hello World")
```

```
Hello World
```

Stabiliamo il tipo dell'oggetto "Hello World" (per meglio dire la "classe")

In [2]:
```
print(type("Hello World"))
```

```
<class 'str'>
```

Assegniamo all'oggetto "Hello World" il nome x

In [3]:
```
x = "Hello World"
```

Stampiamo x (che è il nome che abbiamo assegnato all'oggetto "Hello World")

In [4]:
```
print(x)
```

```
Hello World
```

Stampiamo il tipo di x (che è il nome che abbiamo assegnato all'oggetto "Hello World")

In [5]:
```
print(type(x))
```

```
<class 'str'>
```

Assegniamo al nome y, lo stesso oggetto a cui ho assegnato il nome x

In [6]:
```
y = x
```

Stampiamo y

In [7]:
```
print(y)
```

```
Hello World
```

Posso accedere facilmente a un particolare carattere della stringa

In [8]:
```
print(y[1])
```

```
e
```

In [9]:
```
print(y[1:5])
```

```
ello
```

Analizziamo altri tipi. Partiamo dagli interi

Stampiamo l'oggetto 1

In [10]:
```
print(1)
```

```
1
```

Stabiliamo il tipo dell'oggetto 1

In [11]:
```
print(type(1))
```

```
<class 'int'>
```

Se assegnasi a l'oggetto 1 un determinato nome (scrivendo ad esempio x = 1) varrebbero tutte le considerazioni fatte in precedenza con le stringhe

Float

Stampiamo l'oggetto 1.2

In [12]:
```
print(1.2)
```

```
1.2
```

Stabiliamo il tipo dell'oggetto 1.2

In [13]:
```
print(type(1.2))
```

```
<class 'float'>
```

Booleani

Stampiamo l'oggetto True

In [14]:
```
print(True)
```

```
True
```

Stabiliamo il tipo dell'oggetto "Nicola"

In [15]:
```
print(type(True))
```

```
<class 'bool'>
```

None

Stampiamo l'oggetto None

In [16]:
```
print(None)
```

```
None
```

Stabiliamo il tipo dell'oggetto "None"

In [17]:
```
print(type(None))
```

```
<class 'NoneType'>
```

Tuple

Stampiamo la tupla (1,"a",3,1)

In [18]:
```
print((1,"a",3,1))
```

```
(1, 'a', 3, 1)
```

In [19]:
```
print(type((1,"a",3,1)))
```

```
<class 'tuple'>
```

Alle tuple posso applicare anche altre funzioni e metodi

In [20]:
```
print(len((1,"a",3,1)))
```

```
4
```

In [21]:
```
print((1,"a",3,1).count(3))
```

```
1
```

In [22]:
```
print((1,"a",3,1).index(3))
```

```
2
```

Inoltre posso accedere ad un elemento specifico della tupla

In [23]:
```
print((1,"a",3,1)[0])
```

```
1
```

Tutti gli oggetti visti finora sono immutabili. Cioè non possono essere modificati

Nelle prossime due righe di codice, sto assegnando alla x prima l'oggetto 2 e poi l'oggetto 3. Al termine del codice il nome x sarà assegnato all'oggetto 3. Dire che sto modificando la x non è proprio corretto

In [24]:
```
x = 2
x = 3
```

In [25]:
```
print(x)
```

```
3
```

In questo codice assegno alla y lo stesso valore di x, poi alla y assegno un alto oggetto.

In [26]:
```
y = x
y = 4
```

Siccome "non sto modificando la y" la x resterà con il valore vecchio

In [27]:
```
print(x)
```

```
3
```

In [28]:
```
print(y)
```

```
4
```

Oggetti mutabili

Le liste sono oggetti mutabili (esistono metodi per modificarli)

Andiamo con ordine, partiamo con la stampa della lista [1,"a",3,1]

In [29]:
```
print([1,"a",3,1])
```

```
[1, 'a', 3, 1]
```

Vediamo il tipo

In [30]:
```
print(type([1,"a",3,1]))
```

```
<class 'list'>
```

Anche le liste hanno i metodi count e pos

In [31]:
```
print([1,"a",3,1].count(3))
```

```
1
```

In [32]:
```
print([1,"a",3,1].index(3))
```

```
2
```

Inoltre posso accedere ad un elemento specifico della tupla

In [33]:
```
print([1,"a",3,1][0])
```

```
1
```

Esistono però anche metodi per modificare una lista. In questi casi conviene assegnare un nome alla lista. Partiamo assegnanto alla lista [1,"a",3,1] il nome x

In [34]:
```
x = [1,"a",3,1]
```

Modifichiamo la lista aggiungendo in fondo il valore 3 con il metodo append

In [35]:
```
x.append(3)
```

Stampiamo x

In [36]:
```
print(x)
```

```
[1, 'a', 3, 1, 3]
```

Attenzione: i metodi visti precedentemente count e index restituivano un intero. Append modifica direttamente la lista, non restituisce niente! Proviamo a lanciare questo codice

In [37]:
```
y = x.append(5)
```

Cosa c'è in x e in y?

In [38]:
```
print(x)
```

```
[1, 'a', 3, 1, 3, 5]
```

In [39]:
```
print(y)
```

```
None
```

Vediamo il metodo sort per ordinare la lista

In [40]:
```
z = [3,5,1]
```

In [41]:
```
print(z)
```

```
[3, 5, 1]
```

In [42]:
```
z.sort()
```

In [43]:
```
print(z)
```

```
[1, 3, 5]
```

Se invece proviamo ad ordinare x otteniamo un errore

In [44]:
```
x.sort()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[44], line 1
----> 1 x.sort()

TypeError: '<' not supported between instances of 'str' and 'int'
```

Risulta possibile convertire altri tipi in liste tramite la funzione list

In [45]:
```
x = "nicola"
```

In [46]:
```
print(list(x))
```

```
['n', 'i', 'c', 'o', 'l', 'a']
```

In [47]:
```
x = "nicola,giovanni,alberto"
```

In [48]:
```
print(x.split(","))
```

```
['nicola', 'giovanni', 'alberto']
```

Un altro oggetto mutabile: l'insieme

Stampiamo l'insieme {1,"a",3,1}

In [49]:
```
print({1,"a",3,1})
```

```
{1, 'a', 3}
```

Facciamo attenzione al fatto che nell'insieme non possono esserci duplicati! Inoltre non è presente un concetto di ordine

Vediamo il tipo

In [50]:
```
type({1,"a",3,1})
```

Out[50]:
```
set
```

Agli insiemi posso applicae funzioni e metodi, ma non quelli che riguardano la posizione! Attenzione al risultato

In [51]:
```
print(len({1,"a",3,1}))
```

```
3
```

I metodo count e index danno errore, perché non sono applicabili a insiemi

In [52]:
```
print({1,"a",3,1}.count(1))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[52], line 1
----> 1 print({1,"a",3,1}.count(1))

AttributeError: 'set' object has no attribute 'count'
```

In [53]:
```
print({1,"a",3,1}.index(3))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[53], line 1
----> 1 print({1,"a",3,1}.index(3))

AttributeError: 'set' object has no attribute 'index'
```

Allo stesso modo non accedere ad un elemento specifico dell'insieme

In [54]:
```
print({1,"a",3,1}[0])
```

```
<>:1: SyntaxWarning: 'set' object is not subscriptable; perhaps you missed a comma?
<>:1: SyntaxWarning: 'set' object is not subscriptable; perhaps you missed a comma?
C:\Users\ianto\AppData\Local\Temp\ipykernel_11596\567347472.py:1: SyntaxWarning: 'set' object is not subscriptable; perhaps
you missed a comma?
  print({1,"a",3,1}[0])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[54], line 1
----> 1 print({1,"a",3,1}[0])

TypeError: 'set' object is not subscriptable
```

E nemmeno ordinare

In [55]:
```
x = {1,3,5}
```

In [56]:
```
x.sort()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[56], line 1
----> 1 x.sort()

AttributeError: 'set' object has no attribute 'sort'
```

Posso invece aggiungere un elemento all'insieme. Assegniamo anche in questo caso un nome all'insieme

In [57]:
```
x = {1,"a",3,1}
```

In [58]:
```
x.add(4)
```

In [59]:
```
print(x)
```

```
{1, 'a', 3, 4}
```

Proviamo ad aggiungere un elemento già presente

In [60]:
```
x.add(1)
```

In [61]:
```
print(x)
```

```
{1, 'a', 3, 4}
```

Valgono le stesse considerazioni fatte per le liste

In [62]:
```
x = {1,"a",3,1}
y = x
y.add(4)
print(x)
print(y)
```

```
{1, 'a', 3, 4}
{1, 'a', 3, 4}
```

In [63]:
```
x = {1,"a",3,1}
y = x.add(4)
print(x)
print(y)
```

```
{1, 'a', 3, 4}
None
```

Gli insiemi hanno metodo interessanti non disponibili per le liste

In [64]:
```
x = {1,"a",3}
y = {1,2}
```

In [65]:
```
print(x.union(y))
```

```
{1, 2, 3, 'a'}
```

In [66]:
```
print(x.intersection(y))
```

```
{1}
```

In [67]:
```
print(x.difference(y))
```

```
{'a', 3}
```

Un altro oggetto mutabile: il dizionario

Stampiamo il dizionario {"nome":"Nicola","eta":35}

In [68]:
```
print({"nome":"Nicola","eta":35})
```

```
{'nome': 'Nicola', 'eta': 35}
```

I dizionari sono composti da coppie di chiave-valore. Nel dizionario non possono esserci due elementi con la stessa chiave. Se provo a creare un dizionario con la stessa chiave ripetuta due volte, sarà presa l'ultima coppia.

In [69]:
```
print({"nome":"Nicola","eta":35, "nome":"Giovanni"})
```

```
{'nome': 'Giovanni', 'eta': 35}
```

Dalla versione Python 3.7 i dizionari sono ordinati, nel senso che le chiavi seguono l'ordine di inserimento

Vediamo il tipo

In [70]:
```
type({"nome":"Nicola","eta":35})
```

Out[70]:
```
dict
```

Agli insiemi posso applicare funzioni e metodi. Attenzione al risultato

In [71]:
```
print(len({"nome":"Nicola","eta":35, "nome":"Giovanni"}))
```

```
2
```

Visualizziamo l'elenco di chiavi, valori e coppie chiave-valore

In [72]:
```
print({"nome":"Nicola","eta":35}.keys())
```

```
dict_keys(['nome', 'eta'])
```

In [73]:
```
print({"nome":"Nicola","eta":35}.values())
```

```
dict_values(['Nicola', 35])
```

In [74]:
```
print({"nome":"Nicola","eta":35}.items())
```

```
dict_items([('nome', 'Nicola'), ('eta', 35)])
```

I metodo count e index danno errore, perché non sono applicabili a insiemi

In [75]:
```
print({"nome":"Nicola","eta":35, "nome":"Giovanni"}.count("nome"))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[75], line 1
----> 1 print({"nome":"Nicola","eta":35, "nome":"Giovanni"}.count("nome"))

AttributeError: 'dict' object has no attribute 'count'
```

In [76]:
```
print({"nome":"Nicola","eta":35, "nome":"Giovanni"}.index(3))
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[76], line 1
----> 1 print({"nome":"Nicola","eta":35, "nome":"Giovanni"}.index(3))

AttributeError: 'dict' object has no attribute 'index'
```

Risulta facile accedere al valore di una determinata chiave

In [77]:
```
x = {"nome":"Nicola","eta":35}
```

In [78]:
```
print(x["nome"])
```

```
Nicola
```

Invece, analogamente agli insiemi, non posso accedere ad un elemento specifico del dizionario

In [79]:
```
print({"nome":"Nicola","eta":35}[0])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[79], line 1
----> 1 print({"nome":"Nicola","eta":35}[0])

KeyError: 0
```

Tuttavia da Python 3.7 (in cui è mantenuto l'ordine di inserimento) potrei arrivare al risultato utilizzando le liste

In [80]:
```
x = {"nome":"Nicola","eta":35}
```

In [81]:
```
chiavi = list(x.keys())
print(chiavi)
```

```
['nome', 'eta']
```

In [82]:
```
valori = list(x.values())
print(valori)
```

```
['Nicola', 35]
```

In [83]:
```
print({chiavi[0]:valori[0]})
```

```
{'nome': 'Nicola'}
```

Posso invece aggiungere una coppia al dizionario.

In [84]:
```
x = {"nome":"Nicola","eta":35}
```

In [85]:
```
x["cognome"] = "Iantomasi"
```

In [86]:
```
print(x)
```

```
{'nome': 'Nicola', 'eta': 35, 'cognome': 'Iantomasi'}
```

Proviamo ad aggiungere un elemento già presente

In [87]:
```
x["cognome"] = "Rossi"
```

Sovrascriverò il valore

In [88]:
```
print(x)
```

```
{'nome': 'Nicola', 'eta': 35, 'cognome': 'Rossi'}
```

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
path = r"C:\Users\ianto\Desktop\Corso Python\file"
```

Importare dati da un file .csv

Importiamo un file csv classico

In [3]:
```
clienti = pd.read_csv(filepath_or_buffer = path + r"\Clienti.csv",
            sep = ";",  #separtore del file, la virgola è il default
            header = 0, #per indicare quale riga contiene contiene l'intestazione,
                        #0 vuol dire che l'intestazione è nella prima riga
                        #se non c'è intestazione utilizzare None
            #names = ["NumeroCliente","IsActive","Nome","Cognome","DataNascita"]
            # se non avessi l'intestazione assegnerei i nome con l'attributo names
            #usecols = ["NumeroCliente","Nome","DataNascita","Regione"],  #colonne che vogliamo importare
            #dtype = {"NumeroCliente":np.str}, #tipi delle colonne
            #parse_dates=["DataNascita"], #colonne da importare come date
            #dayfirst=True, #MOLTO IMPORTANTE, per indicare che nelle date il giorno è indicato prima del mese
            #decimal=".", #separatore dei decimali, il default è il punto
            #index_col = "NumeroCliente", #impostare una colonna come indice
            #nrows = 10 #numero di righe da importare,
            #skiprows = 0, #per saltare alcune righe all'inizio del file
            )
```

Guardiamo le prime 5 righe

In [4]:
```
clienti.head(5)
```

Out[4]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
0	1	Nicoletta	01/01/2010	Francia	NaN
1	2	Giovanni	01/03/1976	Italia	Lazio
2	3	Marco	01/04/1980	Italia	Lazio
3	4	Giovanna	01/05/1977	Italia	Lazio
4	5	Alice	01/06/1969	Italia	Sicilia

In [5]:
```
#NOTA INTERESSANTE: altro modo di comporre il percorso del file
import os
path = r"C:\Users\ianto\Desktop\Corso python\file"
os.path.join(path,"Clienti.csv")
```

Out[5]:
```
'C:\\Users\\ianto\\Desktop\\Corso python\\file\\Clienti.csv'
```

In [6]:
```
#Per visualizzare tutte le righe e tutte le colonne
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
```

Ricavare informazioni sul dataframe

In [7]:
```
type(clienti)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [8]:
```
clienti.dtypes
```

Out[8]:
```
NumeroCliente     int64
Nome             object
DataNascita      object
Nazione          object
Regione          object
dtype: object
```

In [9]:
```
clienti.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype 
---  ------         --------------  ----- 
 0   NumeroCliente  40 non-null     int64 
 1   Nome           40 non-null     object
 2   DataNascita    40 non-null     object
 3   Nazione        40 non-null     object
 4   Regione        39 non-null     object
dtypes: int64(1), object(4)
memory usage: 1.7+ KB
```

In [10]:
```
clienti.columns
```

Out[10]:
```
Index(['NumeroCliente', 'Nome', 'DataNascita', 'Nazione', 'Regione'], dtype='object')
```

In [11]:
```
clienti.shape
```

Out[11]:
```
(40, 5)
```

In [12]:
```
clienti.index
```

Out[12]:
```
RangeIndex(start=0, stop=40, step=1)
```

Perché i type sono importanti

La prossima istruzione genera un errore perché il metodo year può essere applicato solo a colonne di tipo datetime

In [13]:
```
clienti["DataNascita"].dt.year
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[13], line 1
----> 1 clienti[             ].dt.year

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\generic.py:6318, in NDFrame.__getattr__(self, name)
   6311 if (
   6312     name not in self._internal_names_set
   6313     and name not in self._metadata
   6314     and name not in self._accessors
   6315     and self._info_axis._can_hold_identifiers_and_holds_name(name)
   6316 ):
   6317     return self[name]
-> 6318 return object.__getattribute__(self, name)

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\accessor.py:224, in CachedAccessor.__get__(self, obj, cls)
    221 if obj is None:
    222     # we're accessing the attribute of the class, i.e., Dataset.geo
    223     return self._accessor
--> 224 accessor_obj = self._accessor(obj)
    225 # Replace the property with the accessor object. Inspired by:
    226 # https://www.pydanny.com/cached-property.html
    227 # We need to use object.__setattr__ because we overwrite __setattr__ on
    228 # NDFrame
    229 object.__setattr__(obj, self._name, accessor_obj)

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\indexes\accessors.py:643, in CombinedDatetimelikeProperties.__
new__(cls, data)
    640 elif isinstance(data.dtype, PeriodDtype):
    641     return PeriodProperties(data, orig)
--> 643 raise AttributeError("Can only use .dt accessor with datetimelike values")

AttributeError: Can only use .dt accessor with datetimelike values
```

Occorre prima convertira la colonna nel tipo date

In [14]:
```
clienti["DataNascita"] = pd.to_datetime(clienti["DataNascita"],format="%d/%m/%Y")
```

In [15]:
```
clienti.head(5)
```

Out[15]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
0	1	Nicoletta	2010-01-01	Francia	NaN
1	2	Giovanni	1976-03-01	Italia	Lazio
2	3	Marco	1980-04-01	Italia	Lazio
3	4	Giovanna	1977-05-01	Italia	Lazio
4	5	Alice	1969-06-01	Italia	Sicilia

In [16]:
```
clienti.dtypes
```

Out[16]:
```
NumeroCliente             int64
Nome                     object
DataNascita      datetime64[ns]
Nazione                  object
Regione                  object
dtype: object
```

In [17]:
```
clienti["DataNascita"].dt.year[0:5]
```

Out[17]:
```
0    2010
1    1976
2    1980
3    1977
4    1969
Name: DataNascita, dtype: int32
```

Importare file da excel

Importiamo un file xlsx

In [18]:
```
from_excel = pd.read_excel(io = path + r"\FattureDettaglio.xlsx",
                           sheet_name = 'Tabelle1',
                            usecols = 'A:D,F',
                            header = 0,
                            dtype = {"IdProdotto":str}
                           )
```

In [19]:
```
type(from_excel)
```

Out[19]:
```
pandas.core.frame.DataFrame
```

In [20]:
```
from_excel.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 904 entries, 0 to 903
Data columns (total 5 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   IdFattura       904 non-null    int64
 1   IdProdotto      904 non-null    object
 2   PrezzoUnitario  904 non-null    float64
 3   Quantita        904 non-null    int64
 4   Omaggio         903 non-null    float64
dtypes: float64(2), int64(2), object(1)
memory usage: 35.4+ KB
```

In [21]:
```
from_excel.head(5)
```

Out[21]:

	IdFattura	IdProdotto	PrezzoUnitario	Quantita	Omaggio
0	1	04	28.8	28	0.0
1	1	13	38.0	24	0.0
2	2	01	7.3	18	0.0
3	2	03	14.4	4	0.0
4	2	09	12.0	43	0.0

Importare file JSON

Importiamo un file Json

In [22]:
```
clienti_j = pd.read_json(path_or_buf = path + r"\Clienti.json")
clienti_j.head(5)
```

Out[22]:

	IdCliente	IsActive	Nome	Cognome	DataNascita	Nazione	Cap	Telefono	Email	Regione	Fax
0	1	True	Nicola	5CF71	2010-01-01	Francia	81622	39 320 3231	Nicola.5CF71@gmail.com	NaN	NaN
1	2	True	Giovanni	A83C2	1976-03-01	Italia	82786	328 32312	Giovanni.A83C2tiscali.com	Lazio	328 32312
2	3	True	Marco	7929A	1980-04-01	Italia	19341	+(39) 327 38312	Marco.7929A@gmail.com	Lazio	+(39) 327 38312
3	4	True	Giovanna	270BC	1977-05-01	Italia	64791	39 320 22312	Giovanna.270BCtiscali.com	Lazio	NaN
4	5	True	Alice	C5B4D	1969-06-01	Italia	99172	320 7231	Alice.C5B4D@gmail.com	Sicilia	320 7231

Importiamo un file Json con una struttura più complessa

In [23]:
```
clienti2_j = pd.read_json(path_or_buf = path + r"\Clienti2.json")
clienti2_j.head(5)
```

Out[23]:

	Cognome	DataNascita	IdCliente	IsActive	Residenza	Contatti	Nome
0	Rossi	1972-05-01	40	true	{'Nazione': 'Italia', 'Regione': 'Molise'}	[39 320 3231, 39 320 1123]	NaN
1	NaN	1980-05-01	41	true	{'Nazione': 'Italia', 'Regione': 'Lombardia'}	[39 320 3199, 39 320 8833]	Nicola

In [24]:
```
clienti2_j.dtypes
```

Out[24]:
```
Cognome        object
DataNascita    object
IdCliente       int64
IsActive       object
Residenza      object
Contatti       object
Nome           object
dtype: object
```

Utilizziamo il metodo json_normalize

In [25]:
```
import json as j

with open( path + r"\Clienti2.json", "r") as f:
    data_json = j.load(f)
clienti2_j = pd.json_normalize(data_json)
```

In [26]:
```
clienti2_j.head(5)
```

Out[26]:

	Cognome	DataNascita	IdCliente	IsActive	Contatti	Residenza.Nazione	Residenza.Regione	Nome
0	Rossi	1972-05-01	40	true	[39 320 3231, 39 320 1123]	Italia	Molise	NaN
1	NaN	1980-05-01	41	true	[39 320 3199, 39 320 8833]	Italia	Lombardia	Nicola

In [27]:
```
type(data_json[0]["Residenza"]["Nazione"])
```

Out[27]:
```
str
```

OSSERVAZIONE: con Python potremmo anche lavorare direttamente data_json

In [28]:
```
data_json
```

Out[28]:
```
[{'Cognome': 'Rossi',
  'DataNascita': '1972-05-01',
  'IdCliente': 40,
  'IsActive': 'true',
  'Residenza': {'Nazione': 'Italia', 'Regione': 'Molise'},
  'Contatti': ['39 320 3231', '39 320 1123']},
 {'Nome': 'Nicola',
  'DataNascita': '1980-05-01',
  'IdCliente': 41,
  'IsActive': 'true',
  'Residenza': {'Nazione': 'Italia', 'Regione': 'Lombardia'},
  'Contatti': ['39 320 3199', '39 320 8833']}]
```

In [29]:
```
type(data_json[0])
```

Out[29]:
```
dict
```

Importiamo un altro file csv più complesso

In [30]:
```
fatture = pd.read_csv(filepath_or_buffer = path + r"\Fatture.csv",
            sep = ";",  #separtore del file
            header = None,
            names = ["NumeroFattura","Tipologia","Importo","Iva","IdCliente","ResidenzaCliente","DataFattura","NumeroForni
            decimal = ","
            )
```

In [31]:
```
fatture.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 8 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   NumeroFattura     18 non-null     int64  
 1   Tipologia         18 non-null     object 
 2   Importo           18 non-null     float64
 3   Iva               17 non-null     float64
 4   IdCliente         18 non-null     int64  
 5   ResidenzaCliente  18 non-null     object 
 6   DataFattura       18 non-null     object 
 7   NumeroFornitore   16 non-null     float64
dtypes: float64(3), int64(2), object(3)
memory usage: 1.3+ KB
```

Creiamo una nuova colonna DataFattura2 convertendo DataFattura con il metodo astype (errato!)

In [32]:
```
fatture['DataFattura2'] =  fatture['DataFattura'].astype("datetime64[ns]")
```

La conversione è errata, la data 01/03/2017 è diventata 2017-01-03

In [33]:
```
fatture.head(5)
```

Out[33]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	ResidenzaCliente	DataFattura	NumeroFornitore	DataFattura2
0	1	A	1120.0	20.0	1	Molise	01/01/2018	1.0	2018-01-01
1	2	V	32.0	20.0	2	Puglia	01/03/2017	1.0	2017-01-03
2	3	A	45.0	20.0	3	Lombardia	01/06/2017	1.0	2017-01-06
3	4	V	64.0	20.0	3	Lombardia	30/01/2019	1.0	2019-01-30
4	5	A	12.0	20.0	5	Umbria	01/01/2018	1.0	2018-01-01

Dobbiamo specificare il formato di partenza tramite l'argomento format del metodo di pandas to_datetime

In [34]:
```
fatture['DataFattura2'] =  pd.to_datetime(fatture['DataFattura'],
                           format="%d/%m/%Y",
                           )
```

In [35]:
```
fatture.head(5)
```

Out[35]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	ResidenzaCliente	DataFattura	NumeroFornitore	DataFattura2
0	1	A	1120.0	20.0	1	Molise	01/01/2018	1.0	2018-01-01
1	2	V	32.0	20.0	2	Puglia	01/03/2017	1.0	2017-03-01
2	3	A	45.0	20.0	3	Lombardia	01/06/2017	1.0	2017-06-01
3	4	V	64.0	20.0	3	Lombardia	30/01/2019	1.0	2019-01-30
4	5	A	12.0	20.0	5	Umbria	01/01/2018	1.0	2018-01-01

ATTENZIONE! Se avessi specificato DattaFattura all'interno del parametro parse_dates avrei ottenuto comunque una conversione errata!

Andiamo avanti. La prossima conversione fallisce perché non posso convertire in int64 una colonna che contiene dei null

In [36]:
```
fatture['NumeroFornitore'] = fatture['NumeroFornitore'].astype('int64')
```

