In [1]:

```
import numpy as np
import pandas as pd
```

identify and detect null value

- **True**

In [2]:

```
pd.isnull(np.nan)
```

Out[2]:

```
True
```

In [3]:

```
pd.isna(None)
```

Out[3]:

```
True
```

In [4]:

```
pd.isna(np.nan)
```

Out[4]:

```
True
```

In [7]:

```
pd.isna(3)
```

Out[7]:

```
False
```

- **false**

In [5]:

```
pd.notnull(np.nan)
```

Out[5]:

```
False
```

In [6]:

```
pd.notna(None)
```

Out[6]:

```
False
```

In [8]:

```
pd.notna(3)
```

Out[8]:

```
True
```

Series

In [10]:

```
S = pd.Series([1 , 0 , np.nan , 5])
pd.notna(S)
```

Out[10]:

```
0     True
1     True
2    False
3     True
dtype: bool
```

DataFrame

In [24]:

```
df = pd.DataFrame({"A":[1 , "cat" , np.nan ] ,
                   "B":[0 , np.nan , "dog"] ,
                  "C":[4 , 7 ,"monkey"] ,
                  "D":[5 , 8 , 3]})
df
```

Out[24]:

	A	B	C	D
0	1	0	4	5
1	cat	NaN	7	8
2	NaN	dog	monkey	3

In [25]:

```
pd.notna(df)
```

Out[25]:

	A	B	C	D
0	True	True	True	True
1	True	False	True	True
2	False	True	True	True

pandas operations with missing values

In [26]:

```
df.count()  # to know missing value
```

Out[26]:

```
A    2
B    2
C    3
D    3
dtype: int64
```

In [27]:

```
pd.Series([1, 2, np.nan]).sum()
```

Out[27]:

```
3.0
```

In [28]:

```
df.mean() #work when the columns is numerical
```

Out[28]:

```
D    5.333333
dtype: float64
```

In [29]:

```
df.sum()  #work when the columns is numerical
```

Out[29]:

```
D    16
dtype: int64
```

filtering missing data(nan)

In [30]:

```
pd.notnull(df)
```

Out[30]:

	A	B	C	D
0	True	True	True	True
1	True	False	True	True
2	False	True	True	True

In [31]:

```
pd.notna(df).sum()
```

Out[31]:

```
A    2
B    2
C    3
D    3
dtype: int64
```

In [32]:

```
pd.isna(df).sum()
```

Out[32]:

```
A    1
B    1
C    0
D    0
dtype: int64
```

In [36]:

```
df[pd.notna(df)]
```

Out[36]:

	A	B	C	D
0	1	0	4	5
1	cat	NaN	7	8
2	NaN	dog	monkey	3

	A	B	C	D
2	NaN	dog	monkey	3

Dropping null values

In [37]:

```
df.dropna()   # drop any row has nan
```

Out[37]:

	A	B	C	D
0	1	0	4	5

In [38]:

```
df.dropna(axis=1) # drop any column has nan
```

Out[38]:

	C	D
0	4	5
1	7	8
2	monkey	3

In [39]:

```
df["E"] = [np.nan , np.nan , np.nan]  #add column
df
```

Out[39]:

	A	B	C	D	E
0	1	0	4	5	NaN
1	cat	NaN	7	8	NaN
2	NaN	dog	monkey	3	NaN

In [64]:

```
df.loc[3] = [np.nan , np.nan , np.nan , np.nan , np.nan]  #add row
df
```

Out[64]:

	A	B	C	D	E
0	1	0	4	5.0	NaN
1	cat	NaN	7	8.0	NaN
2	NaN	dog	monkey	3.0	NaN
3	NaN	NaN	NaN	NaN	NaN

In [65]:

```
df.dropna(how="all")  # drop the row when it's value is nan (3)
```

Out[65]:

	A	B	C	D	E
0	1	0	4	5.0	NaN
1	cat	NaN	7	8.0	NaN

	A	B	C	D	E
2	NaN	dog	monkey	3.0	NaN

In [66]:

```
df.dropna(axis=1 , how="all")  #drop the coulmn when it's value is nan (E)
```

Out[66]:

	A	B	C	D
0	1	0	4	5.0
1	cat	NaN	7	8.0
2	NaN	dog	monkey	3.0
3	NaN	NaN	NaN	NaN

In [67]:

```
df.dropna(axis=0 , how="any") # as all rows have nans
```

Out[67]:

	A	B	C	D	E

In [68]:

```
df.dropna(axis=1 , how="any")  # as all coulmns have nans
```

Out[68]:

0
1
2
3

In [70]:

```
df.dropna(axis=0 , thresh=4)  # only rows that have 4 value
```

Out[70]:

	A	B	C	D	E
0	1	0	4	5.0	NaN

In [71]:

```
df.dropna(axis=1 , thresh=3) # only coumns that have 3 value
```

Out[71]:

	C	D
0	4	5.0
1	7	8.0
2	monkey	3.0
3	NaN	NaN

filling null values

In [72]:

```
df
```

Out[72]:

	A	B	C	D	E
0	1	0	4	5.0	NaN
1	cat	NaN	7	8.0	NaN
2	NaN	dog	monkey	3.0	NaN
3	NaN	NaN	NaN	NaN	NaN

In [74]:

```
df.fillna({"A":0})   #replace nan value with 0 in A column
```

Out[74]:

	A	B	C	D	E
0	1	0	4	5.0	NaN
1	cat	NaN	7	8.0	NaN
2	0	dog	monkey	3.0	NaN
3	0	NaN	NaN	NaN	NaN

In [77]:

```
df.mean()
```

Out[77]:

```
D    5.333333
E         NaN
dtype: float64
```

In [79]:

```
df["E"].fillna(df.mean()[0])
```

Out[79]:

```
0    5.333333
1    5.333333
2    5.333333
3    5.333333
Name: E, dtype: float64
```

In [80]:

```
df
```

Out[80]:

	A	B	C	D	E
0	1	0	4	5.0	NaN
1	cat	NaN	7	8.0	NaN
2	NaN	dog	monkey	3.0	NaN
3	NaN	NaN	NaN	NaN	NaN

Filling nulls with contiguous (close) values

. The method argument is used to fill null values with other values close to that null one:

In [82]:

In [83]:

```
df.fillna(method="ffill") # the value up the nan
```

Out[83]:

	A	B	C	D	E
0	1	0	4	5.0	NaN
1	cat	0	7	8.0	NaN
2	cat	dog	monkey	3.0	NaN
3	cat	dog	monkey	3.0	NaN

In [84]:

```
df.fillna(method="bfill") #the value down the nan
```

Out[84]:

	A	B	C	D	E
0	1	0	4	5.0	NaN
1	cat	dog	7	8.0	NaN
2	NaN	dog	monkey	3.0	NaN
3	NaN	NaN	NaN	NaN	NaN

In [85]:

```
df
```

Out[85]:

	A	B	C	D	E
0	1	0	4	5.0	NaN
1	cat	NaN	7	8.0	NaN
2	NaN	dog	monkey	3.0	NaN
3	NaN	NaN	NaN	NaN	NaN

In [88]:

```
df.fillna({"A":df["A"].fillna(method="ffill") , "B":df["B"].fillna(method="bfill") ,
           "C":0 , "D":df["D"].fillna(df["D"].mean())})
```

Out[88]:

	A	B	C	D	E
0	1	0	4	5.000000	NaN
1	cat	dog	7	8.000000	NaN
2	cat	dog	monkey	3.000000	NaN
3	cat	NaN	0	5.333333	NaN

In [89]:

```
df.fillna(method="ffill" , axis=1) # the value in the previous column
```

Out[89]:

	A	B	C	D	E
0	1.0	0.0	4.0	5.0	5.0
1	cat	cat	7	8.0	8.0

	A	B	C	D	E
2	NaN	dog	monkey	3.0	3.0
3	NaN	NaN	NaN	NaN	NaN

In [90]:

```
df.fillna(axis=1 , method="bfill")
```

Out[90]:

	A	B	C	D	E
0	1.0	0.0	4.0	5.0	NaN
1	cat	7	7	8.0	NaN
2	dog	dog	monkey	3.0	NaN
3	NaN	NaN	NaN	NaN	NaN

In [94]:

```
technologies = [{'Courses':'Spark', 'Fee': 20000, 'Duration':'30days'},
        {'Courses':'Pandas', 'Fee': 25000, 'Duration': '40days'}]

df = pd.DataFrame(technologies)
df
```

Out[94]:

	Courses	Fee	Duration
0	Spark	20000	30days
1	Pandas	25000	40days

2. How do I read a tabular data file into pandas?

In [1]:

```
# read a dataset of Chipotle orders directly from a URL and store the results in a DataFr
ame

df = pd.read_table("http://bit.ly/chiporders")
```

In [3]:

```
df.head(10)
```

Out[3]:

	order_id	quantity	item_name	choice_description	item_price
0	1	1	Chips and Fresh Tomato Salsa	NaN	$2.39
1	1	1	Izze	[Clementine]	$3.39
2	1	1	Nantucket Nectar	[Apple]	$3.39
3	1	1	Chips and Tomatillo-Green Chili Salsa	NaN	$2.39
4	2	2	Chicken Bowl	[Tomatillo-Red Chili Salsa (Hot), [Black Beans...	$16.98
5	3	1	Chicken Bowl	[Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou...	$10.98
6	3	1	Side of Chips	NaN	$1.69
7	4	1	Steak Burrito	[Tomatillo Red Chili Salsa, [Fajita Vegetables...	$11.75
8	4	1	Steak Soft Tacos	[Tomatillo Green Chili Salsa, [Pinto Beans, Ch...	$9.25
9	5	1	Steak Burrito	[Fresh Tomato Salsa, [Rice, Black Beans, Pinto...	$9.25