```
---------------------------------------------------------------------------
IntCastingNaNError                        Traceback (most recent call last)
Cell In[36], line 1
----> 1 fatture['NumeroFornitore'] = fatture[                  ].astype(       )

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\generic.py:6662, in NDFrame.astype(self, dtype, copy, errors)
   6656     results = [
   6657         ser.astype(dtype, copy=copy, errors=errors) for _, ser in self.items()
   6658     ]
   6660 else:
   6661     # else, only a single dtype is given
-> 6662     new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   6663     res = self._constructor_from_mgr(new_data, axes=new_data.axes)
   6664     return res.__finalize__(self, method="astype")

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\internals\managers.py:430, in BaseBlockManager.astype(self, dt
ype, copy, errors)
    427 elif using_copy_on_write():
    428     copy = False
--> 430 return self.apply(
    431             ,
    432     dtype=dtype,
    433     copy=copy,
    434     errors=errors,
    435     using_cow=using_copy_on_write(),
    436 )

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\internals\managers.py:363, in BaseBlockManager.apply(self, f,
align_keys, **kwargs)
    361         applied = b.apply(f, **kwargs)
    362     else:
--> 363         applied = getattr(b, f)(**kwargs)
    364     result_blocks = extend_blocks(applied, result_blocks)
    366 out = type(self).from_blocks(result_blocks, self.axes)

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\internals\blocks.py:784, in Block.astype(self, dtype, copy, er
rors, using_cow, squeeze)
    781         raise ValueError("Can not squeeze with more than one column.")
    782     values = values[0, :]  # type: ignore[call-overload]
--> 784 new_values = astype_array_safe(values, dtype, copy=copy, errors=errors)
    786 new_values = maybe_coerce_values(new_values)
    788 refs = None

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\dtypes\astype.py:237, in astype_array_safe(values, dtype, cop
y, errors)
    234     dtype = dtype.numpy_dtype
    236 try:
--> 237     new_values = astype_array(values, dtype, copy=copy)
    238 except (ValueError, TypeError):
    239     # e.g. _astype_nansafe can fail on object-dtype of strings
    240     #  trying to convert to float
    241     if errors == "ignore":

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\dtypes\astype.py:182, in astype_array(values, dtype, copy)
    179     values = values.astype(dtype, copy=copy)
    181 else:
--> 182     values = _astype_nansafe(values, dtype, copy=copy)
    184 # in pandas we don't store numpy str dtypes, so convert to object
    185 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type, str):

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\dtypes\astype.py:101, in _astype_nansafe(arr, dtype, copy, ski
pna)
     96     return lib.ensure_string_array(
     97         arr, skipna=skipna, convert_na_value=False
     98     ).reshape(shape)
    100 elif np.issubdtype(arr.dtype, np.floating) and dtype.kind in "iu":
--> 101     return _astype_float_to_int_nansafe(arr, dtype, copy)
    103 elif arr.dtype == object:
    104     # if we have a datetime/timedelta array of objects
    105     # then coerce to datetime64[ns] and use DatetimeArray.astype
    107     if lib.is_np_dtype(dtype, "M"):

File ~\Desktop\ambiente_python\Lib\site-packages\pandas\core\dtypes\astype.py:145, in _astype_float_to_int_nansafe(values,
dtype, copy)
    141 """
    142 astype with a check preventing converting NaN to an meaningless integer value.
    143 """
    144 if not np.isfinite(values).all():
--> 145     raise IntCastingNaNError(
    146         "Cannot convert non-finite values (NA or inf) to integer"
    147     )
    148 if dtype.kind == "u":
    149     # GH#45151
    150     if not (values >= 0).all():

IntCastingNaNError: Cannot convert non-finite values (NA or inf) to integer
```

Per acquisire i null come intero devo usare il tipo Int64

In [38]:
```
fatture['NumeroFornitore'] = fatture['NumeroFornitore'].astype('Int64')
```

Oppure sostituire prima i null con un altro valore

In [39]:
```
fatture['Iva'] = fatture['Iva'].fillna(0).astype(np.int64)
```

In [40]:
```
fatture.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 18 entries, 0 to 17
Data columns (total 9 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   NumeroFattura     18 non-null     int64
 1   Tipologia         18 non-null     object
 2   Importo           18 non-null     float64
 3   Iva               18 non-null     int64
 4   IdCliente         18 non-null     int64
 5   ResidenzaCliente  18 non-null     object
 6   DataFattura       18 non-null     object
 7   NumeroFornitore   18 non-null     Int64
 8   DataFattura2      18 non-null     datetime64[ns]
dtypes: Int64(1), datetime64[ns](1), float64(1), int64(3), object(3)
memory usage: 1.4+ KB
```

In [41]:
```
fatture.head(5)
```

Out[41]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	ResidenzaCliente	DataFattura	NumeroFornitore	DataFattura2
0	1	A	1120.0	20	1	Molise	01/01/2018	1	2018-01-01
1	2	V	32.0	20	2	Puglia	01/03/2017	1	2017-03-01
2	3	A	45.0	20	3	Lombardia	01/06/2017	1	2017-06-01
3	4	V	64.0	20	3	Lombardia	30/01/2019	1	2019-01-30
4	5	A	12.0	20	5	Umbria	01/01/2018	1	2018-01-01

Esercizio: importiamo un file excel contenente più fogli

valorizziamo l'argomento sheet_name con None

In [54]:
```
data = pd.read_excel(path + r"\Dati_su_piu_fogli.xlsx",
                     sheet_name=["Fatture","Clienti"])
```

In [55]:
```
type(data)
```

Out[55]:
```
dict
```

In [56]:
```
data.keys()
```

Out[56]:
```
dict_keys(['Fatture', 'Clienti'])
```

In [57]:
```
data["Fatture"].head(5)
```

Out[57]:

	IdFattura	Tipologia	Importo	Iva	IdCliente	IdFornitore
0	1	A	40	20.0	1	1.0
1	2	V	32	20.0	2	1.0
2	3	A	45	20.0	3	1.0
3	4	V	64	20.0	3	1.0
4	5	A	12	20.0	5	1.0

Un dizionario è un insieme di coppie chiave-valore. Visualizziamo l'elenco delle chiavi

Visualizziamo il valore della chiave Fatture

In [58]:
```
data["Clienti"].head(5)
```

Out[58]:

	NumeroCliente	IsActive	Nome	Cognome	DataNascita	Nazione	Regione
0	1	True	Nicoletta	5CF71	2010-01-01	Francia	NaN
1	2	True	Giovanni	A83C2	1976-03-01	Italia	Lazio
2	3	True	Marco	7929A	1980-04-01	Italia	Lazio
3	4	True	Giovanna	270BC	1977-05-01	Italia	Lazio
4	5	True	Alice	C5B4D	1969-06-01	Italia	Sicilia

posso fare un ciclo sulle chiavi

In [59]:
```
for chiave in data.keys():
    numero_righe = len(data[chiave])
    print(f'il foglio {chiave} ha {numero_righe} righe')
```

```
il foglio Fatture ha 18 righe
il foglio Clienti ha 39 righe
```

Posso rimuovere chiavi da un dizionario

Api: application programming interface

In [60]:
```
import requests
```

Creiamo una variabile con l'url per la chiamata api

In [61]:
```
url = "https://api.github.com/repos/iantomasinicola/PortfolioDataAnalyst"
```

Utilizziamo il metodo get della libreria requests

In [62]:
```
import requests
res = requests.get("https://api.github.com/repos/iantomasinicola/CorsoPython")
res
```

Out[62]:
```
<Response [200]>
```

Controlliamo che lo status_code sia 200

In [63]:
```
res.status_code
```

Out[63]:
```
200
```

Visualizziamo il contenuto della risposta: i dati sono difficilmente lavorabili

In [64]:
```
res.content[0:500]
```

Out[64]:
```
b'{"id":985746558,"node_id":"R_kgDOOsFMfg","name":"CorsoPython","full_name":"iantomasinicola/CorsoPython","private":fals
e,"owner":{"login":"iantomasinicola","id":59792312,"node_id":"MDQ6VXNlcjU5NzkyMzEy","avatar_url":"https://avatars.githubu
sercontent.com/u/59792312?v=4","gravatar_id":"","url":"https://api.github.com/users/iantomasinicola","html_url":"https://
github.com/iantomasinicola","followers_url":"https://api.github.com/users/iantomasinicola/followers","following_url":"htt
ps://api.github.com'
```

Anche in formato testo la situazione è difficilmente gestibile

In [65]:
```
res.text[0:500]
```

Out[65]:
```
'{"id":985746558,"node_id":"R_kgDOOsFMfg","name":"CorsoPython","full_name":"iantomasinicola/CorsoPython","private":fals
e,"owner":{"login":"iantomasinicola","id":59792312,"node_id":"MDQ6VXNlcjU5NzkyMzEy","avatar_url":"https://avatars.githubu
sercontent.com/u/59792312?v=4","gravatar_id":"","url":"https://api.github.com/users/iantomasinicola","html_url":"https://
github.com/iantomasinicola","followers_url":"https://api.github.com/users/iantomasinicola/followers","following_url":"htt
ps://api.github.com'
```

Invece convertendo i dati in json, posso utilizzare le funzionalità dei dizionari di Python!

In [66]:
```
res.json()
```

Out[66]:

```
{'id': 985746558,
 'node_id': 'R_kgDOOsFMfg',
 'name': 'CorsoPython',
 'full_name': 'iantomasinicola/CorsoPython',
 'private': False,
 'owner': {'login': 'iantomasinicola',
  'id': 59792312,
  'node_id': 'MDQ6VXNlcjU5NzkyMzEy',
  'avatar_url': 'https://avatars.githubusercontent.com/u/59792312?v=4',
  'gravatar_id': '',
  'url': 'https://api.github.com/users/iantomasinicola',
  'html_url': 'https://github.com/iantomasinicola',
  'followers_url': 'https://api.github.com/users/iantomasinicola/followers',
  'following_url': 'https://api.github.com/users/iantomasinicola/following{/other_user}',
  'gists_url': 'https://api.github.com/users/iantomasinicola/gists{/gist_id}',
  'starred_url': 'https://api.github.com/users/iantomasinicola/starred{/owner}{/repo}',
  'subscriptions_url': 'https://api.github.com/users/iantomasinicola/subscriptions',
  'organizations_url': 'https://api.github.com/users/iantomasinicola/orgs',
  'repos_url': 'https://api.github.com/users/iantomasinicola/repos',
  'events_url': 'https://api.github.com/users/iantomasinicola/events{/privacy}',
  'received_events_url': 'https://api.github.com/users/iantomasinicola/received_events',
  'type': 'User',
  'user_view_type': 'public',
  'site_admin': False},
 'html_url': 'https://github.com/iantomasinicola/CorsoPython',
 'description': None,
 'fork': False,
 'url': 'https://api.github.com/repos/iantomasinicola/CorsoPython',
 'forks_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/forks',
 'keys_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/keys{/key_id}',
 'collaborators_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/collaborators{/collaborator}',
 'teams_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/teams',
 'hooks_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/hooks',
 'issue_events_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/issues/events{/number}',
 'events_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/events',
 'assignees_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/assignees{/user}',
 'branches_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/branches{/branch}',
 'tags_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/tags',
 'blobs_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/git/blobs{/sha}',
 'git_tags_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/git/tags{/sha}',
 'git_refs_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/git/refs{/sha}',
 'trees_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/git/trees{/sha}',
 'statuses_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/statuses/{sha}',
 'languages_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/languages',
 'stargazers_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/stargazers',
 'contributors_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/contributors',
 'subscribers_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/subscribers',
 'subscription_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/subscription',
 'commits_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/commits{/sha}',
 'git_commits_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/git/commits{/sha}',
 'comments_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/comments{/number}',
 'issue_comment_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/issues/comments{/number}',
 'contents_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/contents/{+path}',
 'compare_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/compare/{base}...{head}',
 'merges_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/merges',
 'archive_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/{archive_format}{/ref}',
 'downloads_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/downloads',
 'issues_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/issues{/number}',
 'pulls_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/pulls{/number}',
 'milestones_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/milestones{/number}',
 'notifications_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/notifications{?since,all,participating}',
 'labels_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/labels{/name}',
 'releases_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/releases{/id}',
 'deployments_url': 'https://api.github.com/repos/iantomasinicola/CorsoPython/deployments',
 'created_at': '2025-05-18T12:50:43Z',
 'updated_at': '2025-05-31T14:18:35Z',
 'pushed_at': '2025-05-31T14:18:32Z',
 'git_url': 'git://github.com/iantomasinicola/CorsoPython.git',
 'ssh_url': 'git@github.com:iantomasinicola/CorsoPython.git',
 'clone_url': 'https://github.com/iantomasinicola/CorsoPython.git',
 'svn_url': 'https://github.com/iantomasinicola/CorsoPython',
 'homepage': None,
 'size': 2932,
 'stargazers_count': 0,
 'watchers_count': 0,
 'language': 'Jupyter Notebook',
 'has_issues': True,
 'has_projects': True,
 'has_downloads': True,
 'has_wiki': True,
 'has_pages': False,
 'has_discussions': False,
 'forks_count': 0,
 'mirror_url': None,
 'archived': False,
```

```
 'disabled': False,
 'open_issues_count': 0,
 'license': None,
 'allow_forking': True,
 'is_template': False,
 'web_commit_signoff_required': False,
 'topics': [],
 'visibility': 'public',
 'forks': 0,
 'open_issues': 0,
 'watchers': 0,
 'default_branch': 'main',
 'temp_clone_token': None,
 'network_count': 0,
 'subscribers_count': 1}
```

In [67]:
```
json_res = res.json()
```

In [68]:
```
type(json_res)
```

Out[68]:
```
dict
```

Accedo a dei valori specifici del dizionario

In [69]:
```
owner = json_res["owner"]["avatar_url"]
owner
```

Out[69]:
```
'https://avatars.githubusercontent.com/u/59792312?v=4'
```

In [70]:
```
url_home = json_res["owner"]["html_url"]
url_home
```

Out[70]:
```
'https://github.com/iantomasinicola'
```

In [71]:
```
topics = json_res["topics"]
topics
```

Out[71]:
```
[]
```

In [72]:
```
type(topics)
```

Out[72]:
```
list
```

In [73]:
```
print(json_res["description"])
```

```
None
```

In [74]:
```
lista = [[json_res["name"], json_res["owner"]["html_url"],json_res["topics"]]]
```

In [75]:
```
lista
```

Out[75]:
```
[['CorsoPython', 'https://github.com/iantomasinicola', []]]
```

In [76]:
```
pd.DataFrame(columns=["name","owner","topics"], data= lista)
```

Out[76]:

	name	owner	topics
0	CorsoPython	https://github.com/iantomasinicola	[]

Vediamo come possiamo creare un DataFrame

In [77]:
```
nuovo_dict = {"name": json_res["name"],
              "stato": json_res["owner"]["html_url"],
              "topics":json_res["topics"]}
```

In [78]:
```
nuovo_dict
```

Out[78]:
```
{'name': 'CorsoPython',
 'stato': 'https://github.com/iantomasinicola',
 'topics': []}
```

In [79]:
```
pd.DataFrame(nuovo_dict)
```

Out[79]:

	name	stato	topics

Analisi dichiarative

In [1]:
```
import pandas as pd
import numpy as np
import datetime
path = r"C:\Users\ianto\Desktop\Corso Python\file"
```

Creiamo una funzione per importare due dei file studiati nella lezione precedente

In [2]:
```
def ImportFile():

    clienti = pd.read_csv(filepath_or_buffer = path + r"\Clienti.csv",
            sep = ";",
            header = 0
            )

    clienti["DataNascita"] =  pd.to_datetime(clienti["DataNascita"])

    fatture = pd.read_csv(filepath_or_buffer = path + r"\Fatture.csv",
            sep = ";",  #separtore del file
            header = None,
            names = ["NumeroFattura","Tipologia","Importo","Iva","IdCliente","Regione","DataFattura","NumeroFornitore"],
            decimal = ",",
            )

    fatture['DataFattura'] =  pd.to_datetime(fatture['DataFattura'],
                                             format="%d/%m/%Y"
                                            )

    fatture['NumeroFornitore'] = fatture['NumeroFornitore'].astype('Int64')

    return clienti,fatture
```

In [3]:
```
clienti,fatture = ImportFile()
```

In [4]:
```
fatture.head(5)
```

Out[4]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
0	1	A	1120.0	20.0	1	Molise	2018-01-01	1
1	2	V	32.0	20.0	2	Puglia	2017-03-01	1
2	3	A	45.0	20.0	3	Lombardia	2017-06-01	1
3	4	V	64.0	20.0	3	Lombardia	2019-01-30	1
4	5	A	12.0	20.0	5	Umbria	2018-01-01	1

In [5]:
```
clienti.head(5)
```

Out[5]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
0	1	Nicoletta	2010-01-01	Francia	NaN
1	2	Giovanni	1976-01-03	Italia	Lazio
2	3	Marco	1980-01-04	Italia	Lazio
3	4	Giovanna	1977-01-05	Italia	Lazio
4	5	Alice	1969-01-06	Italia	Sicilia

In [6]:
```
#Per visualizzare tutte le righe e tutte le colonne
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows',None)
```

Selezionare e filtrare i dati

Estrarre le colonne NumeroClienti, Nome e regione dal dataframe Clienti. Visualizzare solo 5 righe

In [7]:
```
#SELECT TOP 5 NumeroCliente, Nome, Regione
#FROM   Clienti

clienti[["NumeroCliente", "Nome", "Regione"]].head(5)
```

Out[7]:

	NumeroCliente	Nome	Regione
0	1	Nicoletta	NaN
1	2	Giovanni	Lazio
2	3	Marco	Lazio
3	4	Giovanna	Lazio
4	5	Alice	Sicilia

Indicando solo il nome del dataframe vedrò tutte le colonne

In [8]:
```
#SELECT TOP 5 *
#FROM   Clienti

clienti.head(5)
```

Out[8]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
0	1	Nicoletta	2010-01-01	Francia	NaN
1	2	Giovanni	1976-01-03	Italia	Lazio
2	3	Marco	1980-01-04	Italia	Lazio
3	4	Giovanna	1977-01-05	Italia	Lazio
4	5	Alice	1969-01-06	Italia	Sicilia

Estrarre tutte le informazioni dei clienti della regione Lazio

In [9]:
```
#SELECT *
#FROM   Clienti
#WHERE  Regione='Lazio'

clienti.query('Regione == "Lazio"')
```

Out[9]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
1	2	Giovanni	1976-01-03	Italia	Lazio
2	3	Marco	1980-01-04	Italia	Lazio
3	4	Giovanna	1977-01-05	Italia	Lazio

In [10]:
```
#vecchio metodo
clienti[clienti["Regione"] == "Lazio"]
```

Out[10]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
1	2	Giovanni	1976-01-03	Italia	Lazio
2	3	Marco	1980-01-04	Italia	Lazio
3	4	Giovanna	1977-01-05	Italia	Lazio

Filtri con AND

Estrarre NumeroCliente e Nome dei clienti della regione Lazio che si chiamano Giovanni

In [11]:
```
#SELECT NumeroCliente,
#       Nome
#FROM   Clienti
#WHERE  Regione='Lazio'
#   AND Nome='Giovanni
clienti.query('Regione == "Lazio" and Nome == "Giovanni"') [["NumeroCliente", "Nome"]]
```

Out[11]:

	NumeroCliente	Nome
1	2	Giovanni

In [12]:
```
#vecchio metodo
clienti[(clienti["Regione"] == "Lazio") & (clienti["Nome"] == "Giovanni")] \
       [["NumeroCliente", "Nome"]]
```

Out[12]:

	NumeroCliente	Nome
1	2	Giovanni

Possiamo anche dichiarare prima il valore da cercare

In [13]:
```
x = "Lazio"
y = "Giovanni"
```

In [14]:
```
clienti.query('Regione == @x and Nome == @y') [["NumeroCliente", "Nome"]]
```

Out[14]:

	NumeroCliente	Nome
1	2	Giovanni

Filtri con OR

Estrarre NumeroCliente, Nome e regione dei clienti che soddisfano almeno una delle seguenti proprietà:

1. sono della regione Lazio
2. si chiamano Giovanni

In [15]:
```
#SELECT TOP 5 NumeroCliente,
#       Nome,
#       Regione
#FROM   Clienti
#WHERE  Regione='Lazio'
#    OR Nome='Giovanni

clienti.query('Regione == "Lazio" or Nome == "Giovanni"') [["NumeroCliente", "Nome", "Regione"]].head(5)
```

Out[15]:

	NumeroCliente	Nome	Regione
1	2	Giovanni	Lazio
2	3	Marco	Lazio
3	4	Giovanna	Lazio
9	10	Giovanni	Toscana
14	15	Giovanni	Toscana

Filtri con isin

Estrarre NumeroCliente, Nome e regione dei clienti residenti nel Lazio o nel Piemonte

In [16]:
```
#SELECT NumeroCliente,
#       Nome,
#       Regione
#FROM   Clienti
#WHERE  Regione = 'Piemonte'
#    OR Regione='Lazio'
#oppure
#SELECT NumeroCliente,
#       Nome,
#       Regione
#FROM   Clienti
#WHERE  Regione IN ('Piemonte', 'Lazio')