In [7]:

```
# read a dataset of movie reviewers (modifying the default parameter values for read_table)
user_cols = ["user_id" , "age" , "gender" , "occupation" , "zip_code"]
df_user = pd.read_table("http://bit.ly/movieusers" , sep="|" , header=None , names = user_cols)
df_user.head()
```

Out[7]:

	user_id	age	gender	occupation	zip_code
0	1	24	M	technician	85711
1	2	53	F	other	94043
2	3	23	M	writer	32067
3	4	24	M	technician	43537
4	5	33	F	other	15213

In [8]:

```
# read_csv is equivalent to read_table, except it assumes a comma separator
df = pd.read_csv("http://bit.ly/uforeports")
df.head()
```

Out[8]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00
3	Abilene	NaN	DISK	KS	6/1/1931 13:00
4	New York Worlds Fair	NaN	LIGHT	NY	4/18/1933 19:00

In [9]:

```
# select the 'City' Series using bracket notation
df["City"]
```

Out[9]:

```
0                       Ithaca
1                  Willingboro
2                      Holyoke
3                      Abilene
4           New York Worlds Fair
                   ...
18236                Grant Park
18237              Spirit Lake
18238              Eagle River
18239              Eagle River
18240                     Ybor
Name: City, Length: 18241, dtype: object
```

In [10]:

```
# or equivalently, use dot notation
df.City
```

Out[10]:

```
0                      Ithaca
1                 Willingboro
2                     Holyoke
3                     Abilene
4        New York Worlds Fair
                ...
18236              Grant Park
18237             Spirit Lake
18238             Eagle River
18239             Eagle River
18240                    Ybor
Name: City, Length: 18241, dtype: object
```

- **Dot notation doesn't work if there are spaces in the Series name**
- **Dot notation doesn't work if the Series has the same name as a DataFrame method or attribute (like 'head' or 'shape')**
- **Dot notation can't be used to define the name of a new Series (see below)**

In [12]:

```
df.describe()  # example method: calculate summary "object" columns
```

Out[12]:

	City	Colors Reported	Shape Reported	State	Time
count	18216	2882	15597	18241	18241
unique	6476	27	27	52	16145
top	Seattle	RED	LIGHT	CA	11/16/1999 19:00
freq	187	780	2803	2529	27

In [14]:

```
df.shape  # example attribute: number of rows and columns
```

Out[14]:

```
(18241, 5)
```

In [15]:

```
df.dtypes
```

Out[15]:

```
City               object
Colors Reported    object
Shape Reported     object
State              object
Time               object
dtype: object
```

In [16]:

```
df.columns
```

Out[16]:

```
Index(['City', 'Colors Reported', 'Shape Reported', 'State', 'Time'], dtype='object')
```

In [17]:

```
# rename two of the columns by using the 'rename' method
df.rename(columns={"Colors Reported":"Colors_Reported" , "Shape Reported":"Shape_Reporte
d" ,
                   "State" : "State" , "Time" : "Time"} , inplace=True)
```

In [18]:

```
df
```

Out[18]:

	City	Colors_Reported	Shape_Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00
3	Abilene	NaN	DISK	KS	6/1/1931 13:00
4	New York Worlds Fair	NaN	LIGHT	NY	4/18/1933 19:00
...	...	...	...	...	...
18236	Grant Park	NaN	TRIANGLE	IL	12/31/2000 23:00
18237	Spirit Lake	NaN	DISK	IA	12/31/2000 23:00
18238	Eagle River	NaN	NaN	WI	12/31/2000 23:45
18239	Eagle River	RED	LIGHT	WI	12/31/2000 23:45
18240	Ybor	NaN	OVAL	FL	12/31/2000 23:59

18241 rows × 5 columns

In [19]:

```
# replace all of the column names by overwriting the 'columns' attribute
cols = ['city', 'colors reported', 'shape reported', 'state', 'time']
df.columns = cols
df.columns
```

Out[19]:

```
Index(['city', 'colors reported', 'shape reported', 'state', 'time'], dtype='object')
```

6. How do I remove columns from a pandas DataFrame?

In [24]:

```
df.drop("colors reported" , axis=1 , inplace=True)    # remove a single column (axis=1 refers to columns)
df.head()
```

Out[24]:

	city	shape reported	state	time
0	Ithaca	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	OTHER	NJ	6/30/1930 20:00
2	Holyoke	OVAL	CO	2/15/1931 14:00
3	Abilene	DISK	KS	6/1/1931 13:00
4	New York Worlds Fair	LIGHT	NY	4/18/1933 19:00

In [25]:

```
# remove multiple columns at once
```

```
df.drop(["city" , "state"] , axis=1 , inplace=True)
df.head()
```

Out[25]:

	shape reported	time
0	TRIANGLE	6/1/1930 22:00
1	OTHER	6/30/1930 20:00
2	OVAL	2/15/1931 14:00
3	DISK	6/1/1931 13:00
4	LIGHT	4/18/1933 19:00

In [26]:

```
# remove multiple rows at once(axis=0 refers to row)
df.drop([0 , 1] , inplace=True)
```

In [27]:

```
df.head()
```

Out[27]:

	shape reported	time
2	OVAL	2/15/1931 14:00
3	DISK	6/1/1931 13:00
4	LIGHT	4/18/1933 19:00
5	DISK	9/15/1934 15:30
6	CIRCLE	6/15/1935 0:00

In [2]:

```
df = pd.read_csv('http://bit.ly/imdbratings')
df.head()
```

Out[2]:

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....