clienti.query('Regione in ["Piemonte","Lazio"]')[["NumeroCliente", "Nome","Regione"]]
```

Out[16]:

	NumeroCliente	Nome	Regione
1	2	Giovanni	Lazio
2	3	Marco	Lazio
3	4	Giovanna	Lazio
19	20	Giovanni	Piemonte
20	22	Franca	Piemonte
21	23	Maria	Piemonte
22	24	Marina	Piemonte

Attenzione ai tipi!

Se cerco una parola in una colonna di tipo intero, non otterrò risultati

In [17]:
```
clienti.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 40 entries, 0 to 39
Data columns (total 5 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   NumeroCliente  40 non-null     int64
 1   Nome           40 non-null     object
 2   DataNascita    40 non-null     datetime64[ns]
 3   Nazione        40 non-null     object
 4   Regione        39 non-null     object
dtypes: datetime64[ns](1), int64(1), object(3)
memory usage: 1.7+ KB
```

In [18]:
```
clienti.query("NumeroCliente == '3'")
```

Out[18]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione

Proviamo a convertire la colonna in stringa in un nuovo DataFrame

In [19]:
```
clienti2 = clienti.copy()
clienti2["NumeroCliente"] = clienti2["NumeroCliente"].astype(str)
```

In [20]:
```
clienti2.query("NumeroCliente == '3'")
```

Out[20]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
2	3	Marco	1980-01-04	Italia	Lazio

Viceversa non otterrò risultati con la prossima query

In [21]:
```
clienti2.query("NumeroCliente == 3 ")
```

Out[21]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione

Attenzione ai null!

Null risulta diverso da Lazio (non accade lo stesso con l'SQL)

In [22]:
```
clienti.query("Regione != 'Lazio'").head(5)
```

Out[22]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
0	1	Nicoletta	2010-01-01	Francia	NaN
4	5	Alice	1969-01-06	Italia	Sicilia
5	6	Fabrizio	1996-01-07	Italia	Sicilia
6	7	Irene	1990-01-08	Italia	Sicilia
7	8	Maria	1999-01-09	Italia	Sicilia

estrarre tutte le righe dove la regione è null

In [23]:
```
#SELECT *
#FROM   Clienti
#WHERE  Regione IS NULL

clienti.query("Regione.isna()")
```

Out[23]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
0	1	Nicoletta	2010-01-01	Francia	NaN

Estrarre 5 righe dove la regione non è null

In [24]:
```
#SELECT TOP 5 *
#FROM   Clienti
#WHERE  Regione IS NOT NULL

clienti.query("Regione.notna()").head(5)
```

Out[24]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
1	2	Giovanni	1976-01-03	Italia	Lazio
2	3	Marco	1980-01-04	Italia	Lazio
3	4	Giovanna	1977-01-05	Italia	Lazio
4	5	Alice	1969-01-06	Italia	Sicilia
5	6	Fabrizio	1996-01-07	Italia	Sicilia

Filtri su colonne di tipo datetime

Estrarre tutte le fatture emesse dopo il 3 marzo 2018

In [25]:
```
fatture.dtypes
```

Out[25]:
```
NumeroFattura              int64
Tipologia                 object
Importo                  float64
Iva                      float64
IdCliente                  int64
Regione                   object
DataFattura        datetime64[ns]
NumeroFornitore             Int64
dtype: object
```

In [26]:
```
#SQL
#SELECT *
#FROM   Fatture
#WHERE  DataFattura > '2018-03-01'

fatture.query("DataFattura > '2018-03-01'")
```

Out[26]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
3	4	V	64.0	20.0	3	Lombardia	2019-01-30	1
7	8	V	54.0	20.0	8	Abruzzo	2019-01-30	2
11	12	A	57.0	20.0	7	Marche	2019-01-30	<NA>
15	16	V	21.0	20.0	1	Molise	2019-02-05	3
16	17	V	1.0	20.0	5	Umbria	2019-01-05	4

Vediamo come utilizzare una variabile

In [27]:
```
from datetime import datetime
data = datetime(2018,3,1)
fatture.query("DataFattura > @data")
```

Out[27]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
3	4	V	64.0	20.0	3	Lombardia	2019-01-30	1
7	8	V	54.0	20.0	8	Abruzzo	2019-01-30	2
11	12	A	57.0	20.0	7	Marche	2019-01-30	<NA>
15	16	V	21.0	20.0	1	Molise	2019-02-05	3
16	17	V	1.0	20.0	5	Umbria	2019-01-05	4

Filtri su funzioni applicate a colonne

Estrarre tutte le fatture del 2018

In [28]:
```
#SELECT *
#FROM   Fatture
#WHERE  YEAR(DataFattura) = 2018

fatture.query("DataFattura.dt.year == 2018")
```

Out[28]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
0	1	A	1120.0	20.0	1	Molise	2018-01-01	1
4	5	A	12.0	20.0	5	Umbria	2018-01-01	1
8	9	A	67.0	20.0	3	Lombardia	2018-01-01	2
17	18	V	2.0	20.0	4	Piemonte	2018-03-01	5

Estrarre tutti i clienti di nome Nicola, facendo una ricerca non case sensitive

In [29]:
```
clienti.query("Nome.str.upper() == 'NICOLA'")
```

Out[29]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
16	17	Nicola	1982-01-06	Italia	Toscana

Aggregare i dati

Contare il numero di righe

Contare il numero di righe del dataframe Fatture

In [30]:
```
#SELECT COUNT(*)
#FROM   Fatture

len(fatture)
```

Out[30]:
```
18
```

In [31]:
```
#oppure
fatture.shape[0]
```

Out[31]:
```
18
```

Calcolare indici statistici su tutte le colonne

Calcolare la somma di tutte le colonne

In [32]:
```
fatture.sum(numeric_only=True)
```

Out[32]:
```
NumeroFattura       171.0
Importo            1723.0
Iva                 342.0
IdCliente            68.0
NumeroFornitore      35.0
dtype: Float64
```

Se volessi solo il dato di importo scriverei

In [33]:
```
fatture.sum(numeric_only=True)["Importo"]
```

Out[33]:
```
np.float64(1723.0)
```

Calcolare i principali indici statistici di tutte le colonne

In [34]:
```
fatture.describe()
```

Out[34]:

	NumeroFattura	Importo	Iva	IdCliente	DataFattura	NumeroFornitore
count	18.000000	18.000000	17.000000	18.000000	18	16.0
mean	9.500000	95.722222	20.117647	3.777778	2017-11-28 21:20:00	2.1875
min	1.000000	1.000000	20.000000	1.000000	2016-01-03 00:00:00	1.0
25%	5.250000	14.250000	20.000000	2.000000	2017-06-01 00:00:00	1.0
50%	9.500000	33.000000	20.000000	3.000000	2017-11-10 12:00:00	2.0
75%	13.750000	56.250000	20.000000	5.000000	2018-10-19 12:00:00	3.0
max	18.000000	1120.000000	22.000000	8.000000	2019-02-05 00:00:00	5.0
std	5.338539	256.775893	0.485071	2.289504	NaN	1.167262

Calcolare correlazioni tra colonne numeriche di un DataFrame

In [35]:
```
fatture.corr(numeric_only=True)
```

Out[35]:

	NumeroFattura	Importo	Iva	IdCliente	NumeroFornitore
NumeroFattura	1.000000	-0.420769	0.233741	0.043314	0.930877
Importo	-0.420769	1.000000	-0.085201	-0.294584	-0.312398
Iva	0.233741	-0.085201	1.000000	0.006740	0.201945
IdCliente	0.043314	-0.294584	0.006740	1.000000	0.074812
NumeroFornitore	0.930877	-0.312398	0.201945	0.074812	1.000000

Aggregare su una colonna

Calcolare il numero di Fatture per ogni fornitore

In [36]:
```
#SELECT   NumeroFornitore, count(*)
#FROM     Fatture
#GROUP BY NumeroFornitore;

fatture.groupby(by="NumeroFornitore",
                as_index=False,
                dropna=False
               )["NumeroFattura"].size()
```

Out[36]:

	NumeroFornitore	size
0	1	5
1	2	6
2	3	3
3	4	1
4	5	1
5	<NA>	2

Se eventualmente voglio cambiare il nome della colonna con il conteggio posso usare il metodo rename

In [37]:
```
fatture.groupby(by="NumeroFornitore",
                as_index=False,
                dropna=False)["NumeroFattura"].size().rename(columns={"size":"numero_totale"})
```

Out[37]:

	NumeroFornitore	numero_totale
0	1	5
1	2	6
2	3	3
3	4	1
4	5	1
5	<NA>	2

Altro metodo

In [38]:
```
#SELECT   NumeroFornitore, count(*)
#FROM     Fatture
#GROUP BY NumeroFornitore;

fatture.groupby(by="NumeroFornitore",
                as_index=False,
                dropna=False) \
       .agg(conteggio    = ("NumeroFattura",
                            np.size)
            )
```

Out[38]:

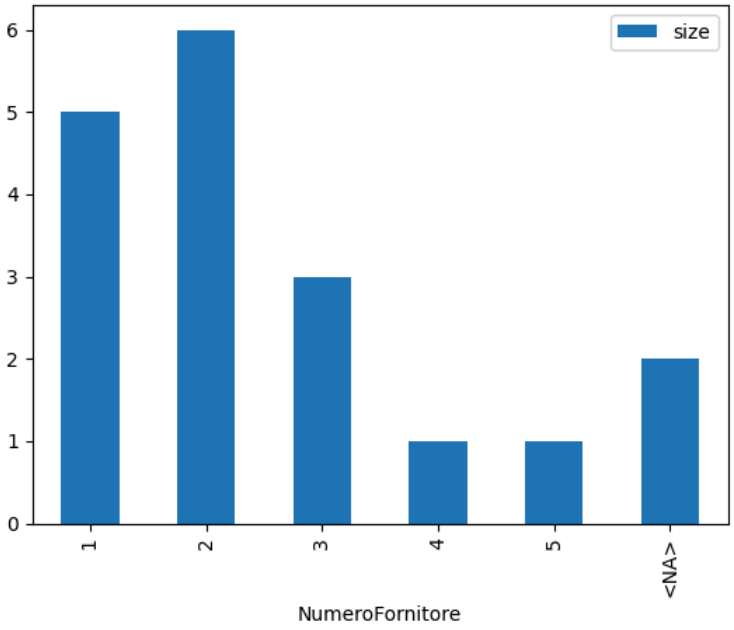
	NumeroFornitore	conteggio
0	1	5
1	2	6
2	3	3
3	4	1
4	5	1
5	<NA>	2

I dati raggruppati si presentano ad essere rappresentati graficamente

In [39]:
```
grafico = fatture.groupby(by="NumeroFornitore",
                          as_index=False,
                          dropna=False).size()

grafico.plot(x = 'NumeroFornitore',
             y = 'size',
             kind = 'bar')
```

Out[39]: <Axes: xlabel='NumeroFornitore'>



Altri esempi di aggregazione

Somma degli importi per ogni fornitore

In [40]:
```
fatture.groupby(by=["NumeroFornitore"],
                as_index=False,
                dropna=False)[["Importo"]].sum()
```

Out[40]:

	NumeroFornitore	Importo
0	1	1273.0
1	2	236.0
2	3	67.0
3	4	1.0
4	5	2.0
5	<NA>	144.0

Media di importi e iva per ogni fornitore

In [41]:
```
fatture.groupby(by=["NumeroFornitore"],
                as_index=False,
                dropna=False)[["Importo","Iva"]].mean().rename(columns={"Importo":"Importo_medio"})
```

Out[41]:

	NumeroFornitore	Importo_medio	Iva
0	1	254.600000	20.0
1	2	39.333333	20.0
2	3	22.333333	21.0
3	4	1.000000	20.0
4	5	2.000000	20.0
5	<NA>	72.000000	20.0

Somma di importo e media di iva per ogni fornitore e cliente

In [42]:
```
fatture.groupby(by=["NumeroFornitore","IdCliente"], as_index=False, dropna=False).\
        agg({"Importo":"sum","Iva":"mean"}).\
        rename(columns={"Importo":"Somma importo", "Iva":"Media iva"}).head(10)
```

Out[42]:

	NumeroFornitore	IdCliente	Somma importo	Media iva
0	1	1	1120.0	20.0
1	1	2	32.0	20.0
2	1	3	109.0	20.0
3	1	5	12.0	20.0
4	2	2	21.0	20.0
5	2	3	79.0	20.0
6	2	6	31.0	20.0
7	2	8	105.0	20.0
8	3	1	55.0	20.0
9	3	4	12.0	22.0

Numero di clienti univoci

In [43]:
```
fatture["IdCliente"].nunique()
```

Out[43]: 8

Numero di clienti univoci al variare del fornitore

In [44]:
```
fatture.groupby(by=["NumeroFornitore"],
                as_index=False, dropna=False)["IdCliente"].nunique()
```

Out[44]:

	NumeroFornitore	IdCliente
0	1	4
1	2	4
2	3	2
3	4	1
4	5	1
5	<NA>	2

Numero di regioni presenti nel Dataframe dei clienti

In [45]:
```
clienti["Regione"].nunique()
```

Out[45]: 7

Elenco di regioni univoche

In [46]:
```
clienti[["Regione"]].drop_duplicates()
```

Out[46]:

	Regione
0	NaN
1	Lazio
4	Sicilia
9	Toscana
19	Piemonte
23	Lombardia
28	Puglia
33	Molise

Combinare dataframe differenti

Riportare in un solo dataframe tutte le colonne dei dataframe Fatture e Clienti.

In [47]:
```
fatture.head(1)
```

Out[47]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
0	1	A	1120.0	20.0	1	Molise	2018-01-01	1

In [48]:
```
#SELECT     TOP 5 *
#FROM       Fatture
#INNER JOIN Clienti
# ON Fatture.IdCliente = Clienti.NumeroCliente

f2 = pd.merge(fatture,
         clienti,
         how = 'inner',
         left_on = "IdCliente",
         right_on = "NumeroCliente",
         suffixes = ('_fatture', '_clienti'))
```

In [49]:
```
f2.head(1)
```

Out[49]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione_fatture	DataFattura	NumeroFornitore	NumeroCliente	Nome	Da
0	1	A	1120.0	20.0	1	Molise	2018-01-01	1	1	Nicoletta	2(

Esercizio: estrarre NumeroCliente e nome dei clienti senza fatture

In [50]:
```
#SQL
#SELECT    TOP 5 Clienti.NumeroCliente, Clienti.Nome
#FROM      Clienti
#LEFT JOIN Fatture
#       ON Clienti.NumeroCliente = Fatture.IdCliente
#WHERE F.IdCliente IS NULL

pd.merge(clienti,
         fatture,
         how = 'left',
         left_on = "NumeroCliente",
         right_on = "IdCliente").query("IdCliente.isna()")[["NumeroCliente","Nome"]].head(5)
```

Out[50]:

	NumeroCliente	Nome
18	9	Grazie
19	10	Giovanni
20	11	Maria
21	12	Giuseppe
22	13	Francesco

Ordinare un dataframe

Ordinare un dataframe per una colonna

Visualizzare le fatture dalla più recente alla meno recente

In [51]:
```
#SQL
#SELECT TOP 5 *
#FROM   Fatture
#ORDER BY DataFattura DESC

fatture.sort_values(by=['Importo'], ascending=False).head(5)
```

Out[51]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
0	1	A	1120.0	20.0	1	Molise	2018-01-01	1
12	13	V	87.0	20.0	2	Puglia	2016-01-03	<NA>
8	9	A	67.0	20.0	3	Lombardia	2018-01-01	2
3	4	V	64.0	20.0	3	Lombardia	2019-01-30	1
11	12	A	57.0	20.0	7	Marche	2019-01-30	<NA>

Creare un nuovo dataframe con il nuovo ordine

In [52]:
```
fatture_new = fatture.sort_values(by=['DataFattura'], ascending=False).copy()
fatture_new.head(5)
```

Out[52]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
15	16	V	21.0	20.0	1	Molise	2019-02-05	3
7	8	V	54.0	20.0	8	Abruzzo	2019-01-30	2
3	4	V	64.0	20.0	3	Lombardia	2019-01-30	1
11	12	A	57.0	20.0	7	Marche	2019-01-30	<NA>
16	17	V	1.0	20.0	5	Umbria	2019-01-05	4

Resettiamo l'indice di FattureNew

In [53]:
```
fatture_new = fatture_new.reset_index(drop=True)
fatture_new.head(5)
```

Out[53]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
0	16	V	21.0	20.0	1	Molise	2019-02-05	3
1	8	V	54.0	20.0	8	Abruzzo	2019-01-30	2
2	4	V	64.0	20.0	3	Lombardia	2019-01-30	1
3	12	A	57.0	20.0	7	Marche	2019-01-30	<NA>
4	17	V	1.0	20.0	5	Umbria	2019-01-05	4

Ordinare un dataframe in maniera randomica

Visualizzare il dataframe Fatture con un ordinamento randomico

In [54]:
```
fatture.sample(frac=1,
               random_state=0 #per rendere l'ordinamento ripetibile)
              ).head(5)
```

Out[54]:

	NumeroFattura	Tipologia	Importo	Iva	IdCliente	Regione	DataFattura	NumeroFornitore
1	2	V	32.0	20.0	2	Puglia	2017-03-01	1
6	7	A	12.0	20.0	3	Lombardia	2017-06-01	2
8	9	A	67.0	20.0	3	Lombardia	2018-01-01	2
10	11	A	21.0	20.0	2	Puglia	2017-06-01	2
14	15	A	34.0	NaN	1	Molise	2017-06-01	3

Modificare un dataframe

Copiare un dataframe

Copiare il dataframe Clienti in ClientiBis

In [55]:
```
#SQL
#SELECT *
#INTO   ClientiBis
#FROM   Clienti

clienti_bis = clienti.copy()
```

Convertire un dataframe in un array di numpy

Creare un array di numpy a partire dal dataframe clienti

In [56]:
```
array = clienti.values
```

In [57]:
```
type(array)
```

Out[57]:
```
numpy.ndarray
```

In [58]:
```
array[0:4]
```

Out[58]:
```
array([[1, 'Nicoletta', Timestamp('2010-01-01 00:00:00'), 'Francia', nan],
       [2, 'Giovanni', Timestamp('1976-01-03 00:00:00'), 'Italia',
        'Lazio'],
       [3, 'Marco', Timestamp('1980-01-04 00:00:00'), 'Italia', 'Lazio'],
       [4, 'Giovanna', Timestamp('1977-01-05 00:00:00'), 'Italia',
        'Lazio']], dtype=object)
```

Convertire un dataframe in una lista di liste

Convertire il dataframe Clienti in una lista

In [59]:
```
lista_clienti = clienti.values.tolist()
```

In [60]:
```
type(lista_clienti)
```

Out[60]:
```
list
```

In [61]:
```
lista_clienti[0:4]
```

Out[61]:
```
[[1, 'Nicoletta', Timestamp('2010-01-01 00:00:00'), 'Francia', nan],
 [2, 'Giovanni', Timestamp('1976-01-03 00:00:00'), 'Italia', 'Lazio'],
 [3, 'Marco', Timestamp('1980-01-04 00:00:00'), 'Italia', 'Lazio'],
 [4, 'Giovanna', Timestamp('1977-01-05 00:00:00'), 'Italia', 'Lazio']]
```

Creare un dataframe a partire da una lista

Creare un dataframe a partire da una lista, assegnando i nomi alle colonne

In [62]:
```
new_df = pd.DataFrame(data = lista_clienti,
                      columns = ['NumeroCliente', 'Nome', 'Cognome', 'DataNascita', 'Regione'] )
```

In [63]:
```
new_df.dtypes
```

Out[63]:
```
NumeroCliente            int64
Nome                    object
Cognome          datetime64[ns]
DataNascita             object
Regione                 object
dtype: object
```

Rinominare una colonna

Rinominare la colonna Iva del dataframe Fatture in Tax

In [64]:
```
fatture = fatture.rename(columns={"Iva": "Tax"})
fatture.head(3)
```

Out[64]:

	NumeroFattura	Tipologia	Importo	Tax	IdCliente	Regione	DataFattura	NumeroFornitore
0	1	A	1120.0	20.0	1	Molise	2018-01-01	1
1	2	V	32.0	20.0	2	Puglia	2017-03-01	1
2	3	A	45.0	20.0	3	Lombardia	2017-06-01	1

Eliminare una colonna

Eliminare la colonna NumeroFornitore dal dataframe Fatture

In [65]:
```
fatture.columns
```

Out[65]:
```
Index(['NumeroFattura', 'Tipologia', 'Importo', 'Tax', 'IdCliente', 'Regione',
       'DataFattura', 'NumeroFornitore'],
      dtype='object')
```

In [66]:
```
#SQL
#ALTER TABLE Fatture
#DROP COLUMN NumeroFornitore

fatture = fatture.drop(["NumeroFornitore"], axis=1)
#fatture.head(3)
```

Aggiornare una colonna

Guardiamo i clienti della regione Lazio

In [67]:
```
clienti.query("Regione == 'Lazio'")
```

Out[67]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
1	2	Giovanni	1976-01-03	Italia	Lazio
2	3	Marco	1980-01-04	Italia	Lazio
3	4	Giovanna	1977-01-05	Italia	Lazio

Modificare il nome di tutti i clienti della regione Lazio valorizzandolo con "Nicola"

In [68]:
```
#SQL
#UPDATE Clienti
#SET    Nome = 'Nicola'
#WHERE  Regione = 'Lazio'

clienti.loc[clienti["Regione"] == "Lazio", "Nome"] = 'Nicola'
```

In [69]:
```
clienti.query("Regione == 'Lazio'")
```

Out[69]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
1	2	Nicola	1976-01-03	Italia	Lazio
2	3	Nicola	1980-01-04	Italia	Lazio
3	4	Nicola	1977-01-05	Italia	Lazio

In [70]:
```
#oppure

clienti["Nome"] = np.where(clienti["Regione"] == "Lazio",
                           "Nicola",
                           clienti["Nome"]  )
```

Aggiornare una colonna sostituendo i null con un valore

Sostituire i null presenti nella colonna Regione del dataframe Cliente con la stringa "Non conosciuta"

In [71]:
```
#SQL
#UPDATE Clienti
#SET    Regione = COALESCE(Regione,'Non conosciuta')
#WHERE  Regione IS NULL

clienti["Regione"] = clienti["Regione"].fillna("Non conosciuta")
```

In [72]:
```
clienti.head(3)
```

Out[72]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione
0	1	Nicoletta	2010-01-01	Francia	Non conosciuta
1	2	Nicola	1976-01-03	Italia	Lazio
2	3	Nicola	1980-01-04	Italia	Lazio

Concatenare una colonna

Concatenare le colonne Nazione e Regione del dataframe clienti.

Attenzione, in presenza di un null il risultato della concatenazione sarà null. Per evitare che ciò accada, il metodo fillna può essere usato per sostituire i null con ''.

In [73]:
```
clienti["ColonnaConcatenata"] = clienti["Nazione"].fillna("") + "," + clienti["Regione"].fillna("")
clienti.head(3)
```

Out[73]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione	ColonnaConcatenata
0	1	Nicoletta	2010-01-01	Francia	Non conosciuta	Francia,Non conosciuta
1	2	Nicola	1976-01-03	Italia	Lazio	Italia,Lazio
2	3	Nicola	1980-01-04	Italia	Lazio	Italia,Lazio

Splittare una colonna

Splittare tramite la virgola la colonna appena creata in due colonne: ColonnaSplit1 e ColonnaSplit2

In [74]:
```
clienti[["ColonnaSplit1","ColonnaSplit2"]] = clienti["ColonnaConcatenata"].str.split(pat = ',', expand=True)
clienti.head(3)
```

Out[74]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione	ColonnaConcatenata	ColonnaSplit1	ColonnaSplit2
0	1	Nicoletta	2010-01-01	Francia	Non conosciuta	Francia,Non conosciuta	Francia	Non conosciuta
1	2	Nicola	1976-01-03	Italia	Lazio	Italia,Lazio	Italia	Lazio
2	3	Nicola	1980-01-04	Italia	Lazio	Italia,Lazio	Italia	Lazio

Creare una colonna in base ad una condizione

Creare nel dataframe Fatture la colonna TipologiaPrezzo contenente "Prezzo alto" se il valore della colonna Importo è maggiore di 50, "Prezzo basso" altrimenti.

In [75]:
```
#SQL
#CASE WHEN Importo > 50
#     THEN 'Prezzo alto'
#     ELSE 'Prezzo basso'
#END

fatture["TipologiaPrezzo"] = np.where(fatture["Importo"] > 50, "Prezzo alto", "Prezzo basso")
fatture.head(5)
```

Out[75]:

	NumeroFattura	Tipologia	Importo	Tax	IdCliente	Regione	DataFattura	TipologiaPrezzo
0	1	A	1120.0	20.0	1	Molise	2018-01-01	Prezzo alto
1	2	V	32.0	20.0	2	Puglia	2017-03-01	Prezzo basso
2	3	A	45.0	20.0	3	Lombardia	2017-06-01	Prezzo basso
3	4	V	64.0	20.0	3	Lombardia	2019-01-30	Prezzo alto
4	5	A	12.0	20.0	5	Umbria	2018-01-01	Prezzo basso

Creare una colonna in base a più condizioni

Creare nel dataframe Fatture la colonna TipologiaPrezzo contenente:

- "Prezzo alto" se il valore della colonna Importo è maggiore di 50
- "Prezzo medio" se è compreso tra 30(escluso) e 50(incluso)
- "Prezzo basso" se minore o uguale a 30

In [76]:
```
fatture["TipologiaPrezzo2"] = np.select([fatture["Importo"] > 50,
                                         (fatture["Importo"] > 30) & (fatture["Importo"] <=50),
                                         fatture["Importo"] <= 30],
                                         ["Prezzo alto",
                                          "Prezzo medio",
                                          "Prezzo basso"],
```

```
                                        default='Non classificato')
fatture.head(5)
```

Out[76]:

	NumeroFattura	Tipologia	Importo	Tax	IdCliente	Regione	DataFattura	TipologiaPrezzo	TipologiaPrezzo2
0	1	A	1120.0	20.0	1	Molise	2018-01-01	Prezzo alto	Prezzo alto
1	2	V	32.0	20.0	2	Puglia	2017-03-01	Prezzo basso	Prezzo medio
2	3	A	45.0	20.0	3	Lombardia	2017-06-01	Prezzo basso	Prezzo medio
3	4	V	64.0	20.0	3	Lombardia	2019-01-30	Prezzo alto	Prezzo alto
4	5	A	12.0	20.0	5	Umbria	2018-01-01	Prezzo basso	Prezzo basso

Creare colonne con porzioni di stringhe di un'altra colonna

Creare una colonna con le iniziali di nome e cognome per ogni cliente

In [77]:
```
#SQL
#SUBSTRING(Nome,1,1)

clienti["Iniziale"] =  clienti["Nome"].str.slice(start=0, stop =1)
clienti.head(3)
```

Out[77]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione	ColonnaConcatenata	ColonnaSplit1	ColonnaSplit2	Iniziale
0	1	Nicoletta	2010-01-01	Francia	Non conosciuta	Francia,Non conosciuta	Francia	Non conosciuta	N
1	2	Nicola	1976-01-03	Italia	Lazio	Italia,Lazio	Italia	Lazio	N
2	3	Nicola	1980-01-04	Italia	Lazio	Italia,Lazio	Italia	Lazio	N

Visualizzare i dati senza duplicati

Il prossimo codice visualizza le righe del DataFrame clienti senza duplicati (considerando i valori in tutte le colonne).

ATTENZIONE! Per DataFrame con molte righe può essere un'operazione molto lunga

In [78]:
```
clienti.drop_duplicates().head(3)
```

Out[78]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione	ColonnaConcatenata	ColonnaSplit1	ColonnaSplit2	Iniziale
0	1	Nicoletta	2010-01-01	Francia	Non conosciuta	Francia,Non conosciuta	Francia	Non conosciuta	N
1	2	Nicola	1976-01-03	Italia	Lazio	Italia,Lazio	Italia	Lazio	N
2	3	Nicola	1980-01-04	Italia	Lazio	Italia,Lazio	Italia	Lazio	N

Più utile è utilizzare drop_duplicates per ottenere l'elenco di valori distinti in una colonna (o combinazione di colonne)

In [79]:
```
#Elenco delle regioni presenti: primo metodo

clienti[["Regione"]].drop_duplicates()
```

Out[79]:

	Regione
0	Non conosciuta
1	Lazio
4	Sicilia
9	Toscana
19	Piemonte
23	Lombardia
28	Puglia
33	Molise

In [80]:
```
#Elenco delle regioni presenti: secondo metodo

clienti.drop_duplicates(subset=["Regione"], keep = "first")
```

Out[80]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione	ColonnaConcatenata	ColonnaSplit1	ColonnaSplit2	Iniziale
0	1	Nicoletta	2010-01-01	Francia	Non conosciuta	Francia,Non conosciuta	Francia	Non conosciuta	N
1	2	Nicola	1976-01-03	Italia	Lazio	Italia,Lazio	Italia	Lazio	N
4	5	Alice	1969-01-06	Italia	Sicilia	Italia,Sicilia	Italia	Sicilia	A
9	10	Giovanni	1971-01-11	Italia	Toscana	Italia,Toscana	Italia	Toscana	G
19	20	Giovanni	1981-01-09	Italia	Piemonte	Italia,Piemonte	Italia	Piemonte	G
23	25	Cristina	1991-01-02	Italia	Lombardia	Italia,Lombardia	Italia	Lombardia	C
28	30	Seth	1967-01-07	Italia	Puglia	Italia,Puglia	Italia	Puglia	S
33	35	Giovanni	1974-01-12	Italia	Molise	Italia,Molise	Italia	Molise	G

In [81]:
```
#Proviamo a ordinare prima il DataFrame
```

In [82]:
```
clienti.sort_values(by="DataNascita",ascending=False).drop_duplicates(subset=["Regione"], keep = "first")
```

Out[82]:

	NumeroCliente	Nome	DataNascita	Nazione	Regione	ColonnaConcatenata	ColonnaSplit1	ColonnaSplit2	Iniziale
0	1	Nicoletta	2010-01-01	Francia	Non conosciuta	Francia,Non conosciuta	Francia	Non conosciuta	N
7	8	Maria	1999-01-09	Italia	Sicilia	Italia,Sicilia	Italia	Sicilia	M
12	13	Francesco	1999-01-02	Italia	Toscana	Italia,Toscana	Italia	Toscana	F
36	38	Mario	1994-01-03	Italia	Molise	Italia,Molise	Italia	Molise	M
23	25	Cristina	1991-01-02	Italia	Lombardia	Italia,Lombardia	Italia	Lombardia	C
22	24	Marina	1990-01-01	Italia	Piemonte	Italia,Piemonte	Italia	Piemonte	M
30	32	Maria	1985-01-09	Italia	Puglia	Italia,Puglia	Italia	Puglia	M
2	3	Nicola	1980-01-04	Italia	Lazio	Italia,Lazio	Italia	Lazio	N

Pivot e unpivot dei dati

Raggruppiamo i dati per cliente e tipologia

In [83]:
```
df = fatture.groupby(by=["IdCliente","Tipologia"], as_index=False, dropna=False)["Importo"].sum()

df
```

Out[83]:

	IdCliente	Tipologia	Importo
0	1	A	1154.0
1	1	V	21.0
2	2	A	21.0
3	2	V	119.0
4	3	A	124.0
5	3	V	64.0
6	4	V	14.0
7	5	A	12.0
8	5	V	1.0
9	6	V	31.0
10	7	A	57.0
11	8	A	51.0
12	8	V	54.0

Modifichiamo la forma dell'output, visualizziamo una per ogni cliente due colonne per gli importi in A e V

In [84]:
```
df_pivot = df.pivot(columns = "Tipologia",  #da quale colonna dell'input costruire le colonne dell'output
                    index = "IdCliente",    #quale sarà l'indice del nuovo dataframe
                    values = "Importo" #cosa riportare nelle righe
                    )
df_pivot
```

Out[84]:

Tipologia	A	V
IdCliente		
1	1154.0	21.0
2	21.0	119.0
3	124.0	64.0
4	NaN	14.0
5	12.0	1.0
6	NaN	31.0
7	57.0	NaN
8	51.0	54.0

L'IdCliente è l'indice del dataframe

In [85]:
```
df_pivot.index
```

Out[85]:
```
Index([1, 2, 3, 4, 5, 6, 7, 8], dtype='int64', name='IdCliente')
```

Rendiamolo una colonna

In [86]:
```
df_pivot.reset_index()
```

Out[86]:

	Tipologia	IdCliente	A	V
0	1	1154.0	21.0
1	2	21.0	119.0
2	3	124.0	64.0
3	4	NaN	14.0
4	5	12.0	1.0
5	6	NaN	31.0
6	7	57.0	NaN
7	8	51.0	54.0

eliminiamo il nome dell'indice

In [87]:
```
df_pivot = df_pivot.reset_index().rename_axis(None, axis=1)
df_pivot
```

Out[87]:

	IdCliente	A	V
0	1	1154.0	21.0
1	2	21.0	119.0
2	3	124.0	64.0
3	4	NaN	14.0
4	5	12.0	1.0
5	6	NaN	31.0
6	7	57.0	NaN
7	8	51.0	54.0

Torniamo alla visualizzazione con le colonne IdCliente e Tipologia con unpivot

In [88]:
```
df_unpivot = pd.melt(df_pivot,
        id_vars='IdCliente', #colonna da lasciare nell'outoput
        value_vars=list(df_pivot.columns).remove("IdCliente"), #colonne da trasformare in righe
        var_name='Tipologia', #nome della nuova colonna contenente le vecche colonne
        value_name='Importo' #nome della nuova colonna contenente i valori
                    )