In [29]:

```
df.title.sort_values()
```

Out[29]:

```
542      (500) Days of Summer
5                  12 Angry Men
201          12 Years a Slave
```

```
698                 127 Hours
110    2001: A Space Odyssey
                ...
955          Zero Dark Thirty
677                    Zodiac
615                Zombieland
526                      Zulu
864                    [Rec]
Name: title, Length: 979, dtype: object
```

In [31]:

```
df.sort_values('title').head(20)
```

Out[31]:

	star_rating	title	content_rating	genre	duration	actors_list
542	7.8	(500) Days of Summer	PG-13	Comedy	95	[u'Zooey Deschanel', u'Joseph Gordon-Levitt', ...
5	8.9	12 Angry Men	NOT RATED	Drama	96	[u'Henry Fonda', u'Lee J. Cobb', u'Martin Bals...
201	8.1	12 Years a Slave	R	Biography	134	[u'Chiwetel Ejiofor', u'Michael Kenneth Willia...
698	7.6	127 Hours	R	Adventure	94	[u'James Franco', u'Amber Tamblyn', u'Kate Mara']
110	8.3	2001: A Space Odyssey	G	Mystery	160	[u'Keir Dullea', u'Gary Lockwood', u'William S...
910	7.5	2046	R	Drama	129	[u'Tony Chiu Wai Leung', u'Ziyi Zhang', u'Faye...
596	7.7	21 Grams	R	Crime	124	[u'Sean Penn', u'Benicio Del Toro', u'Naomi Wa...
624	7.7	25th Hour	R	Crime	135	[u'Edward Norton', u'Barry Pepper', u'Philip S...
708	7.6	28 Days Later...	R	Horror	113	[u'Cillian Murphy', u'Naomie Harris', u'Christ...
60	8.5	3 Idiots	PG-13	Comedy	170	[u'Aamir Khan', u'Madhavan', u'Mona Singh']
225	8.1	3-Iron	R	Crime	88	[u'Seung-yeon Lee', u'Hyun-kyoon Lee', u'Hyuk-...
570	7.8	300	R	Action	117	[u'Gerard Butler', u'Lena Headey', u'David Wen...
555	7.8	3:10 to Yuma	R	Adventure	122	[u'Russell Crowe', u'Christian Bale', u'Ben Fo...
427	7.9	4 Months, 3 Weeks and 2 Days	NOT RATED	Drama	113	[u'Anamaria Marinca', u'Vlad Ivanov', u'Laura ...
824	7.5	42	PG-13	Biography	128	[u'Chadwick Boseman', u'T.R. Knight', u'Harris...
597	7.7	50/50	R	Comedy	100	[u'Joseph Gordon-Levitt', u'Seth Rogen', u'Ann...
203	8.1	8 1/2	NOT RATED	Drama	138	[u'Marcello Mastroianni', u'Anouk Aim\xe9e', u...
170	8.2	A Beautiful Mind	PG-13	Biography	135	[u'Russell Crowe', u'Ed Harris', u'Jennifer Co...
941	7.4	A Bridge Too Far	PG	Drama	175	[u'Sean Connery', u"Ryan O'Neal", u'Michael Ca...
571	7.8	A Bronx Tale	R	Crime	121	[u'Robert De Niro', u'Chazz Palminteri', u'Lil...

In [37]:

```
# sort the DataFrame first by 'content_rating', then by 'duration'
```

```
df.sort_values(['content_rating', 'duration']).head(10)
```

Out[37]:

	star_rating	title	content_rating	genre	duration	actors_list
713	7.6	The Jungle Book	APPROVED	Animation	78	[u'Phil Harris', u'Sebastian Cabot', u'Louis P...
513	7.8	Invasion of the Body Snatchers	APPROVED	Horror	80	[u'Kevin McCarthy', u'Dana Wynter', u'Larry Ga...
272	8.1	The Killing	APPROVED	Crime	85	[u'Sterling Hayden', u'Coleen Gray', u'Vince E...
703	7.6	Dracula	APPROVED	Horror	85	[u'Bela Lugosi', u'Helen Chandler', u'David Ma...
612	7.7	A Hard Day's Night	APPROVED	Comedy	87	[u'John Lennon', u'Paul McCartney', u'George H...
58	8.5	Paths of Glory	APPROVED	Drama	88	[u'Kirk Douglas', u'Ralph Meeker', u'Adolphe M...
210	8.1	Laura	APPROVED	Film-Noir	88	[u'Gene Tierney', u'Dana Andrews', u'Clifton W...
656	7.7	Snow White and the Seven Dwarfs	APPROVED	Animation	88	[u'Adriana Caselotti', u'Harry Stockwell', u'L...
844	7.5	Pinocchio	APPROVED	Animation	88	[u'Dickie Jones', u'Christian Rub', u'Mel Blanc']
233	8.1	The Night of the Hunter	APPROVED	Crime	92	[u'Robert Mitchum', u'Shelley Winters', u'Lill...