df_unpivot
```

Out[88]:

	IdCliente	Tipologia	Importo
0	1	A	1154.0
1	2	A	21.0
2	3	A	124.0
3	4	A	NaN
4	5	A	12.0
5	6	A	NaN
6	7	A	57.0
7	8	A	51.0
8	1	V	21.0
9	2	V	119.0
10	3	V	64.0
11	4	V	14.0
12	5	V	1.0
13	6	V	31.0
14	7	V	NaN
15	8	V	54.0

Rappresentazioni grafiche

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Importiamo un dataset d'esempio sui dati del titanic con seaborn.

Citazione:Hind, Philip. Encyclopedia Titanica.

In [2]:
```
titanic=sns.load_dataset("titanic")
```

In [3]:
```
titanic.head(5)
```

Out[3]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [4]:
```
titanic.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype   
---  ------       --------------  -----   
 0   survived     891 non-null    int64   
 1   pclass       891 non-null    int64   
 2   sex          891 non-null    object  
 3   age          714 non-null    float64 
 4   sibsp        891 non-null    int64   
 5   parch        891 non-null    int64   
 6   fare         891 non-null    float64 
 7   embarked     889 non-null    object  
 8   class        891 non-null    category
 9   who          891 non-null    object  
 10  adult_male   891 non-null    bool    
 11  deck         203 non-null    category
 12  embark_town  889 non-null    object  
 13  alive        891 non-null    object  
 14  alone        891 non-null    bool    
dtypes: bool(2), category(2), float64(2), int64(4), object(5)
memory usage: 80.7+ KB
```

pclass-> classe del biglietto

sibsp -> numero di fratelli/sorelle/coniugi

parch -> numero genitori/figli

deck -> ponte

Otteniamo le informazioni di statistica monovariata di base

In [5]:
```
titanic.describe(include='all')
```

Out[5]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
count	891.000000	891.000000	891	714.000000	891.000000	891.000000	891.000000	889	891	891	891	203	889	891	891
unique	NaN	NaN	2	NaN	NaN	NaN	NaN	3	3	3	2	7	3	2	2
top	NaN	NaN	male	NaN	NaN	NaN	NaN	S	Third	man	True	C	Southampton	no	True
freq	NaN	NaN	577	NaN	NaN	NaN	NaN	644	491	537	537	59	644	549	537
mean	0.383838	2.308642	NaN	29.699118	0.523008	0.381594	32.204208	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
std	0.486592	0.836071	NaN	14.526497	1.102743	0.806057	49.693429	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
min	0.000000	1.000000	NaN	0.420000	0.000000	0.000000	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25%	0.000000	2.000000	NaN	20.125000	0.000000	0.000000	7.910400	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50%	0.000000	3.000000	NaN	28.000000	0.000000	0.000000	14.454200	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
75%	1.000000	3.000000	NaN	38.000000	1.000000	0.000000	31.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
max	1.000000	3.000000	NaN	80.000000	8.000000	6.000000	512.329200	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Otteniamo le informazioni di statistica bivariata di base

In [6]:
```
titanic.corr(numeric_only=True)
#titanic.corr(numeric_only=True, method='spearman')
```

Out[6]:

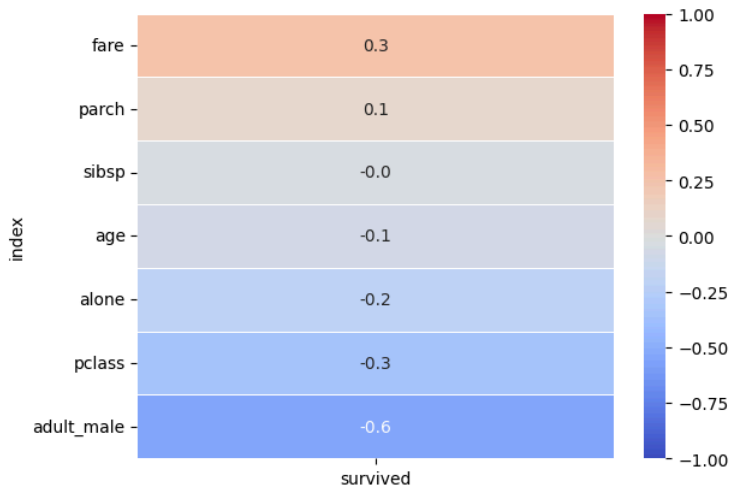
	survived	pclass	age	sibsp	parch	fare	adult_male	alone
survived	1.000000	-0.338481	-0.077221	-0.035322	0.081629	0.257307	-0.557080	-0.203367
pclass	-0.338481	1.000000	-0.369226	0.083081	0.018443	-0.549500	0.094035	0.135207
age	-0.077221	-0.369226	1.000000	-0.308247	-0.189119	0.096067	0.280328	0.198270
sibsp	-0.035322	0.083081	-0.308247	1.000000	0.414838	0.159651	-0.253586	-0.584471
parch	0.081629	0.018443	-0.189119	0.414838	1.000000	0.216225	-0.349943	-0.583398
fare	0.257307	-0.549500	0.096067	0.159651	0.216225	1.000000	-0.182024	-0.271832
adult_male	-0.557080	0.094035	0.280328	-0.253586	-0.349943	-0.182024	1.000000	0.404744
alone	-0.203367	0.135207	0.198270	-0.584471	-0.583398	-0.271832	0.404744	1.000000

In [7]:
```
titanic.corr(numeric_only=True)[["survived"]].sort_values(by="survived",ascending=False).\
        reset_index().query("index != 'survived'")
```

Out[7]:

	index	survived
1	fare	0.257307
2	parch	0.081629
3	sibsp	-0.035322
4	age	-0.077221
5	alone	-0.203367
6	pclass	-0.338481
7	adult_male	-0.557080

In [8]:
```
g = sns.heatmap(data = titanic.corr(numeric_only=True)[["survived"]].sort_values(by="survived",ascending=False).sort_values(by="survived",ascendi
                vmin=-1,
                vmax=1,
                annot=True,
                fmt=".1f",
                linewidth=.5,
                cmap='coolwarm')
```



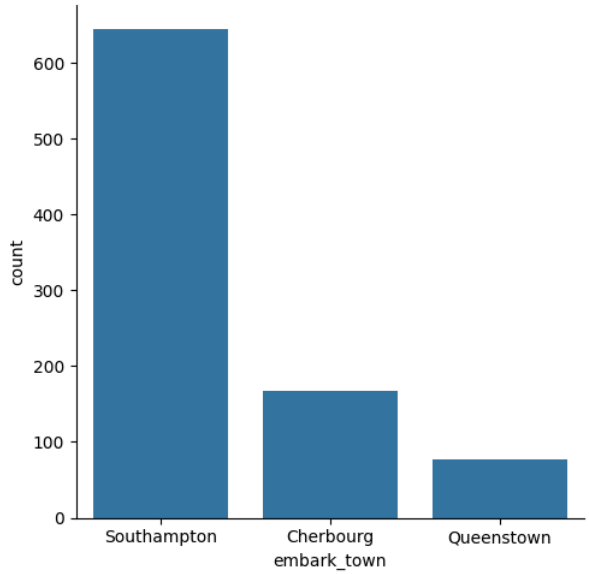
Rappresentiamo una variabile nominale

In [9]:
```
titanic.head(5)
```

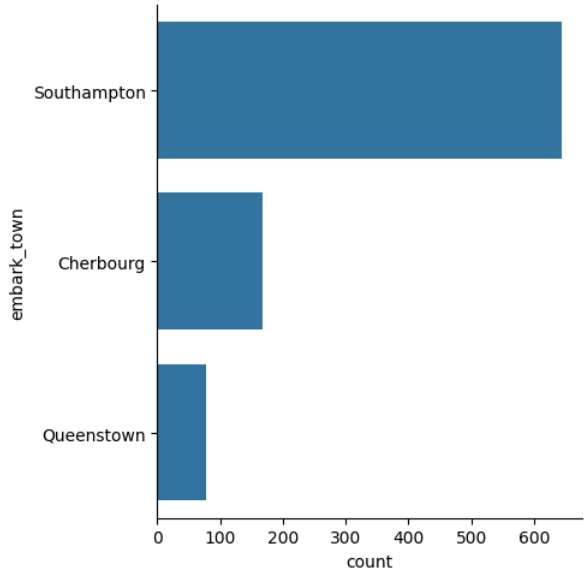
Out[9]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

In [10]:
```
g = sns.catplot(data=titanic, x="embark_town", kind="count")
```

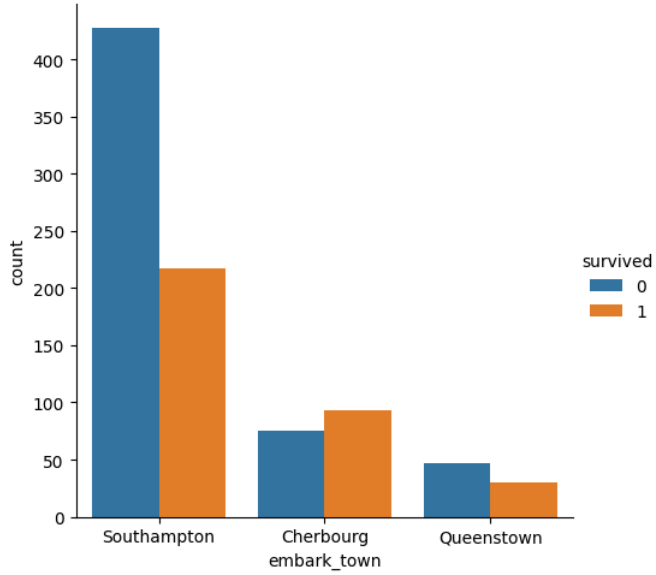


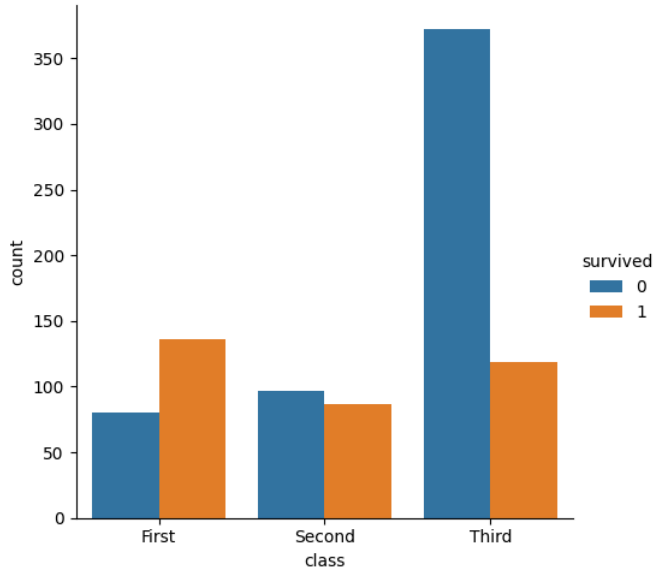
In [11]:
```
g = sns.catplot(data=titanic, y="embark_town", kind="count")
```

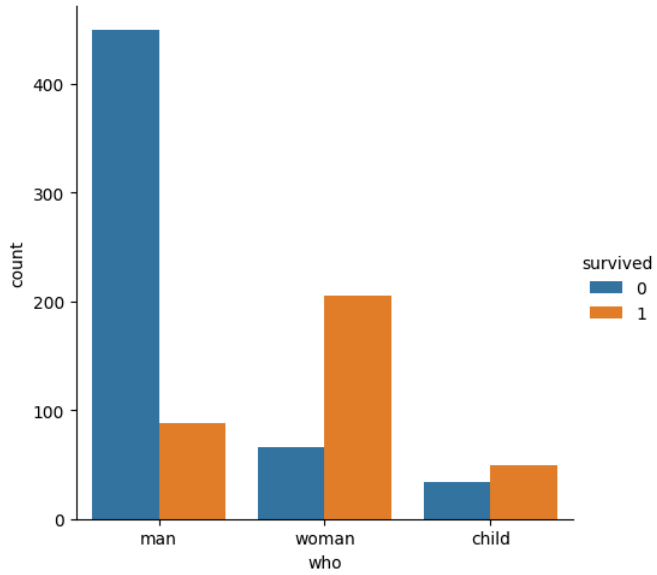


Rappresentiamo due variabili nominali

In [12]:
```
for colonna in ["embark_town","class","who"]:
    g = sns.catplot(
        data=titanic, x=colonna, hue="survived", kind="count"
        )
```



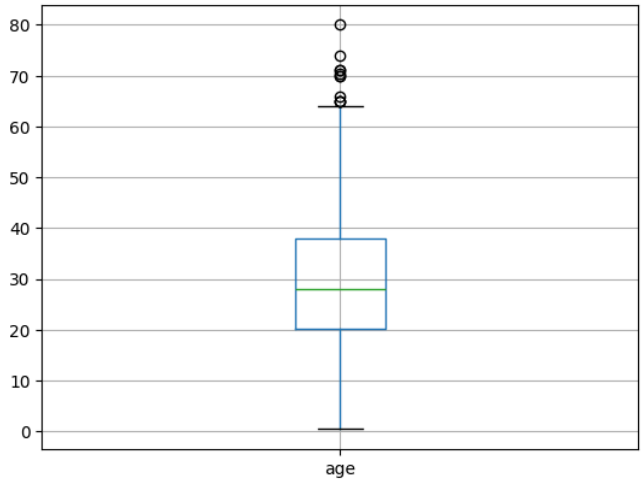




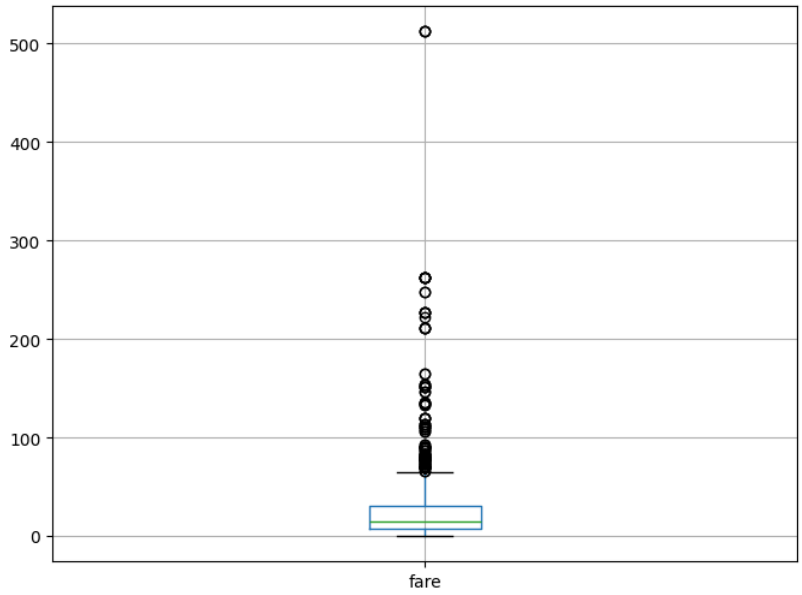
Rappresentiamo una variabile quantitativa

Boxplot

In [13]:
```
g = titanic.boxplot(column = "age")
```



In [14]:
```
g = titanic.boxplot(column = "fare", figsize = (8,6))
```



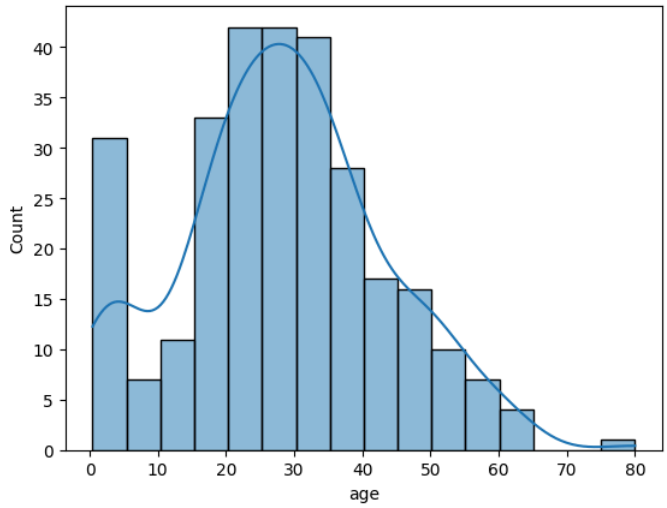
In [15]:
```
titanic.query("fare > 500")
```

Out[15]:

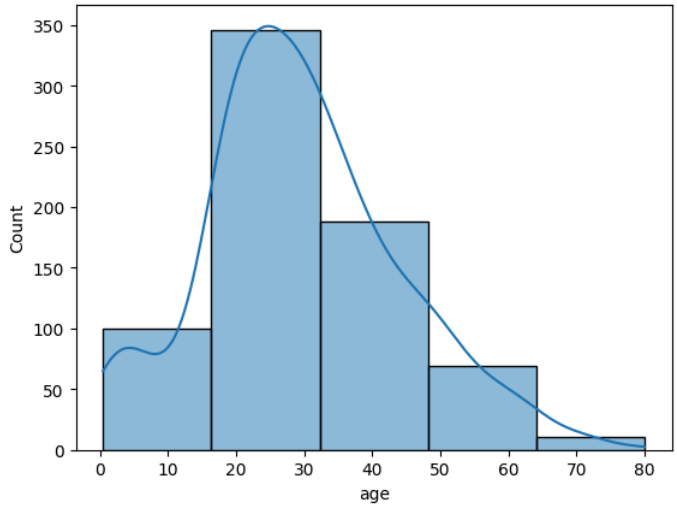
	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
258	1	1	female	35.0	0	0	512.3292	C	First	woman	False	NaN	Cherbourg	yes	True
679	1	1	male	36.0	0	1	512.3292	C	First	man	True	B	Cherbourg	yes	False
737	1	1	male	35.0	0	0	512.3292	C	First	man	True	B	Cherbourg	yes	True

Istogramma

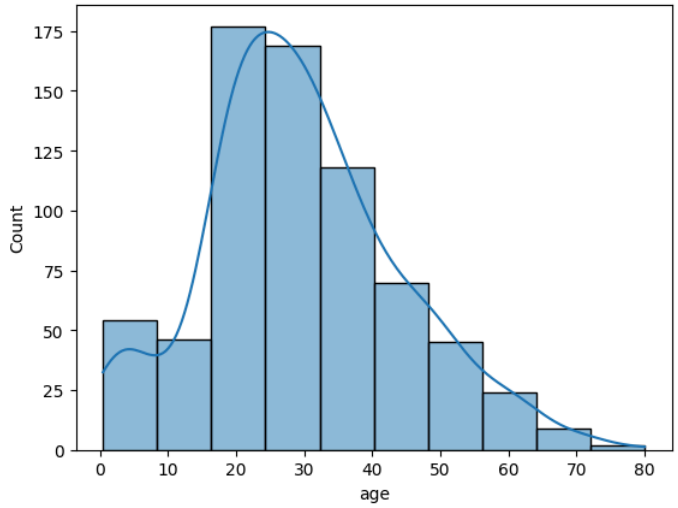
In [16]:
```
g = sns.histplot(data=titanic.query("survived==1"), x="age",kde=True)
```



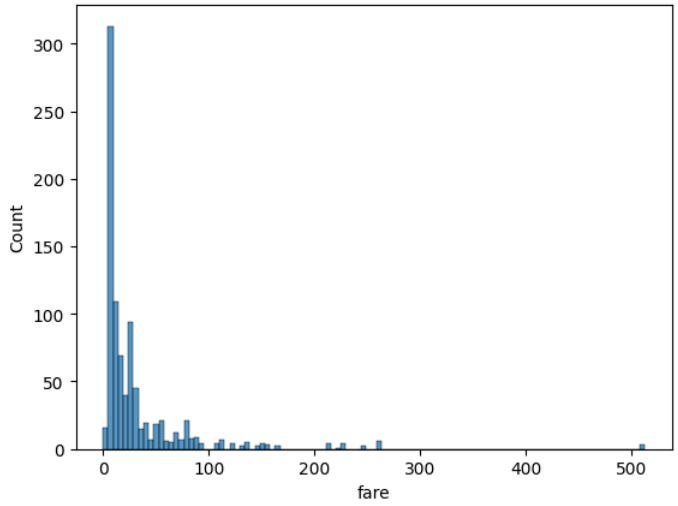
In [17]:
```
g = sns.histplot(data=titanic, x="age",kde=True,bins=5)
```



In [18]:
```
g = sns.histplot(data=titanic, x="age",kde=True,binwidth = 8)
```



In [19]:
```
g = sns.histplot(data=titanic, x="fare")
```



Rappresentiamo due variabili quantitative

In [20]:
```
g = sns.pairplot(data=titanic[["age","fare"]])
```

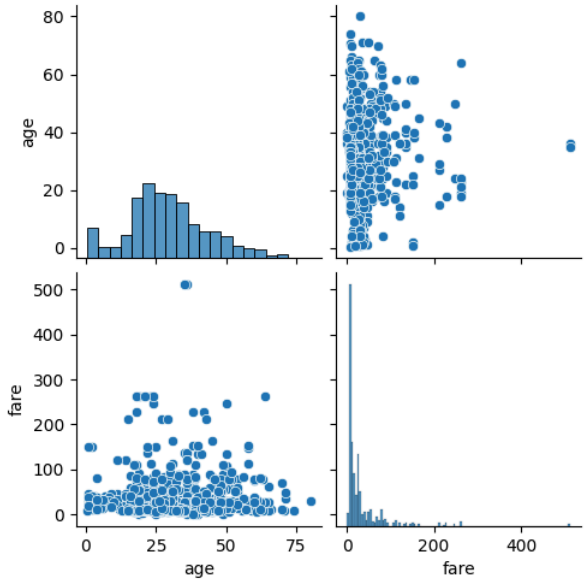
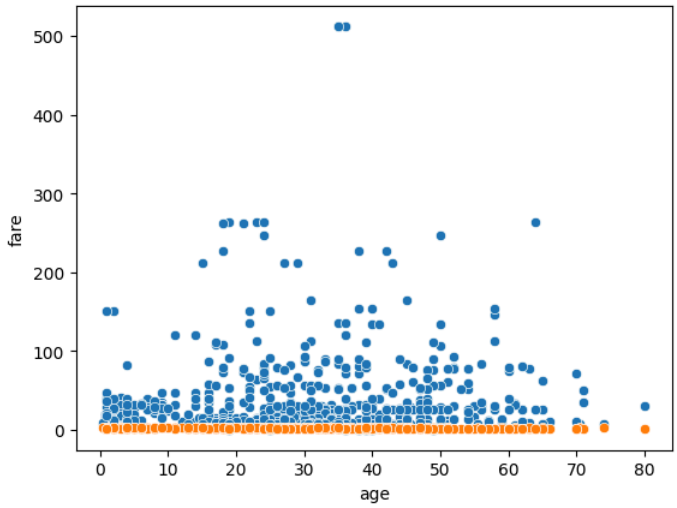


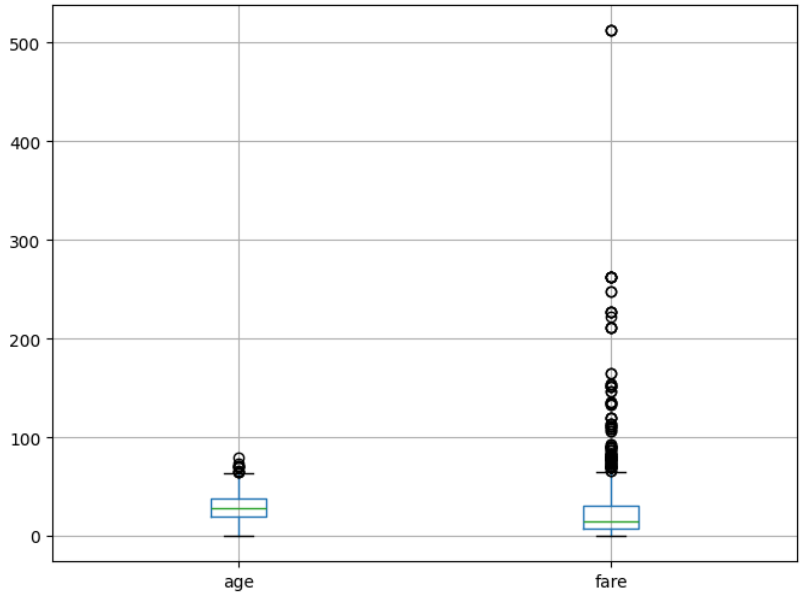
Diagramma a dispersione

In [21]:
```
for colonna in ["fare","pclass"]:
    g = sns.scatterplot(data=titanic, x="age", y=colonna )
```

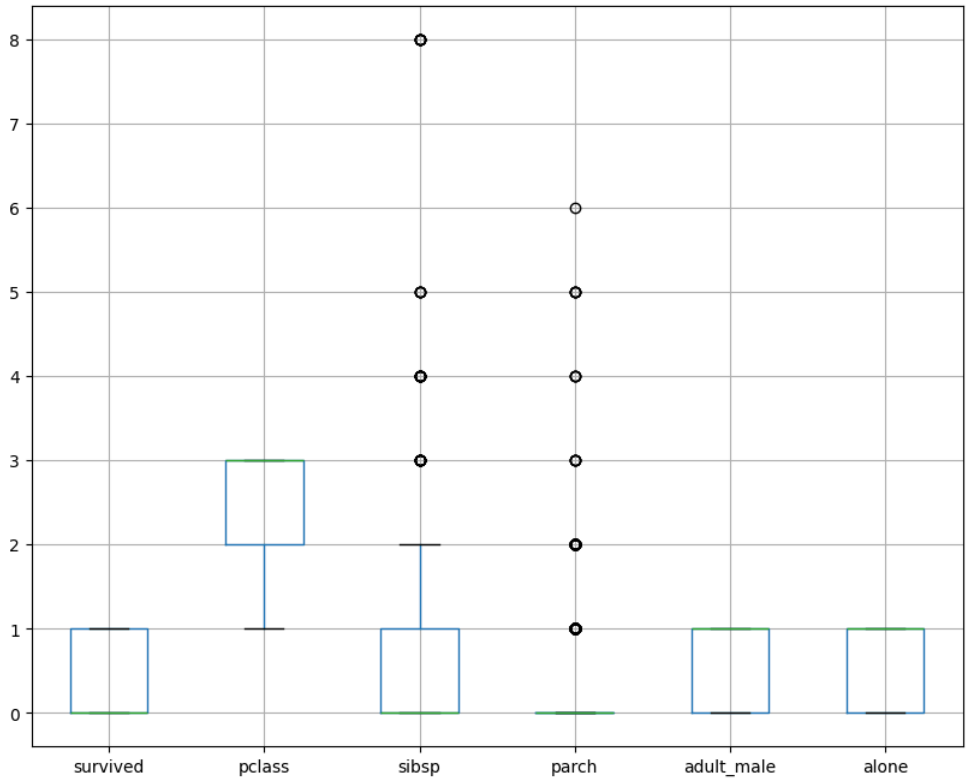


Boxplot

In [22]:
```
g = titanic.boxplot(column = ["age","fare"], figsize = (8,6))
```

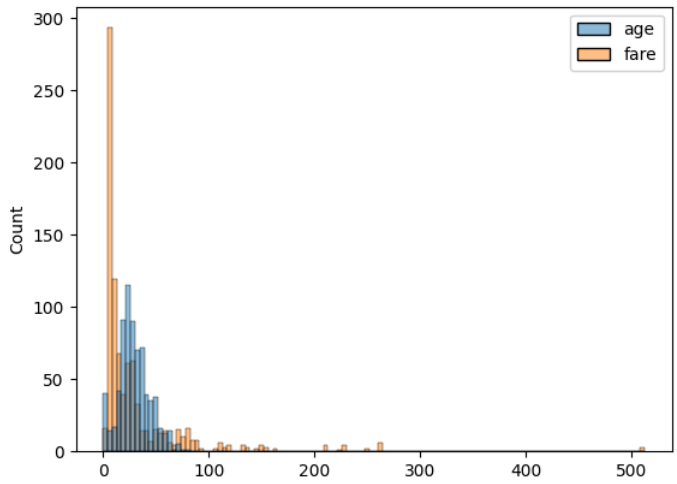


In [23]:
```
g = titanic.drop(["fare","age"],axis=1).boxplot(figsize = (10,8))
```

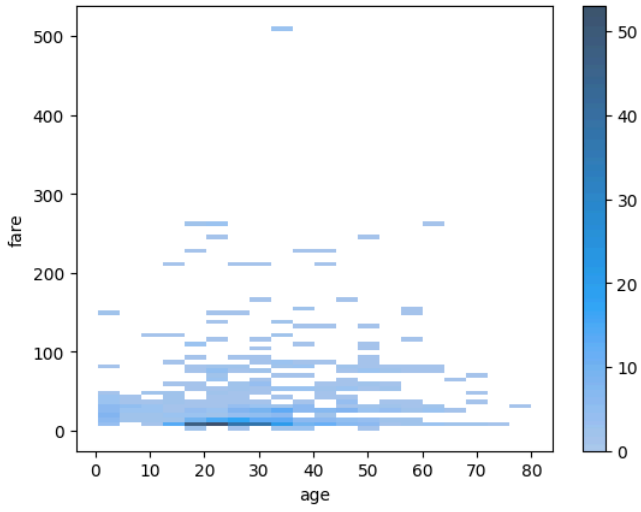


Istogramma

In [24]:
```
g = sns.histplot(data=titanic[["age","fare"]])
```



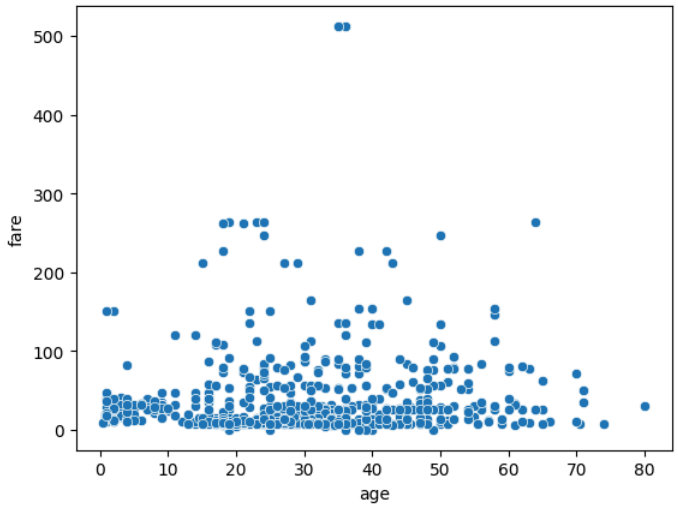
In [25]:
```
g = sns.histplot(data=titanic, x="age",y="fare",  cbar=True)
```



Rappresentazioni miste

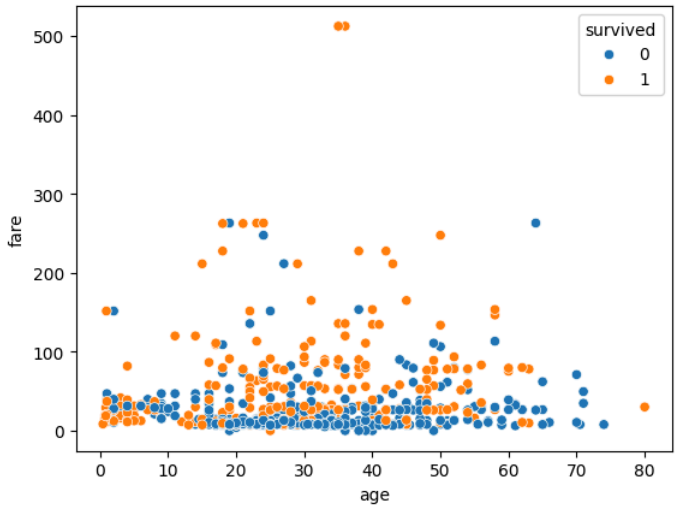
Partendo da qui

In [26]:
```
g = sns.scatterplot(data=titanic, x="age", y="fare" )
```



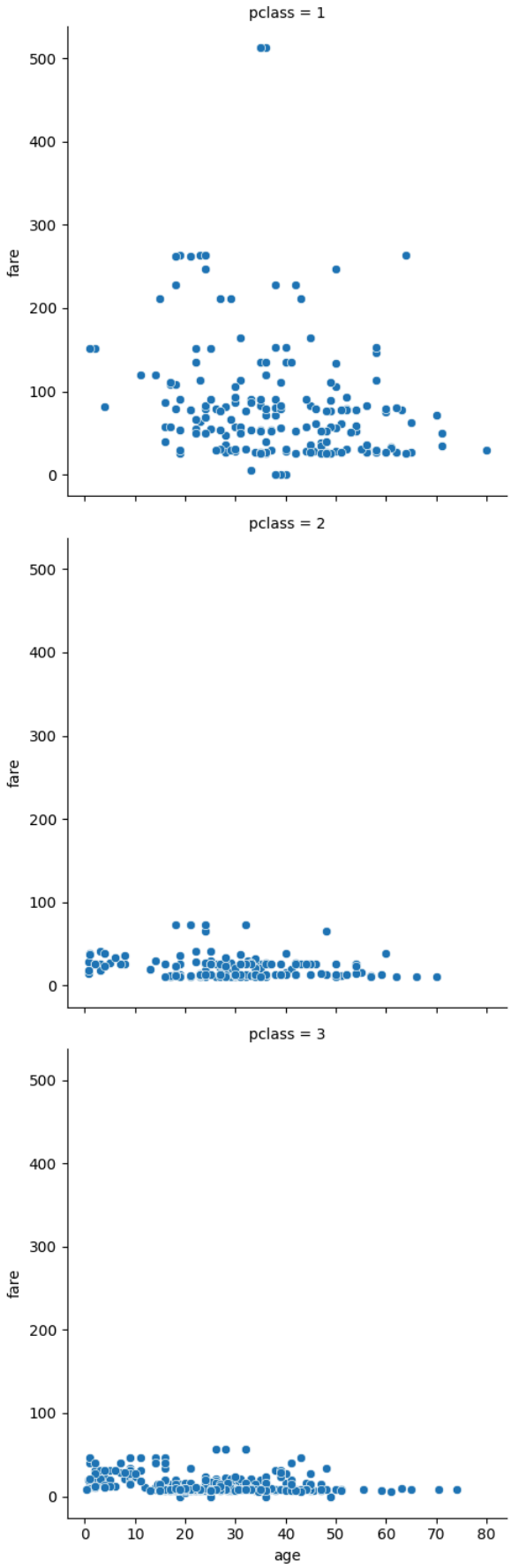
aggiungiamo una qualitativa nello stesso grafico

In [27]:
```
g = sns.scatterplot(data=titanic, x="age", y="fare", hue="survived" )
```



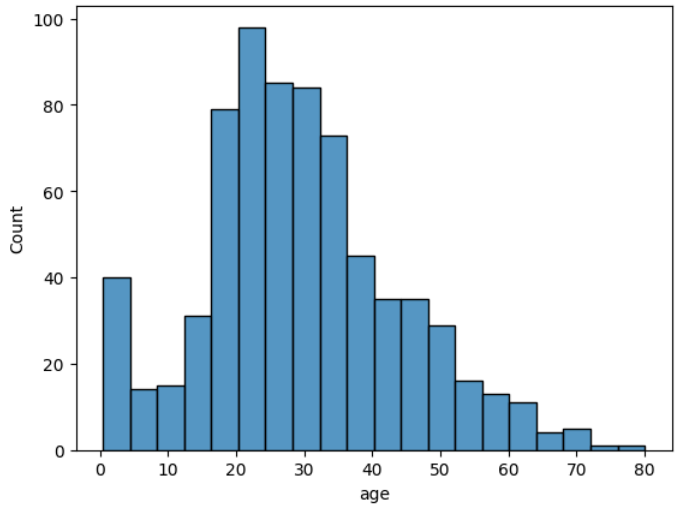
o splittando in grafico diversi

In [28]:
```
g = sns.relplot(x="age", y="fare", data=titanic, row="pclass")
```



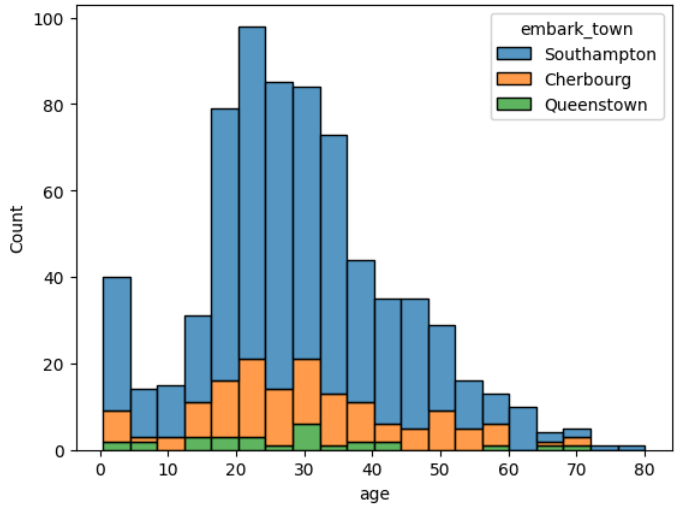
Partendo da qui

In [29]:
```
g = sns.histplot(data=titanic, x="age")
```



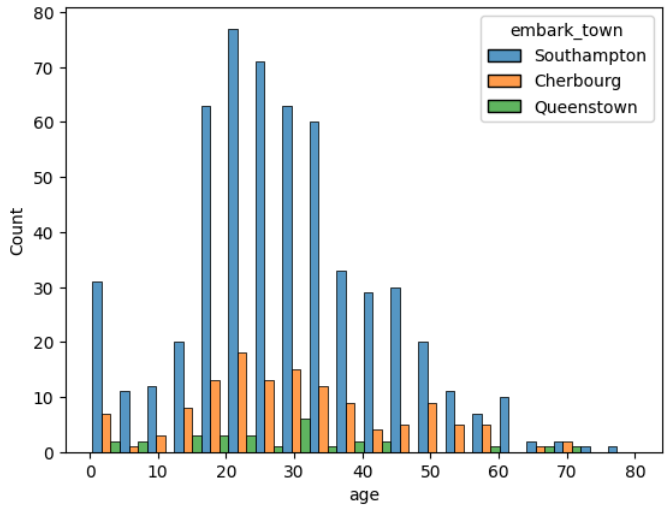
aggiungiamo una qualitativa così

In [30]:
```
g = sns.histplot(
    data=titanic, x="age", hue="embark_town", multiple="stack"
    )
```



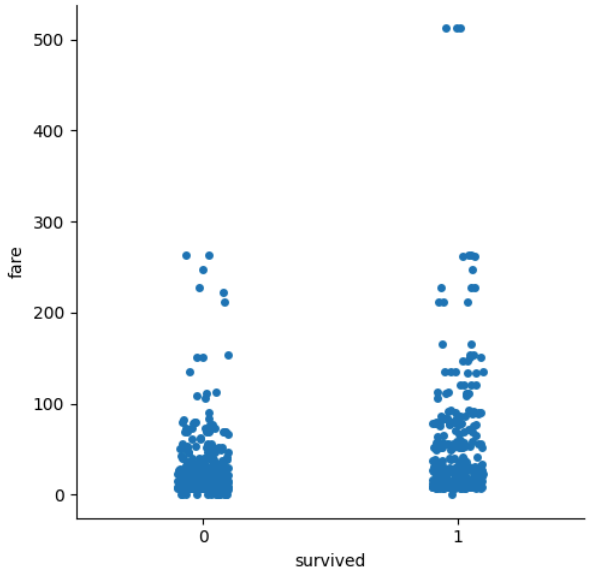
oppure così

In [31]:
```
g = sns.histplot(
    data=titanic, x="age", hue="embark_town", multiple="dodge"
    )
```



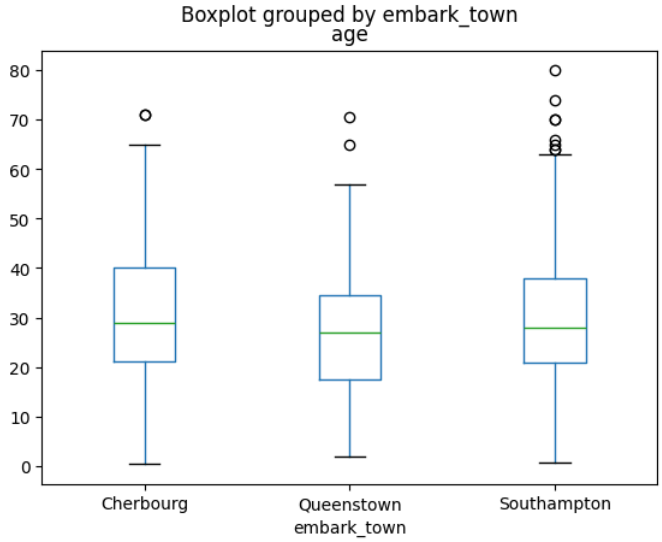
o al contrario

In [32]:
```
g = sns.catplot(data=titanic, x="survived", y="fare")
```



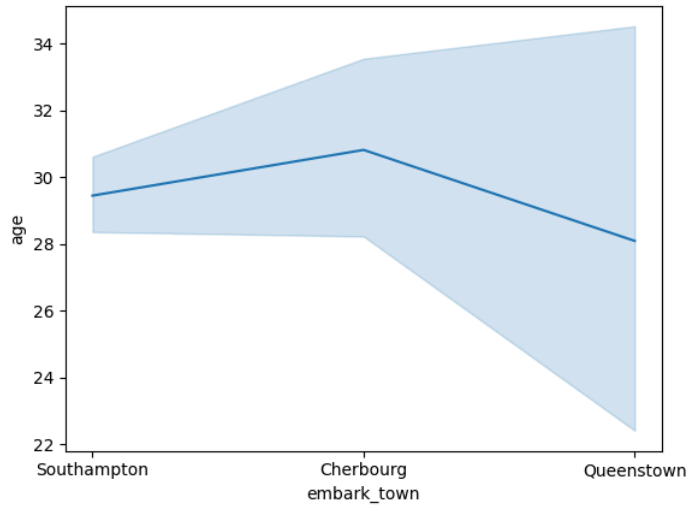
o se voglio i box plot

In [33]:
```
g = titanic.boxplot(by ='embark_town', column =['age'], grid = False)
```



se invece embark_town fosse una data...

In [34]:
```
g = sns.lineplot( data=titanic, x="embark_town",  y="age")
```



Esercitazione sul dataset penguins

In [35]:
```
import seaborn as sns
import pandas as pd
```

Importiamo tramite la libreria seaborn il dataset **penguins**

Citazione: Gorman KB, Williams TD, Fraser WR (2014) Ecological Sexual Dimorphism and Environmental Variability within a Community of Antarctic Penguins (Genus Pygoscelis). PLoS ONE 9(3): e90081. doi:10.1371/journal.pone.0090081

In [36]:
```
penguins = sns.load_dataset("penguins")
```

Spiegazione variabili

- species -> variabile target contente la specie del pinguino
- island -> nome dell'isola
- bill length -> lunghezza becco
- bill_depth -> profondità becco
- flipper length -> lunghezza pinna
- body mass -> massa corporea
- sex -> sesso

In [37]:
```
penguins.head(10)
```

Out[37]:

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
5	Adelie	Torgersen	39.3	20.6	190.0	3650.0	Male
6	Adelie	Torgersen	38.9	17.8	181.0	3625.0	Female
7	Adelie	Torgersen	39.2	19.6	195.0	4675.0	Male
8	Adelie	Torgersen	34.1	18.1	193.0	3475.0	NaN
9	Adelie	Torgersen	42.0	20.2	190.0	4250.0	NaN

In [38]:
```
penguins[["species"]].drop_duplicates()
```

Out[38]:

	species
0	Adelie
152	Chinstrap
220	Gentoo

Iniziamo a ottenere informazione sui tipi e sul numero di null

In [39]:
```
penguins.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   species            344 non-null    object 
 1   island             344 non-null    object 
 2   bill_length_mm     342 non-null    float64
 3   bill_depth_mm      342 non-null    float64
 4   flipper_length_mm  342 non-null    float64
 5   body_mass_g        342 non-null    float64
 6   sex                333 non-null    object 
dtypes: float64(4), object(3)
memory usage: 18.9+ KB
```

e sulla statistica monovariata di base

In [40]:
```
penguins.describe(include='all')
```

Out[40]:

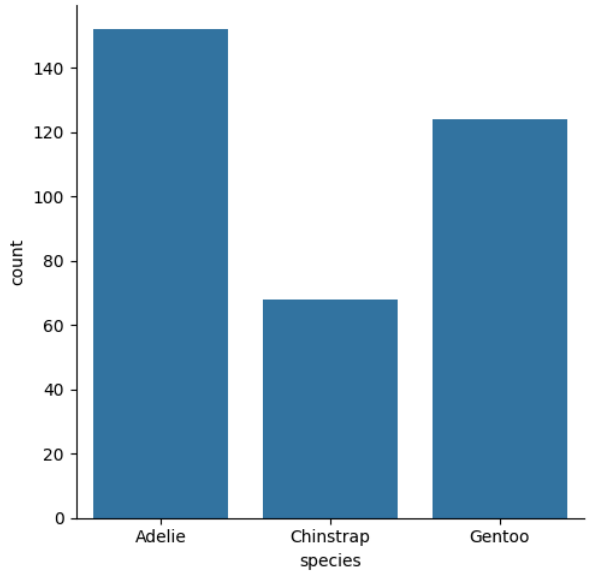
	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
count	344	344	342.000000	342.000000	342.000000	342.000000	333
unique	3	3	NaN	NaN	NaN	NaN	2
top	Adelie	Biscoe	NaN	NaN	NaN	NaN	Male
freq	152	168	NaN	NaN	NaN	NaN	168
mean	NaN	NaN	43.921930	17.151170	200.915205	4201.754386	NaN
std	NaN	NaN	5.459584	1.974793	14.061714	801.954536	NaN
min	NaN	NaN	32.100000	13.100000	172.000000	2700.000000	NaN
25%	NaN	NaN	39.225000	15.600000	190.000000	3550.000000	NaN
50%	NaN	NaN	44.450000	17.300000	197.000000	4050.000000	NaN
75%	NaN	NaN	48.500000	18.700000	213.000000	4750.000000	NaN
max	NaN	NaN	59.600000	21.500000	231.000000	6300.000000	NaN

Iniziamo a studiare la distribuzione della variabile target species

In [41]:
```
penguins.groupby(by="species").size()
```

Out[41]:
```
species
Adelie       152
Chinstrap     68
Gentoo       124
dtype: int64
```

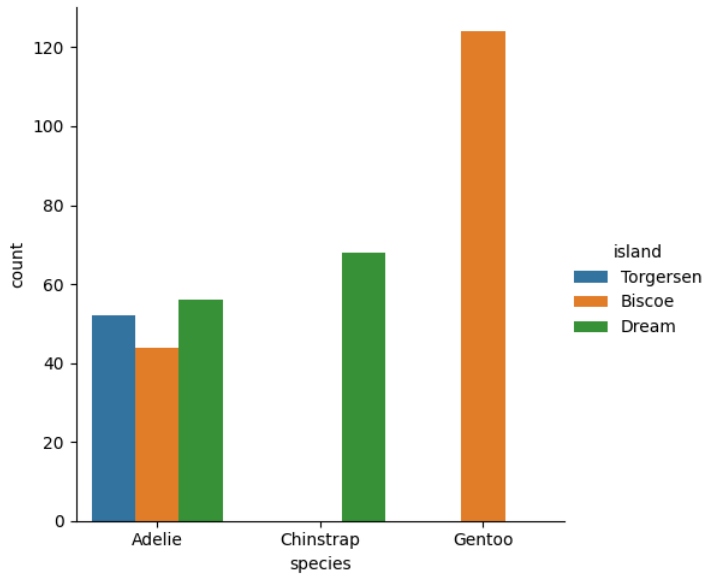
In [42]:
```
g = sns.catplot(data=penguins, x="species", kind="count")
```



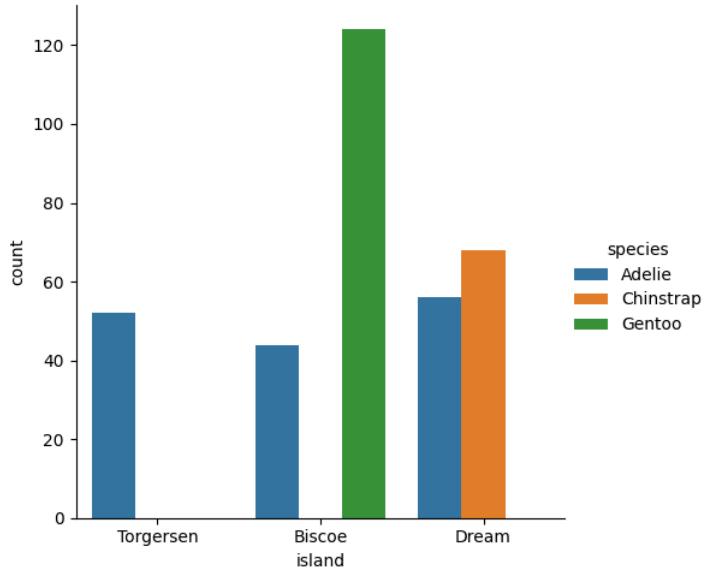
Conclusioni: abbiamo tre classi distinte, non distribuite uniformemente

Rappresentare con un opportuno grafico le relazioni tra le variabili species e island

In [43]:
```
g = sns.catplot(
    data=penguins, x="species", hue="island", kind="count"
    )
```



In [44]:
```
g = sns.catplot(
    data=penguins, x="island", hue="species", kind="count"
    )
```



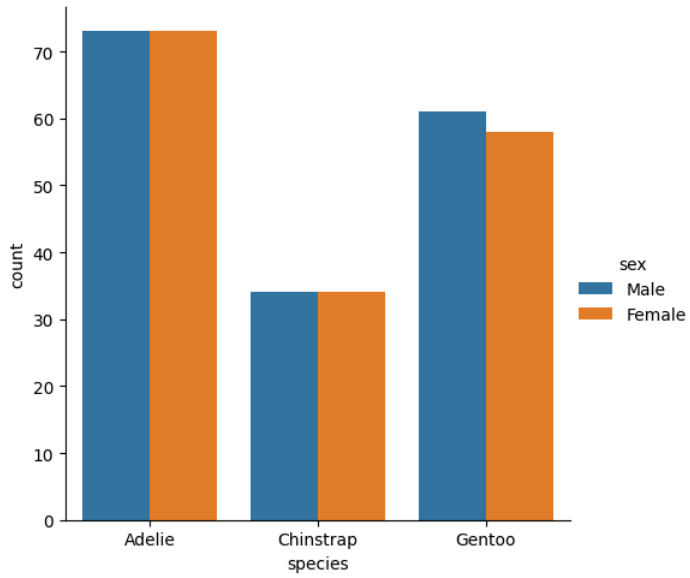
Conclusioni:

- la specie Gentoo vive solo sull'isola Biscoe
- la specie Chinstrap vive solo sull'isola Dream
- la specie Adelie vive in modo abbastanza uniforme su tutte e tre le isole

Di conseguenza sull'isola Torgersen vivono solo pingiuini della specie Adelie. Sulle altre due isole vivono solo due specie alla volta, di cui una è sempre Torgersen

Rappresentare con un opportuno grafico le relazioni tra le variabili species e il sesso

In [45]:
```
g = sns.catplot(
    data=penguins, x="species", hue="sex", kind="count"
    )
```

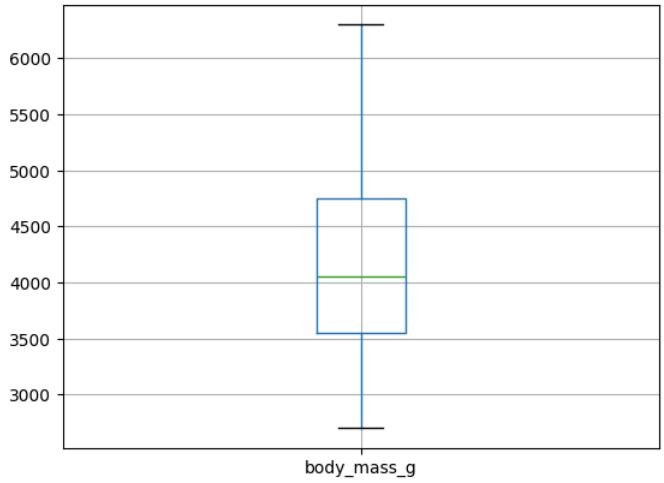


Conclusione: Di ogni specie è presente praticamente lo stesso numero di maschi e femmine. Il sesso non sembra dunque una variabile significativa per predire la specie, almeno considerata singolarmente (potrebbe diventare significativa se abbinata ad altre variabili).

Rappresentare le variabili numeriche per individuare outlier e correlazioni

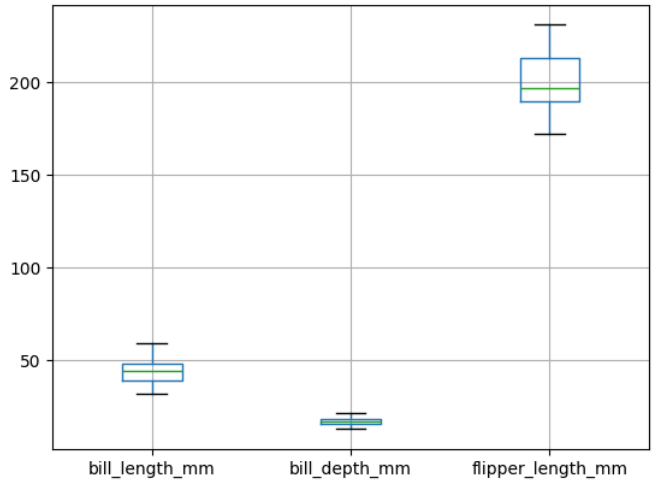
In [46]:
```
penguins[["body_mass_g"]].boxplot()
```

Out[46]: <Axes: >

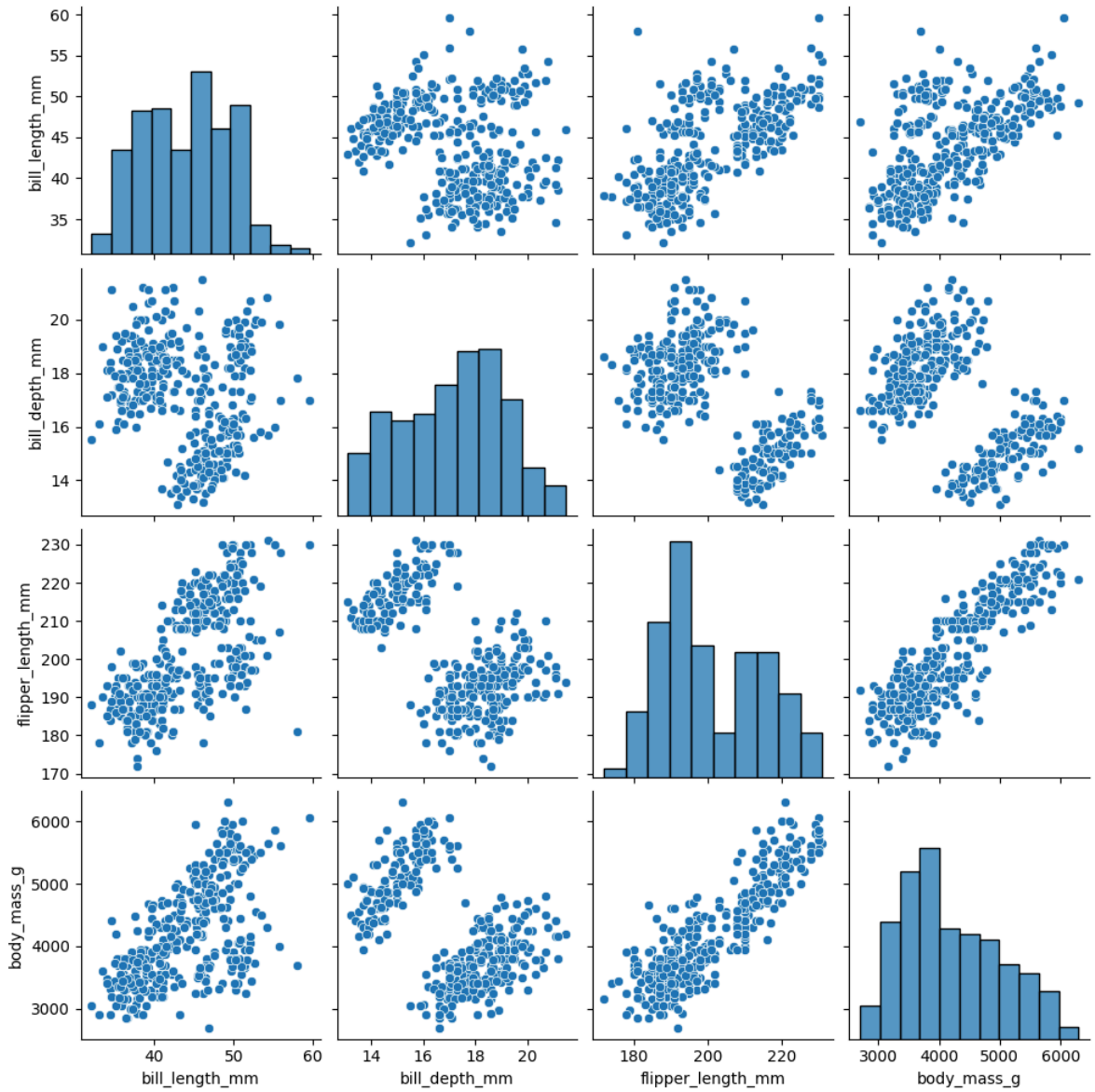


In [47]:
```
penguins[["bill_length_mm","bill_depth_mm","flipper_length_mm"]].boxplot()
```

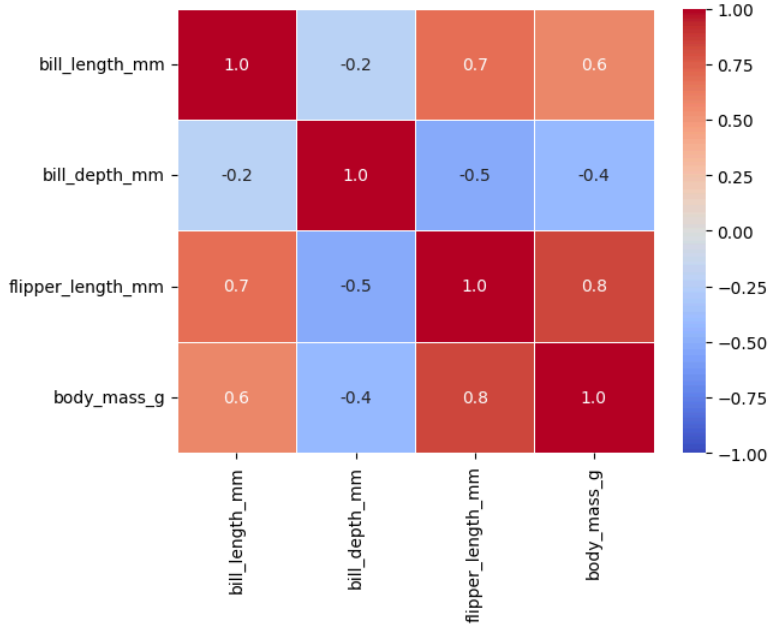
Out[47]: <Axes: >



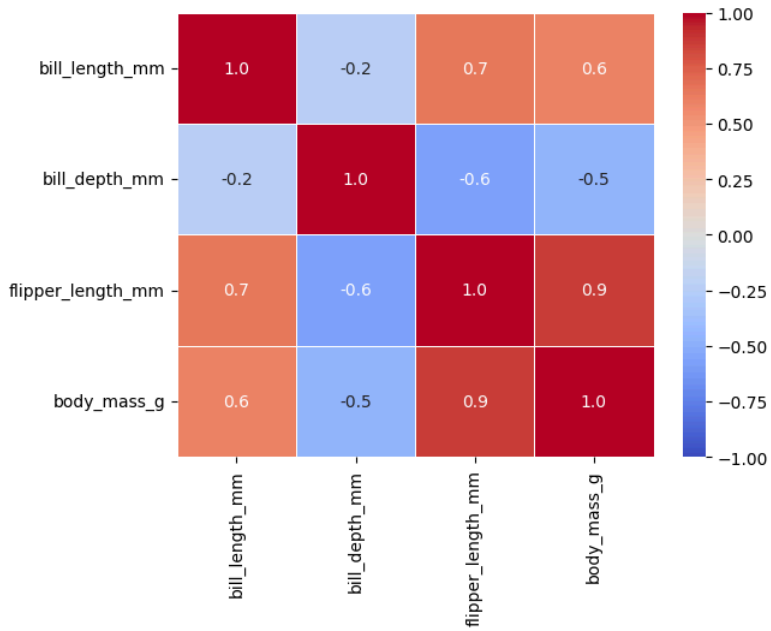
In [48]:
```
g = sns.pairplot(data=penguins[["bill_length_mm","bill_depth_mm","flipper_length_mm","body_mass_g"]])
```



In [49]:
```
g = sns.heatmap(data = penguins.corr(numeric_only=True, method="spearman"), vmin=-1, vmax=1, annot=True, fmt=".1f", linewidth=.5, cmap='coolwarm'
```



In [50]:
```
g = sns.heatmap(data = penguins.corr(numeric_only=True, method="pearson"), vmin=-1, vmax=1, annot=True, fmt=".1f", linewidth=.5, cmap='coolwarm')
```



Conclusioni:

i boxplot e gli istogrammi non mostrano la presenza di outlier significativi. La matrice di correlazione e i box plot mostrano una forte correlazione positiva tra :

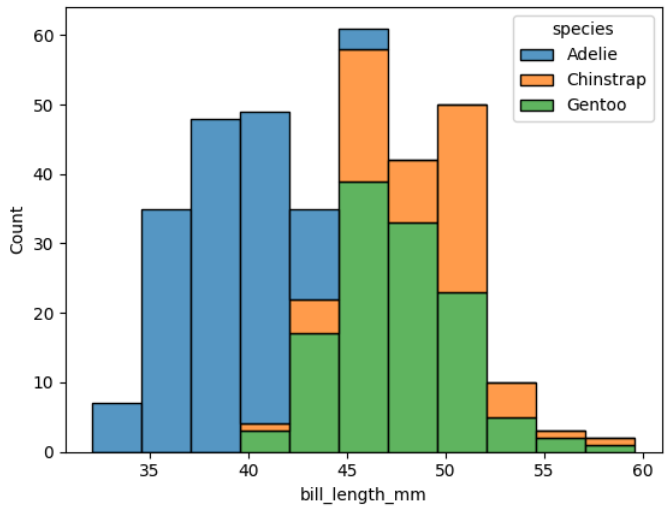
- le variabili flipper_length_mm e body_mass_g (lunghezza della pinna e massa corporea)
- leggermente meno accentuata tra flipper_length_mm e bill_length_mm (lunghezza della pinna e lunghezza del becco)

Si potrebbe pensare di effettuare dei test senza la variabile flipper_length_mm

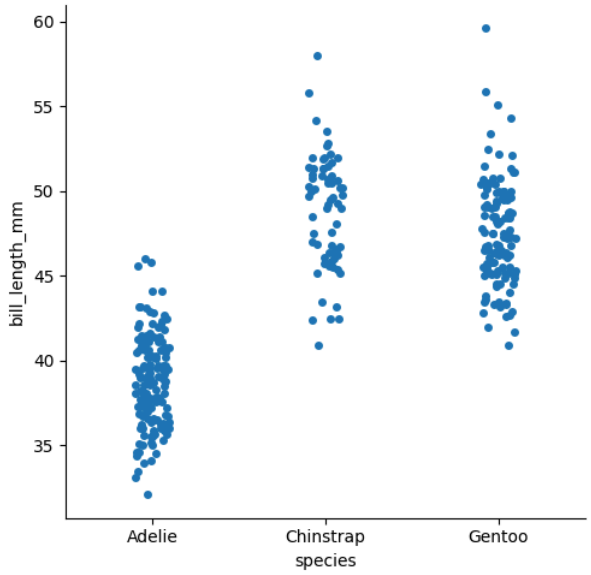
Notiamo invece che (un po' a sorpresa per me) non c'è correlazione tra lunghezza e profondità del becco (bill_length_mm e bill_depth_mm)

Rappresentare con un opportuno grafico le relazioni tra le variabili species e la lunghezza del becco

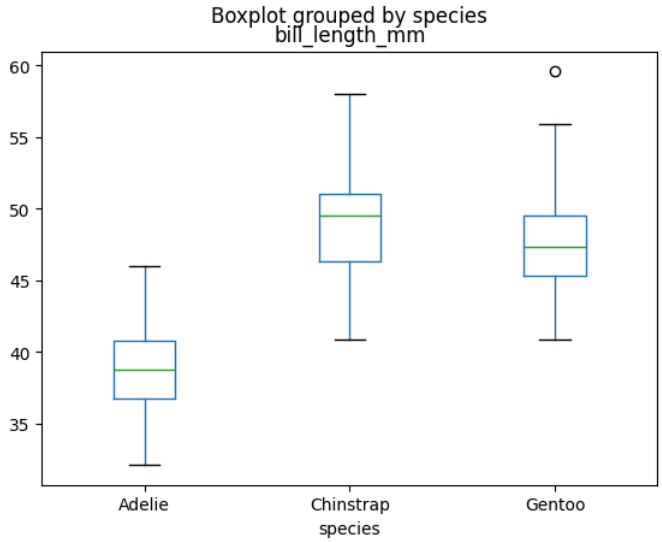
In [51]:
```
g = sns.histplot(
    data=penguins, x="bill_length_mm", hue="species", multiple="stack"
    )
```



In [52]:
```
g = sns.catplot(data=penguins, x="species", y="bill_length_mm")
```



In [53]:
```
g = penguins.boxplot(by ='species', column =['bill_length_mm'], grid = False)
```



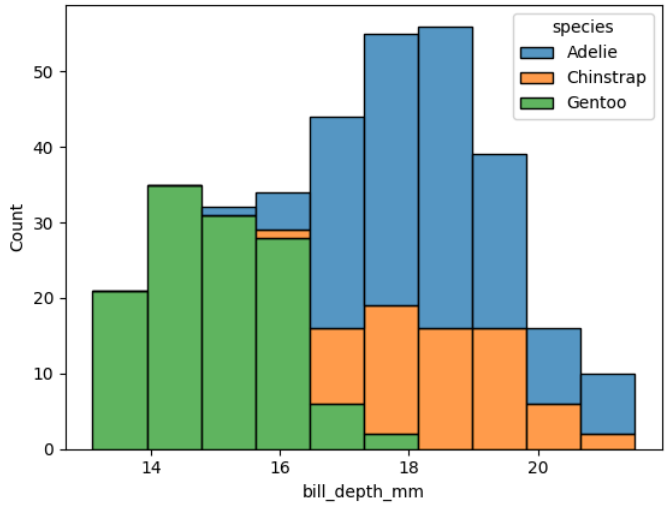
Conclusioni:

I tre grafici mostrano in modo diverso la stessa conclusione: la lunghezza del becco si distribuisce in modo simile tra Chinstrap e Gentoo, ma in modo significativamente differente tra queste due specie e Adelie. La lunghezza dei becchi di Adelie è generalmente minore (solo Adelie ha becchi lunghi meno di 40mm, nessun Adelie ha becchi oltre 47mm circa, restano dei dubbi soltando in questo intervallo, soprattutto tra 42 e 45).

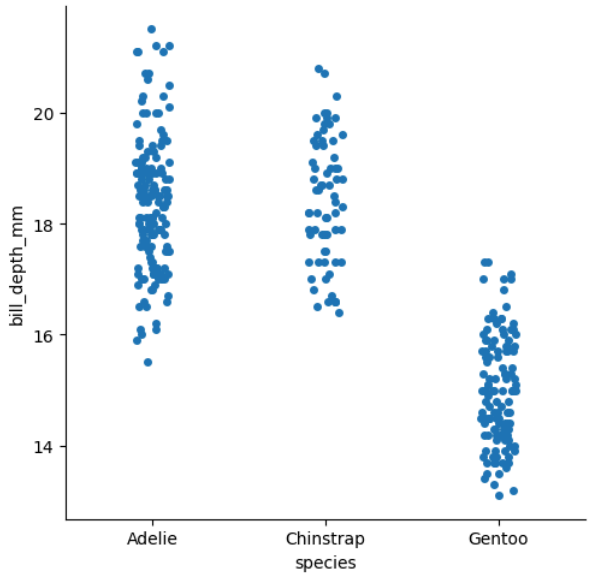
Questa informazioni è molto importante in quanto può essere combinata a quanto già visto riguardo la relazione tra specie e isola! Chinstrap e Gentoo infatti vivono su isole diverse, all'interno della stessa isola convivono solo con Adelie e tra loro possono essere riconosciuti abbastanza bene dalla lunghezza del becco

Rappresentare con un opportuno grafico le relazioni tra le variabili species e la profondità del becco

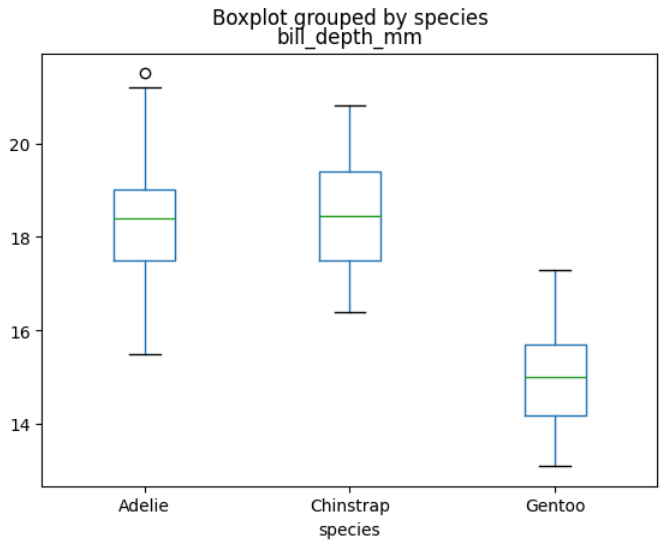
In [54]:
```
g = sns.histplot(
    data=penguins, x="bill_depth_mm", hue="species", multiple="stack"
    )
```



In [55]:
```
g = sns.catplot(data=penguins, x="species", y="bill_depth_mm")
```



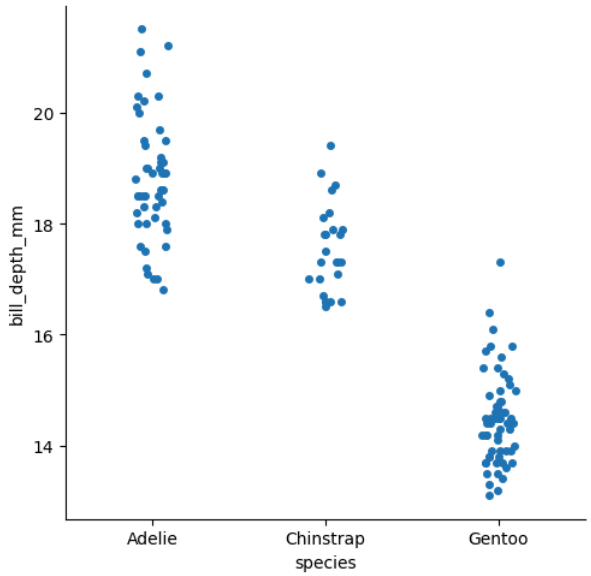
In [56]:
```
g = penguins.boxplot(by ='species', column =['bill_depth_mm'], grid = False)
```



Conclusioni:

In questo caso osserviamo che Adelie e Chinstrap hanno becchi di profondità simili, invece i valori per i Gentoo sono generalmente minori. Questo potrebbe dare agli algoritmi di Machine Learning un nuovo criterio di scelta tra le specie Gentoo e Adelie quando la lunghezza dei becchi è tra 40 e 47. Vediamo cosa succede per questo range dove c'era sovrapposizione tra le specie.

In [57]:
```
g = sns.catplot(data=penguins.query("bill_length_mm >= 40 and bill_length_mm <=47"), x="species", y="bill_depth_mm")
```



Osserviamo che la specie Gentoo ha quasi sempre un becco di profondità inferiore

Studiamo la condizione if

Con l'istruzione if possiamo eseguire soltanto porzioni di codice in base al verificarsi di una determinata condizione.

Ecco un esempio: le istruzioni print("zero") e print("numero negativo") non saranno eseguite

In [1]:
```
x = 1
if x>0:
    print("numero positivo")
elif x == 0:
    print("zero")
else:
    print("numero negativo")
```