8. How do I filter rows of a pandas DataFrame by column value?

In [3]:

```
df[df.duration >=200]  # or equivalently, write it in one line (no need to create the 'is_long' object)
```

Out[3]:

	star_rating	title	content_rating	genre	duration	actors_list
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
7	8.9	The Lord of the Rings: The Return of the King	PG-13	Adventure	201	[u'Elijah Wood', u'Viggo Mortensen', u'Ian McK...
17	8.7	Seven Samurai	UNRATED	Drama	207	[u'Toshir\xf4 Mifune', u'Takashi Shimura', u'K...
78	8.4	Once Upon a Time in America	R	Crime	229	[u'Robert De Niro', u'James Woods', u'Elizabet...
85	8.4	Lawrence of Arabia	PG	Adventure	216	[u"Peter O'Toole", u'Alec Guinness', u'Anthony...
142	8.3	Lagaan: Once Upon a Time in India	PG	Adventure	224	[u'Aamir Khan', u'Gracy Singh', u'Rachel Shell...
157	8.2	Gone with the Wind	G	Drama	238	[u'Clark Gable', u'Vivien Leigh', u'Thomas Mit...
204	8.1	Ben-Hur	G	Adventure	212	[u'Charlton Heston', u'Jack Hawkins', u'Stephe...
445	7.9	The Ten Commandments	APPROVED	Adventure	220	[u'Charlton Heston', u'Yul Brynner', u'Anne Ba...
476	7.8	Hamlet	PG-13	Drama	242	[u'Kenneth Branagh', u'Julie Christie', u'Dere...

	star_rating	title	content_rating	genre	duration	actors_list
630	7.7	Malcolm X	PG-13	Biography	202	[u'Denzel Washington', u'Angela Bassett', u'De...
767	7.6	It's a Mad, Mad, Mad, Mad World	APPROVED	Action	205	[u'Spencer Tracy', u'Milton Berle', u'Ethel Me...

In [13]:

```
df.loc[df.duration <= 200, "genre"]
```

Out[13]:

```
0          Crime
1          Crime
2          Crime
3         Action
4          Crime
         ...
974       Comedy
975    Adventure
976       Action
977       Horror
978        Crime
Name: genre, Length: 968, dtype: object
```

9. How do I apply multiple filter criteria to a pandas DataFrame?

In [14]:

```
# demonstration of the 'and' operator
print(True and True)
print(True and False)
print(False and False)
```

```
True
False
False
```

In [15]:

```
# demonstration of the 'or' operator
print(True or True)
print(True or False)
print(False or False)
```

```
True
True
False
```

- **use & instead of and**
- **use | instead of or**

In [17]:

```
# CORRECT: use the '&' operator to specify that both conditions are required
df[(df.duration >= 200) & (df.genre == "Drama")].head()
```

Out[17]:

	star_rating	title	content_rating	genre	duration	actors_list
17	8.7	Seven Samurai	UNRATED	Drama	207	[u'Toshir\xf4 Mifune', u'Takashi Shimura', u'K...
157	8.2	Gone with the Wind	G	Drama	238	[u'Clark Gable', u'Vivien Leigh', u'Thomas Mit...
476	7.8	Hamlet	PG-13	Drama	242	[u'Kenneth Branagh', u'Julie Christie', u'Dere...

In [18]:

```
# INCORRECT: using the '|' operator would have shown movies that are either long or drama
s (or both)
df[(df.duration >= 200) | (df.genre == "Drama")].head()
```

Out[18]:

	star_rating	title	content_rating	genre	duration	actors_list
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
5	8.9	12 Angry Men	NOT RATED	Drama	96	[u'Henry Fonda', u'Lee J. Cobb', u'Martin Bals...
7	8.9	The Lord of the Rings: The Return of the King	PG-13	Adventure	201	[u'Elijah Wood', u'Viggo Mortensen', u'Ian McK...
9	8.9	Fight Club	R	Drama	139	[u'Brad Pitt', u'Edward Norton', u'Helena Bonh...
13	8.8	Forrest Gump	PG-13	Drama	142	[u'Tom Hanks', u'Robin Wright', u'Gary Sinise']

In [19]:

```
# or equivalently, use the 'isin' method
df[df.genre.isin(["Crime" , "Drama" , "Action"])]
```