```
numero positivo
```

In questo caso. invece, l'istruzione print("numero positivo") non sarà eseguita

In [2]:
```
x = -1
if x>0:
    print("numero positivo")
else:
    print("numero negativo")
```

```
numero negativo
```

Attenzione a questi tre codici simili. Studiare il risultato nei vari casi

In [3]:
```
x = 10
if x > 5:
    print("numero maggiore di 5")
elif x >= 0 :
    print("numero maggiore o uguale a 0")
else:
    print("numero negativo")
```

```
numero maggiore di 5
```

In [4]:
```
x = 10
if x >= 0:
    print("numero maggiore o uguale a 0")
elif x > 5 :
    print("numero maggiore di 5")
else:
    print("numero negativo")
```

```
numero maggiore o uguale a 0
```

In [5]:
```
x = 10
if x > 5:
    print("numero maggiore di 5")
if x >= 0:
    print("numero maggiore o uguale a 0")
else:
    print("numero negativo")
```

```
numero maggiore di 5
numero maggiore o uguale a 0
```

Studiamo la condizione while

Con l'istruzione while posso ripetere delle porzioni di codice più volte, fin quando resta verificata una determinata condizione espressa all'inizio del ciclo while.

Ad esempio questo codice ripeterà più volte l'istruzione x = input("Inserisci un numero"), finquando non inseriremo un numero negativo

In [6]:
```
x = input("Inserisci un numero")

while int(x) >= 0:
    x = input("Inserisci un numero")

print("hai interrotto il ciclo perché hai inserito un numero negativo")
```

```
hai interrotto il ciclo perché hai inserito un numero negativo
```

Break e continue all'interno del while

L'istruzione break provoca **immediatamente** l'uscita dal ciclo While, interrompando l'esecuzione corrente e annullando anche le eventuali successive.

Ad esempio il codice seguente stampa soltanto i valori 1, 2. Quando x è uguale a 3, la presenza del break provoca sia la fine dell'iterazione corrente e sia la fine dell'intero ciclo while, anche se la condizione x<=5 è ancora vera

In [7]:
```
x = 1
while x<5:
    if(x==3):
        break
    print(x)
    x = x+1
```

```
1
2
```

L'istruzione continue provoca soltanto l'interruzione dell'iterazione corrente. La condizione del while sarà nuovamente rivalutata.

Ad esempio nel codice seguente saltiamo l'istruzione print(x) quando x è uguale a 3

In [8]:
```
x = 0
while x<5:
    x=x+1
    if(x==3):
        continue
    print(x)
```

```
1
2
4
5
```

Facciamo attenzione! **Potremmo incappare facilmente in codici di durata infinita**, come il seguente:

```
x = 0

while x<5:

    if(x==3):
        continue

    x=x+1
```

Una volta arriviti a 3, entriamo in un loop infinito: la x non si incrementa per via del continue. D'altra parte il ciclo non finisce perché la x continua ad essere sempre minore di 5

Gestione errori con try expect

Se un'istruzione presente nel blocco try genera un errore, il codice passa immediatamente al blocco except, senza eseguire le istruzioni seguenti nel blocco try.

Ad esempio nel caso seguente non verrà eseguita l'istruzione print("sono nel try") a causa dell'errore generato dalla divisione per 0

In [9]:
```
try:
    1/0
    print("sono nel try")
except:
    print("si è generato un errore")
```

```
si è generato un errore
```

Se le istruzioni del blocco try non generano errori, il blocco except non viene eseguito

In [10]:
```
try:
    1/1
    print("sono nel try")
except:
    print("si è generato un errore")
```

```
sono nel try
```

Machine Learning con Python

Carichiamo i dati del famaso dataset iris.

Citazione Fisher,R. A.. (1988). Iris. UCI Machine Learning Repository. https://doi.org/10.24432/C56C76. This dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license. https://creativecommons.org/licenses/by/4.0/legalcode

In [1]:
```
import seaborn as sns
import pandas as pd
import numpy as np
```

In [2]:
```
iris=sns.load_dataset("iris")
```

In [3]:
```
iris.head(5)
```

Out[3]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [4]:
```
#se non hai installato seaborn
iris = pd.read_csv(filepath_or_buffer=r"C:\Users\ianto\Desktop\Corso Python\file\iris.csv",
                   header=1)
```

Diamo preliminarmente un'occhiata ai dati

In [5]:
```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

Partiamo con una lista contenente i valori di un nuovo iris

In [6]:
```
nuovo_iris = [4.3, 3.2, 1.4, 0.3]
```

In [7]:
```
#Osservazione: su python posso valorizzare più variabili contemporneamente con questa sintassi
#x, y, z, w = 4.3, 3.2, 1.4, 0.3
```

Creiamo una colonna distanza

In [8]:
```
iris["distanza"] = np.sqrt( (iris["sepal_length"]-nuovo_iris[0])**2 + \
                            (iris["sepal_width"]-nuovo_iris[1])**2 + \
                            (iris["petal_length"]-nuovo_iris[2])**2 + \
                            (iris["petal_width"]-nuovo_iris[3])**2)
```

Ordiniam i valori dello storico per distanza e prendiamo l'iris più vicino

In [9]:
```
iris.sort_values(by="distanza", ascending = True).head(1)
```

Out[9]:

	sepal_length	sepal_width	petal_length	petal_width	species	distanza
42	4.4	3.2	1.3	0.2	setosa	0.173205

Generalizziamo il procedimento con scikit-learn

In [25]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

```
import pandas as pd
import numpy as np
import seaborn as sns
```

Importiamo il dataset iris

In [26]:
```
iris=sns.load_dataset("iris")
```

In [27]:
```
#se non hai installato seaborn
iris = pd.read_csv(filepath_or_buffer=r"C:\Users\ianto\Desktop\Corso Python\file\iris.csv",
                   header=1)
```

Ripassiamo come si "elimina" una colonna

In [28]:
```
iris.drop("species", axis=1).head(5)
```

Out[28]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Dividiamo i dati in training e test utilizzando from il metodo train_test_split di scikit-learn

In [29]:
```
features_training,features_test, target_training, target_test = train_test_split(iris.drop("species", axis=1),
                                                                                 iris[["species"]],
                                                                                 test_size = 0.3,
                                                                                 shuffle = True,
                                                                                 random_state = 1)
```

Guardiamo i quattro dataset

In [30]:
```
features_training.head(5)
```

Out[30]:

	sepal_length	sepal_width	petal_length	petal_width
118	7.7	2.6	6.9	2.3
18	5.7	3.8	1.7	0.3
4	5.0	3.6	1.4	0.2
45	4.8	3.0	1.4	0.3
59	5.2	2.7	3.9	1.4

In [31]:
```
target_training.head(5)
```

Out[31]:

	species
118	virginica
18	setosa
4	setosa
45	setosa
59	versicolor

In [32]:
```
features_test.head(5)
```

Out[32]:

	sepal_length	sepal_width	petal_length	petal_width
14	5.8	4.0	1.2	0.2
98	5.1	2.5	3.0	1.1
75	6.6	3.0	4.4	1.4
16	5.4	3.9	1.3	0.4
131	7.9	3.8	6.4	2.0

In [33]:
```
target_test.head(5)
```

Out[33]:

	species
14	setosa
98	versicolor
75	versicolor
16	setosa
131	virginica

Istanziamo l'algoritmo K-Neighbours tramite scikit-learn

In [34]:
```
knc = KNeighborsClassifier(n_neighbors=1)
```

Alleniamolo soltanto sui dati di training tramite il metodo fit

In [35]:
```
#per eliminare il warning usare target_training.values.ravel()
knc.fit(features_training,target_training)
```

```
C:\Users\ianto\Desktop\ambiente_python\Lib\site-packages\sklearn\neighbors\_classification.py:239: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  return self._fit(X, y)
```

Out[35]:

KNeighborsClassifier

KNeighborsClassifier(n_neighbors=1)

Utilizziamo il metodo allenato per calcolare le predizioni sui dati di test tramite il metodo predict

In [36]:
```
predizioni = knc.predict(features_test)
predizioni
```

Out[36]:
```
array(['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica',
       'versicolor', 'virginica', 'setosa', 'setosa', 'virginica',
       'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
       'setosa', 'versicolor', 'versicolor', 'setosa', 'setosa',
       'versicolor', 'versicolor', 'versicolor', 'setosa', 'virginica',
       'versicolor', 'setosa', 'setosa', 'versicolor', 'virginica',
       'versicolor', 'virginica', 'versicolor', 'virginica', 'virginica',
       'setosa', 'versicolor', 'setosa', 'versicolor', 'virginica',
       'virginica', 'setosa', 'versicolor', 'virginica', 'versicolor'],
      dtype=object)
```

Con il metodo score posso calcolare quante di queste predizioni sono corrette

In [37]:
```
score = knc.score(features_test,target_test)
score
```

Out[37]:
```
0.9777777777777777
```

Ricreiamo il dataset features_test aggiungendo i valori delle classi reali e predette

In [38]:
```
iris_completo = features_test.copy()
iris_completo["species_originale"]=target_test["species"]
iris_completo["species_predetta"]=predizioni
iris_completo.head(5)
```

Out[38]:

	sepal_length	sepal_width	petal_length	petal_width	species_originale	species_predetta
14	5.8	4.0	1.2	0.2	setosa	setosa
98	5.1	2.5	3.0	1.1	versicolor	versicolor
75	6.6	3.0	4.4	1.4	versicolor	versicolor
16	5.4	3.9	1.3	0.4	setosa	setosa
131	7.9	3.8	6.4	2.0	virginica	virginica