Out[19]:

	star_rating	title	content_rating	genre	duration	actors_list
0	9.3	The Shawshank Redemption	R	Crime	142	[u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt...
1	9.2	The Godfather	R	Crime	175	[u'Marlon Brando', u'Al Pacino', u'James Caan']
2	9.1	The Godfather: Part II	R	Crime	200	[u'Al Pacino', u'Robert De Niro', u'Robert Duv...
3	9.0	The Dark Knight	PG-13	Action	152	[u'Christian Bale', u'Heath Ledger', u'Aaron E...
4	8.9	Pulp Fiction	R	Crime	154	[u'John Travolta', u'Uma Thurman', u'Samuel L....
...	...	...	...	...	...	...
970	7.4	Wonder Boys	R	Drama	107	[u'Michael Douglas', u'Tobey Maguire', u'Franc...
972	7.4	Blue Valentine	NC-17	Drama	112	[u'Ryan Gosling', u'Michelle Williams', u'John...
973	7.4	The Cider House Rules	PG-13	Drama	126	[u'Tobey Maguire', u'Charlize Theron', u'Micha...
976	7.4	Master and Commander: The Far Side of the World	PG-13	Action	138	[u'Russell Crowe', u'Paul Bettany', u'Billy Bo...
978	7.4	Wall Street	R	Crime	126	[u'Charlie Sheen', u'Michael Douglas', u'Tamar...

538 rows × 6 columns

In [35]:

```
df.describe()  # pass the string 'all' to describe all columns
```

Out[35]:

	star_rating	duration
count	979.000000	979.000000

	star_rating	duration
mean	7.889785	120.979571
std	0.336069	26.218010
min	7.400000	64.000000
25%	7.600000	102.000000
50%	7.800000	117.000000
75%	8.100000	134.000000
max	9.300000	242.000000

In [36]:

```
# pass a list even if you only want to describe a single data type
df.describe(include=['object'])
```

Out[36]:

	title	content_rating	genre	actors_list
count	979	976	979	979
unique	975	12	16	969
top	The Girl with the Dragon Tattoo	R	Drama	[u'Daniel Radcliffe', u'Emma Watson', u'Rupert...
freq	2	460	278	6

In [43]:

```
df.mean(axis=0)  # or equivalently, specify the axis explicitly
```

Out[43]:

```
star_rating      7.889785
duration       120.979571
dtype: float64
```

In [42]:

```
df.mean(axis=1).head()  # calculate the mean of each row
```

Out[42]:

```
0     75.65
1     92.10
2    104.55
3     80.50
4     81.45
dtype: float64
```

10. How do I change the data type of a pandas Series?

In [17]:

```
df = pd.read_csv("http://bit.ly/drinksbycountry")
df.dtypes
```

Out[17]:

```
country                          object
beer_servings                     int64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

In [51]:

```
df
```

Out[51]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0.0	0	0	0.0	Asia
1	Albania	89.0	132	54	4.9	Europe
2	Algeria	25.0	0	14	0.7	Africa
3	Andorra	245.0	138	312	12.4	Europe
4	Angola	217.0	57	45	5.9	Africa
...	...	...	...	...	...	...
188	Venezuela	333.0	100	3	7.7	South America
189	Vietnam	111.0	2	1	2.0	Asia
190	Yemen	6.0	0	0	0.1	Asia
191	Zambia	32.0	19	4	2.5	Africa
192	Zimbabwe	64.0	18	4	4.7	Africa

193 rows × 6 columns

In [46]:

```
df["beer_servings"] = df.beer_servings.astype(float)
df.dtypes
```

Out[46]:

```
country                          object
beer_servings                   float64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

11. When should I use a "groupby" in pandas?

In [52]:

```
# calculate the mean beer servings just for countries in Africa
df[df.continent == "Africa"].beer_servings.mean()
```

Out[52]:

```
61.471698113207545
```

In [53]:

```
# calculate the mean beer servings for each continent
df.groupby("continent").beer_servings.mean()
```

Out[53]:

```
continent
Africa             61.471698
Asia               37.045455
Europe            193.777778
North America     145.434783
Oceania            89.687500
South America     175.083333
Name: beer_servings, dtype: float64
```

In [54]:

```
# other aggregation functions (such as 'max') can also be used with groupby
```

```
df.groupby('continent').beer_servings.max()
```

Out[54]:

```
continent
Africa            376.0
Asia              247.0
Europe            361.0
North America     285.0
Oceania           306.0
South America     333.0
Name: beer_servings, dtype: float64
```

In [55]:

```
# multiple aggregation functions can be applied simultaneously
df.groupby('continent').beer_servings.agg(['count', 'mean', 'min', 'max'])
```

Out[55]:

continent	count	mean	min	max
Africa	53	61.471698	0.0	376.0
Asia	44	37.045455	0.0	247.0
Europe	45	193.777778	0.0	361.0
North America	23	145.434783	1.0	285.0
Oceania	16	89.687500	0.0	306.0
South America	12	175.083333	93.0	333.0

In [56]:

```
# specifying a column to which the aggregation function should be applied is not required
df.groupby('continent').mean()
```

Out[56]:

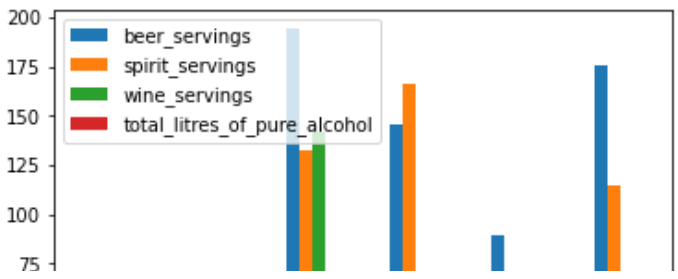
continent	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol
Africa	61.471698	16.339623	16.264151	3.007547
Asia	37.045455	60.840909	9.068182	2.170455
Europe	193.777778	132.555556	142.222222	8.617778
North America	145.434783	165.739130	24.521739	5.995652
Oceania	89.687500	58.437500	35.625000	3.381250
South America	175.083333	114.750000	62.416667	6.308333

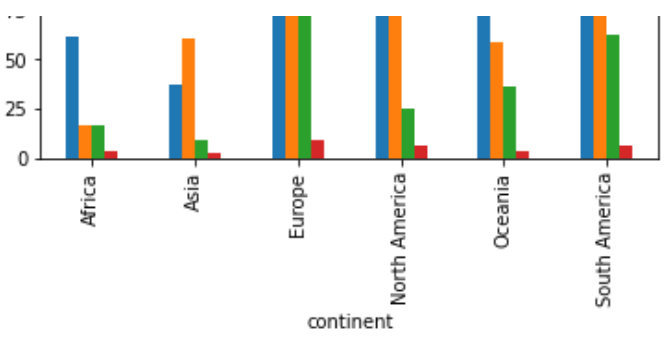
In [57]:

```
# side-by-side bar plot of the DataFrame directly above
df.groupby('continent').mean().plot(kind='bar')
```

Out[57]:

```
<AxesSubplot:xlabel='continent'>
```





In [63]:

```
df.continent.value_counts()
```

Out[63]:

```
Africa            53
Europe            45
Asia              44
North America     23
Oceania           16
South America     12
Name: continent, dtype: int64
```

In [60]:

```
# display percentages instead of raw counts
df.continent.value_counts(normalize=True)
```

Out[60]:

```
Africa            0.274611
Europe            0.233161
Asia              0.227979
North America     0.119171
Oceania           0.082902
South America     0.062176
Name: continent, dtype: float64
```

In [61]:

```
df.continent.unique()
```

Out[61]:

```
array(['Asia', 'Europe', 'Africa', 'North America', 'South America',
       'Oceania'], dtype=object)
```

In [64]:

```
# count the number of unique values in the Series
df.continent.nunique()
```

Out[64]:

```
6
```

12. What do I need to know about the pandas index? (Part 1)

In [3]:

```
# every DataFrame has an index (sometimes called the "row labels")
df.index
```

Out[3]:

```
RangeIndex(start=0, stop=193, step=1)
```

In [4]:

```
# column names are also stored in a special "index" object
df.columns
```

Out[4]:

```
Index(['country', 'beer_servings', 'spirit_servings', 'wine_servings',
       'total_litres_of_pure_alcohol', 'continent'],
      dtype='object')
```

In [5]:

```
# neither the index nor the columns are included in the shape
df.shape
```

Out[5]:

```
(193, 6)
```

What is the index used for?

- **identification**
- **selection**
- **alignment**

In [6]:

```
# identification: index remains with each row when filtering the DataFrame
df[df.continent == "South America"]
```

Out[6]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
6	Argentina	193	25	221	8.3	South America
20	Bolivia	167	41	8	3.8	South America
23	Brazil	245	145	16	7.2	South America
35	Chile	130	124	172	7.6	South America
37	Colombia	159	76	3	4.2	South America
52	Ecuador	162	74	3	4.2	South America
72	Guyana	93	302	1	7.1	South America
132	Paraguay	213	117	74	7.3	South America
133	Peru	163	160	21	6.1	South America
163	Suriname	128	178	7	5.6	South America
185	Uruguay	115	35	220	6.6	South America
188	Venezuela	333	100	3	7.7	South America

In [7]:

```
# selection: select a portion of the DataFrame using the index
df.loc[24 , "beer_servings"]
```

Out[7]:

```
31
```

In [12]:

```
df.beer_servings[df.country == "Brazil"]
```

Out[12]:

```
23    245
```

```
Name: beer_servings, dtype: int64
```

In [13]:

```
df.loc[df.country == "Brazil" , "beer_servings"]
```

Out[13]:

```
23    245
Name: beer_servings, dtype: int64
```

In [18]:

```
# set an existing column as the index
df.set_index("country" , inplace=True)
df.head()
```

Out[18]:

country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
Afghanistan	0	0	0	0.0	Asia
Albania	89	132	54	4.9	Europe
Algeria	25	0	14	0.7	Africa
Andorra	245	138	312	12.4	Europe
Angola	217	57	45	5.9	Africa