Vediamo gli errori

In [39]:
```
iris_completo.query("species_originale != species_predetta")
```

Out[39]:

	sepal_length	sepal_width	petal_length	petal_width	species_originale	species_predetta
119	6.0	2.2	5.0	1.5	virginica	versicolor

Calcoliamo di nuovo l'accuratezza "manualmente"

In [40]:
```
len(iris_completo.query("species_originale == species_predetta"))/len(iris_completo)
```

Out[40]:
```
0.9777777777777777
```

Calcoliamo la matrice di confusione

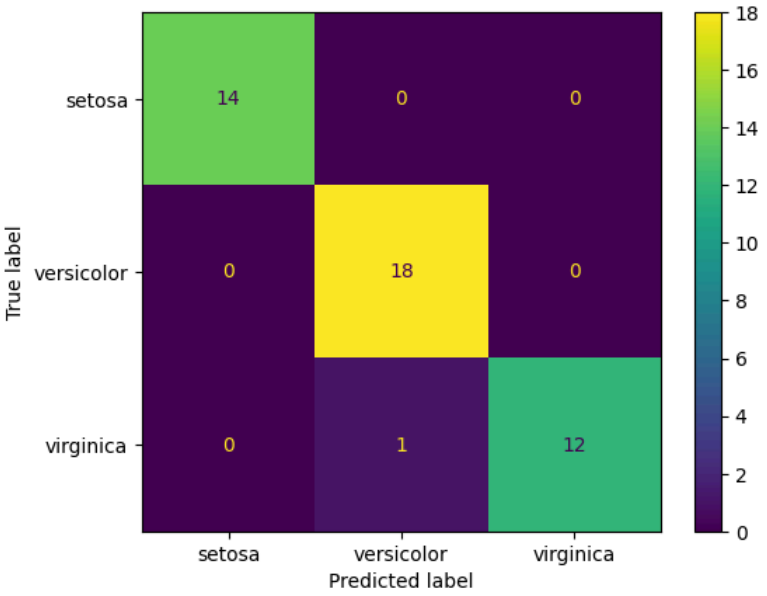
In [41]:
```
confusion_matrix(target_test, predizioni)
```

Out[41]:
```
array([[14,  0,  0],
       [ 0, 18,  0],
       [ 0,  1, 12]])
```

Graficamente

In [42]:
```
ConfusionMatrixDisplay.from_predictions(target_test, predizioni)
```

Out[42]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x26f2f248830>
```



Utilizziamo altri algoritmi di Machine Learning ed estendiamo il processo di lavorazione dei dati

In [43]:
```
from sklearn.impute import SimpleImputer

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay

import pandas as pd
import numpy as np
import seaborn as sns
```

Reimportiamo il dataset e dividiamo i dati in training e test (random_state = 0 ci garantisce di avere gli stessi dati di prima)

In [44]:
```
iris=sns.load_dataset("iris")
```

In [45]:
```
features_training,features_test, target_training, target_test = train_test_split(iris.drop("species", axis=1),
                                                                                  iris[["species"]],
                                                                                  test_size = 0.3,
                                                                                  shuffle = True,
                                                                                  random_state = 1)
```

Valorizziamo i valori mancanti. Partiamo istanziando la classe SimpleImputer di Scikit-learn

In [46]:
```
imputer = SimpleImputer(strategy='mean')
```

Alleniamo l'oggetto imputer sui dati di training

In [47]:
```
imputer.fit(features_training)
```

Out[47]:
```
SimpleImputer
SimpleImputer()
```

Utilizziamo il metodo allenato per trasformare i dati di training e di test

In [48]:
```
features_training_imputed = pd.DataFrame(
    imputer.transform(features_training),
    columns=features_training.columns,
    index=features_training.index
)
```

In [49]:
```
features_training_imputed.head(5)
```

Out[49]:

	sepal_length	sepal_width	petal_length	petal_width
118	7.7	2.6	6.9	2.3
18	5.7	3.8	1.7	0.3
4	5.0	3.6	1.4	0.2
45	4.8	3.0	1.4	0.3
59	5.2	2.7	3.9	1.4

In [50]:
```
features_test_imputed = pd.DataFrame(
    imputer.transform(features_test),
    columns=features_test.columns,
    index=features_test.index
)
```

In [51]:
```
features_test_imputed.head(5)
```

Out[51]:

	sepal_length	sepal_width	petal_length	petal_width
14	5.8	4.0	1.2	0.2
98	5.1	2.5	3.0	1.1
75	6.6	3.0	4.4	1.4
16	5.4	3.9	1.3	0.4
131	7.9	3.8	6.4	2.0

Applichiamo la normalizzazione dei dati. Partiamo istanziando la classe StandardScaler di Scikit-learn

In [52]:
```
scaler = StandardScaler()
```

Alleniamo l'oggetto sui dati di training

In [53]:
```
scaler.fit(features_training_imputed)
```

Out[53]:
```
StandardScaler
StandardScaler()
```

Utilizziamo il metodo allenato per trasformare i dati di training e di test

In [54]:
```
features_training_standard = pd.DataFrame(
    scaler.transform(features_training_imputed),
    columns=features_training_imputed.columns,
    index=features_training_imputed.index
)
```

In [55]:
```
features_training_standard.head(5)
```

Out[55]:

	sepal_length	sepal_width	petal_length	petal_width
118	2.260502	-1.050897	1.776229	1.423710
18	-0.118974	1.827647	-1.144919	-1.142634
4	-0.951790	1.347889	-1.313447	-1.270951
45	-1.189738	-0.091382	-1.313447	-1.142634
59	-0.713843	-0.811018	0.090951	0.268855

In [56]:
```
features_test_standard = pd.DataFrame(
    scaler.transform(features_test_imputed),
    columns=features_test_imputed.columns,
    index=features_test_imputed.index
)
```

In [57]:
```
features_test_standard.head(5)
```

Out[57]:

	sepal_length	sepal_width	petal_length	petal_width
14	0.000000	2.307404	-1.425798	-1.270951
98	-0.832816	-1.290775	-0.414632	-0.116096
75	0.951790	-0.091382	0.371831	0.268855
16	-0.475895	2.067525	-1.369623	-1.014317
131	2.498449	1.827647	1.495350	1.038758

Applichiamo un altro algoritmo di Machine Learning: il Perceptron. Istanziamo la classe relativa da scikit-learn

In [58]:
```
pcp = Perceptron(random_state = 0, alpha=0.001)
```

Alleniamola soltanto sui dati di training

In [59]:
```
#per eliminare il warning usare target_training.values.ravel()
pcp.fit(features_training_standard,target_training )
```

```
C:\Users\ianto\Desktop\ambiente_python\Lib\site-packages\sklearn\utils\validation.py:1408: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)
```

Out[59]:

Perceptron

```
Perceptron(alpha=0.001)
```

Utilizziamo il metodo allenato per calcolare le predizioni sui dati di test

In [60]:
```
predizioni = pcp.predict(features_test_standard)
predizioni
```

Out[60]:
```
array(['setosa', 'versicolor', 'versicolor', 'setosa', 'virginica',
       'virginica', 'virginica', 'setosa', 'setosa', 'virginica',
       'versicolor', 'setosa', 'virginica', 'versicolor', 'versicolor',
       'setosa', 'versicolor', 'versicolor', 'setosa', 'setosa',
       'versicolor', 'versicolor', 'virginica', 'setosa', 'virginica',
       'versicolor', 'setosa', 'setosa', 'versicolor', 'versicolor',
       'versicolor', 'virginica', 'versicolor', 'virginica', 'virginica',
       'setosa', 'versicolor', 'setosa', 'versicolor', 'versicolor',
       'virginica', 'setosa', 'versicolor', 'virginica', 'versicolor'],
      dtype='<U10')
```

Con il metodo score posso calcolare quante di queste predizioni sono corrette

In [61]:
```
score = pcp.score(features_test_standard,target_test)
score
```

Out[61]:
```
0.8888888888888888
```

Ricreiamo il dataset features_test aggiungendo i valori delle classi reali e predette

In [62]:
```
iris_completo = features_test.copy()
iris_completo["species_originale"]=target_test["species"]
iris_completo["species_predetta"]=predizioni
iris_completo.head(5)
```

Out[62]:

	sepal_length	sepal_width	petal_length	petal_width	species_originale	species_predetta
14	5.8	4.0	1.2	0.2	setosa	setosa
98	5.1	2.5	3.0	1.1	versicolor	versicolor
75	6.6	3.0	4.4	1.4	versicolor	versicolor
16	5.4	3.9	1.3	0.4	setosa	setosa
131	7.9	3.8	6.4	2.0	virginica	virginica

In [63]:
```
iris_completo.query("species_originale != species_predetta")
```

Out[63]:

	sepal_length	sepal_width	petal_length	petal_width	species_originale	species_predetta
56	6.3	3.3	4.7	1.6	versicolor	virginica
77	6.7	3.0	5.0	1.7	versicolor	virginica
146	6.3	2.5	5.0	1.9	virginica	versicolor
108	6.7	2.5	5.8	1.8	virginica	versicolor
119	6.0	2.2	5.0	1.5	virginica	versicolor

Calcoliamo di nuovo l'accuratezza "manualmente"

In [64]:
```
len(iris_completo.query("species_originale == species_predetta"))/len(iris_completo)
```

Out[64]:
```
0.8888888888888888
```

Calcoliamo la matrice di confusione

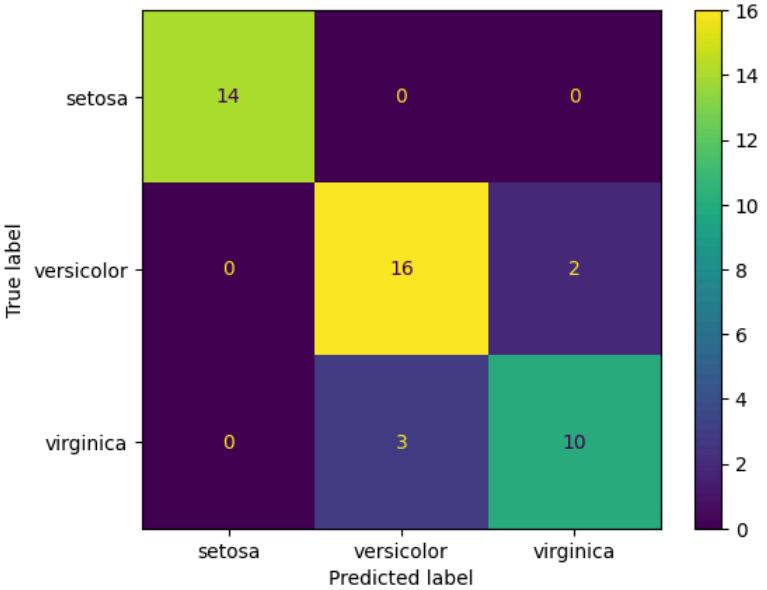
In [65]:
```
confusion_matrix(target_test, predizioni)
```

Out[65]:
```
array([[14,  0,  0],
       [ 0, 16,  2],
       [ 0,  3, 10]])
```

Graficamente

In [66]:
```
ConfusionMatrixDisplay.from_predictions(target_test, predizioni)
```

Out[66]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x26f324a3c50>
```



Regressione

In [96]:
```
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso
```

Importiamo il dataset diamonds

Hadley Wickham. ggplot2: Elegant Graphics for Data Analysis. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. Disponibile online: https://ggplot2.tidyverse.org

In [97]:
```
diamonds=sns.load_dataset("diamonds")
```

In [98]:
```
diamonds.head(5)
```

Out[98]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

features:

- carat peso del diamante
- cut è la qualità del taglio secondo questa scala Fair, Good, Very Good, Premium, Ideal
- color riguarda il colore secondo questa scala J, I, H, G, F, E, D
- clarity la chiarezza secondo questa scala I3, I2, I1, SI2, SI1, VS2, VS1, VVS2, VVS1, IF, FL
- depth altezza in percentuale
- table larghezza in percentuale
- x lunghezza in millimetri
- y largheza in millimetri
- z prodonfita in millimetri

target:

- price

Codifichiamo le variabili non quantitative (sono comunque tutte ordinali).

In [99]:
```
codifica_cut = {"Fair":1, "Good":2, "Very Good":3, "Premium":4, "Ideal":5}
codifica_color = {"J":1, "I":2, "H":3, "G":4, "F":5, "E":6, "D":7}
codifica_clarity = {"I3":1, "I2": 2, "I1":3, "SI2":4, "SI1":5, "VS2":6, "VS1":7, "VVS2":8, "VVS1":9, "IF":10, "FL":11}
```

In [100…
```
diamonds["cut"] = diamonds["cut"].map(codifica_cut).astype("int64")
diamonds["color"] = diamonds["color"].map(codifica_color).astype("int64")
diamonds["clarity"] = diamonds["clarity"].map(codifica_clarity).astype("int64")
```

Vediamo il nuovo dataset

In [101…
```
diamonds.head(5)
```

Out[101…

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	5	6	4	61.5	55.0	326	3.95	3.98	2.43
1	0.21	4	6	5	59.8	61.0	326	3.89	3.84	2.31
2	0.23	2	6	7	56.9	65.0	327	4.05	4.07	2.31
3	0.29	4	2	6	62.4	58.0	334	4.20	4.23	2.63
4	0.31	2	1	4	63.3	58.0	335	4.34	4.35	2.75

Se ci fossero variabili non quantitative e non ordinali, potrei usare il metodo di pandas get_dummies()

In [102…
```
diamonds2=sns.load_dataset("diamonds")
```

In [103…
```
pd.get_dummies(data = diamonds2["color"], dtype=int).head(5)
```

Out[103…

	D	E	F	G	H	I	J
0	0	1	0	0	0	0	0
1	0	1	0	0	0	0	0
2	0	1	0	0	0	0	0
3	0	0	0	0	0	1	0
4	0	0	0	0	0	0	1

In [104…
```
diamonds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  int64  
 2   color    53940 non-null  int64  
 3   clarity  53940 non-null  int64  
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64  
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(4)
memory usage: 4.1 MB
```

Eseguiamo il metodo describe

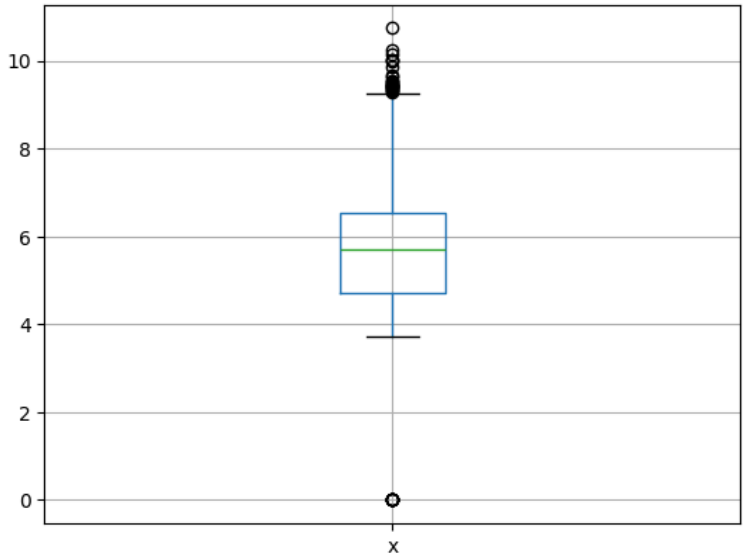
In [105…
```
diamonds.describe(include='all')
```

Out[105…

	carat	cut	color	clarity	depth	table	price	x	y
count	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000	53940.000000
mean	0.797940	3.904097	4.405803	6.051020	61.749405	57.457184	3932.799722	5.731157	5.734526
std	0.474011	1.116600	1.701105	1.647136	1.432621	2.234491	3989.439738	1.121761	1.142135
min	0.200000	1.000000	1.000000	3.000000	43.000000	43.000000	326.000000	0.000000	0.000000
25%	0.400000	3.000000	3.000000	5.000000	61.000000	56.000000	950.000000	4.710000	4.720000
50%	0.700000	4.000000	4.000000	6.000000	61.800000	57.000000	2401.000000	5.700000	5.710000
75%	1.040000	5.000000	6.000000	7.000000	62.500000	59.000000	5324.250000	6.540000	6.540000
max	5.010000	5.000000	7.000000	10.000000	79.000000	95.000000	18823.000000	10.740000	58.900000

In [106…
```
diamonds[["x"]].boxplot()
```

Out[106… <Axes: >



In [107…
```
len(diamonds.query("x==0"))
```

Out[107… 8

sostituiamo gli zero nelle colonne x, y e z con dei null

In [108…
```
diamonds = diamonds.replace(to_replace={'x':{0:np.nan},
                                        'y':{0:np.nan},
                                        'z':{0:np.nan}
                                })
```

ora sono presenti dei null

In [109…
```
diamonds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  int64  
 2   color    53940 non-null  int64  
 3   clarity  53940 non-null  int64  
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64  
 7   x        53932 non-null  float64
 8   y        53933 non-null  float64
 9   z        53920 non-null  float64
dtypes: float64(6), int64(4)
memory usage: 4.1 MB
```

In [110…
```
features_training,features_test, target_training, target_test  = train_test_split(diamonds.drop("price",axis=1),
                                                                                  diamonds[["price"]],
                                                                                  test_size = 0.3,
                                                                                  shuffle = True,
                                                                                  random_state = 1)
```

Creiamo una pipeline standard con i passi

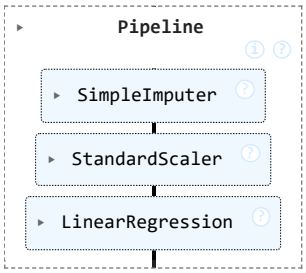
- simple imputer
- standard scaler
- regressione lineare

In [111…
```
basic_pipeline = Pipeline([('si', SimpleImputer()),
                           ('st', StandardScaler()),
                           ('rg', LinearRegression())
                          ])
```

Alleniamo la pipeline

In [112…
```
basic_pipeline.fit(features_training,target_training)
```

Out[112…



Vediamo le predizioni

In [113…
```
predizioni = basic_pipeline.predict(features_test)
predizioni
```

Out[113…
```
array([[-761.054826  ],
       [6670.89949191],
       [3735.28640372],
       ...,
       [ 224.659066  ],
       [1568.25785906],
       [4225.6362373 ]], shape=(16182, 1))
```

Analizziamo lo score

In [114…
```
basic_pipeline.score(features_test,target_test)
```

Out[114…
```
0.9090518474707515
```

Ricomponiamo il Dataset completo

In [115…
```
diamonds_completo = features_test.copy()
diamonds_completo["prezzo_originale"]=target_test["price"]
diamonds_completo["prezzo_predetto"]=predizioni
diamonds_completo.head(10)
```

Out[115...

	carat	cut	color	clarity	depth	table	x	y	z	prezzo_originale	prezzo_predetto
2714	0.33	5	3	5	61.7	55.0	4.43	4.46	2.74	564	-761.054826
14653	1.20	5	2	6	62.1	57.0	6.78	6.71	4.19	5914	6670.899492
52760	0.62	5	7	7	61.0	57.0	5.51	5.54	3.37	2562	3735.286404
48658	0.34	2	3	6	63.1	56.0	4.41	4.46	2.80	537	-627.724262
14812	1.20	5	6	4	62.5	55.0	6.77	6.84	4.25	5964	6990.900456
37498	0.53	4	6	4	62.2	57.0	5.20	5.16	3.22	984	984.845853
12456	1.08	5	7	4	60.2	57.0	6.63	6.67	4.00	5247	6271.139770
16738	0.41	4	4	4	61.4	58.0	4.75	4.80	2.93	611	-529.595352
21542	1.50	4	4	5	61.2	58.0	7.34	7.44	4.52	9645	9485.405304
40732	0.35	5	7	8	60.8	56.0	4.58	4.60	2.79	1162	2173.415451

Clustering

In [116...
```
from sklearn.cluster import KMeans
```

Citazione:Hind, Philip. Encyclopedia Titanica.

In [117...
```
titanic=sns.load_dataset("titanic")
```

eliminiamo feature ridondanti e correlate

In [118...
```
titanic = titanic.drop(["alive","pclass","embarked","deck","adult_male","sex","alone"],axis=1)
```

In [119...
```
titanic.head()
```

Out[119...

	survived	age	sibsp	parch	fare	class	who	embark_town
0	0	22.0	1	0	7.2500	Third	man	Southampton
1	1	38.0	1	0	71.2833	First	woman	Cherbourg
2	1	26.0	0	0	7.9250	Third	woman	Southampton
3	1	35.0	1	0	53.1000	First	woman	Southampton
4	0	35.0	0	0	8.0500	Third	man	Southampton

Decodifichiamo le variabili non quantitative

In [120...
```
codifica_class = {"First":1, "Second":2, "Third":3}
titanic["class"] = titanic["class"].map(codifica_class).astype("int64")
```

In [121...
```
titanic_2 = pd.get_dummies(data=titanic, dtype=int)
titanic_2.head(5)
```

Out[121...

	survived	age	sibsp	parch	fare	class	who_child	who_man	who_woman	embark_town_Cherbourg	embark_town_Queenstow
0	0	22.0	1	0	7.2500	3	0	1	0	0	
1	1	38.0	1	0	71.2833	1	0	0	1	1	
2	1	26.0	0	0	7.9250	3	0	0	1	0	
3	1	35.0	1	0	53.1000	1	0	0	1	0	
4	0	35.0	0	0	8.0500	3	0	1	0	0	

Valorizziamo i null

In [122...
```
imputer = SimpleImputer(strategy='mean')
```

In [123...
```
imputer.fit(titanic_2)
```

Out[123...

▾ SimpleImputer

```
SimpleImputer()
```

In [124...
```
titanic_2_imputed = pd.DataFrame(
    imputer.transform(titanic_2),
```

```
    columns=titanic_2.columns,
    index=titanic_2.index
)
```

Istanziamo l'algoritmo Kmeans di Scikit-learn

In [125…
```
km = KMeans (n_clusters = 2,
             n_init = 5,
             random_state = 0)
```

Eseguiamo il fit e predict su tutti i dati tranne survived

In [126…
```
cluster = km.fit_predict(titanic_2_imputed.drop("survived",axis=1))
cluster[:5]
```

Out[126…
```
array([0, 0, 0, 0, 0], dtype=int32)
```

Creiamo la colonna "cluster"

In [127…
```
titanic["cluster"] = cluster
```

In [128…
```
titanic.head(10)
```

Out[128…

	survived	age	sibsp	parch	fare	class	who	embark_town	cluster
0	0	22.0	1	0	7.2500	3	man	Southampton	0
1	1	38.0	1	0	71.2833	1	woman	Cherbourg	0
2	1	26.0	0	0	7.9250	3	woman	Southampton	0
3	1	35.0	1	0	53.1000	1	woman	Southampton	0
4	0	35.0	0	0	8.0500	3	man	Southampton	0
5	0	NaN	0	0	8.4583	3	man	Queenstown	0
6	0	54.0	0	0	51.8625	1	man	Southampton	0
7	0	2.0	3	1	21.0750	3	child	Southampton	0
8	1	27.0	0	2	11.1333	3	woman	Southampton	0
9	1	14.0	1	0	30.0708	2	child	Cherbourg	0

In questo caso i due cluster non sembrano corrispondere con la suddivisione tra survived e non survived.

Tuttavia la suddivisione in clustered ha senso compiuto anche se non applicata ad un problema di classificazione