In [19]:

```
# country name can now be used for selection
df.loc['Brazil', 'beer_servings']
```

Out[19]:

```
245
```

In [20]:

```
# restore the index name, and move the index back to a column
df.reset_index(inplace=True)
df.head()
```

Out[20]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa
3	Andorra	245	138	312	12.4	Europe
4	Angola	217	57	45	5.9	Africa

In [24]:

```
df.describe()  # many DataFrame methods output a DataFrame
```

Out[24]:

	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol
count	193.000000	193.000000	193.000000	193.000000
mean	106.160622	80.994819	49.450777	4.717098
std	101.143103	88.284312	79.697598	3.773298
min	0.000000	0.000000	0.000000	0.000000

	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol
25%	20.000000	4.000000	1.000000	1.300000
50%	76.000000	56.000000	8.000000	4.200000
75%	188.000000	128.000000	59.000000	7.200000
max	376.000000	438.000000	370.000000	14.400000

In [30]:

```
df["beer_servings"].describe()["25%"]
```

Out[30]:

```
20.0
```

In [31]:

```
# you can interact with any DataFrame using its index and columns
df.describe().loc['25%', 'beer_servings']
```

Out[31]:

```
20.0
```

In [35]:

```
df.continent.value_counts()["Africa"]   # elements in a Series can be selected by index (
using bracket notation)
```

Out[35]:

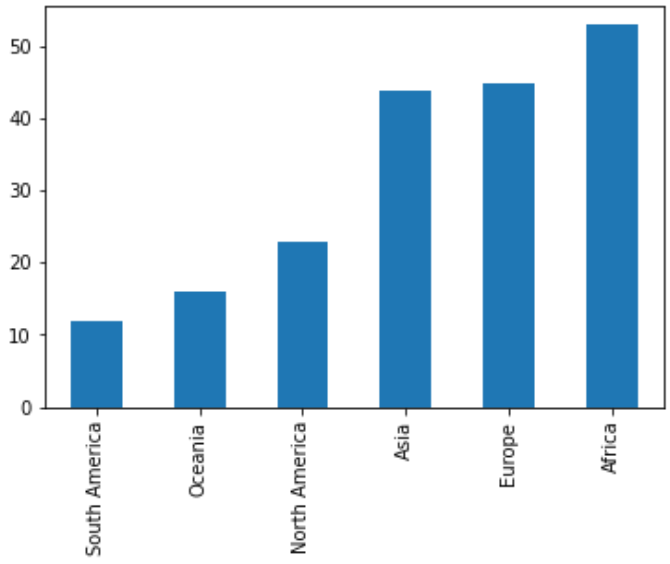
```
53
```

In [45]:

```
# any Series can be sorted by its values
df.continent.value_counts().sort_values().plot(kind="bar")
```

Out[45]:

```
<AxesSubplot:>
```



In [47]:

```
people = pd.Series([300000 , 85000] , name="population")
people
```

Out[47]:

```
0    300000
1     85000
Name: population, dtype: int64
```

In [49]:

```
pd.concat([df , people] , axis=1).head() # concatenate the 'drinks' DataFrame with the 'population' Series (aligns by the index)
```

Out[49]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent	population
0	Afghanistan	0	0	0	0.0	Asia	300000.0
1	Albania	89	132	54	4.9	Europe	85000.0
2	Algeria	25	0	14	0.7	Africa	NaN
3	Andorra	245	138	312	12.4	Europe	NaN
4	Angola	217	57	45	5.9	Africa	NaN

13. How do I select multiple rows and columns from a pandas DataFrame?

The loc method is used to select rows and columns by label. You can pass it:

- A single label
- A list of labels
- A slice of labels
- A boolean Series
- A colon (which indicates "all labels")

In [50]:

```
# row 0, all columns
df.loc[0 , :]
```

Out[50]:

```
country                         Afghanistan
beer_servings                             0
spirit_servings                           0
wine_servings                             0
total_litres_of_pure_alcohol            0.0
continent                              Asia
Name: 0, dtype: object
```

In [51]:

```
# rows 0 and 1 and 2, all columns
df.loc[[0, 1, 2], :]
```

Out[51]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa

In [52]:

```
# rows 0 through 2 (inclusive), all columns
df.loc[[0,1,2] , :]
```

Out[52]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa

In [53]:

```
# this implies "all columns", but explicitly stating "all columns" is better
df.loc[0 : 2]
```

Out[53]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa

In [54]:

```
# rows 0 through 2 (inclusive), column 'continent' , "country"
df.loc[0 : 2 , ["country" , "continent"]]
```

Out[54]:

	country	continent
0	Afghanistan	Asia
1	Albania	Europe
2	Algeria	Africa

In [55]:

```
# accomplish the same thing using double brackets - but using 'loc' is preferred since it
's more explicit
df[['country', 'continent']].head(3)
```

Out[55]:

	country	continent
0	Afghanistan	Asia
1	Albania	Europe
2	Algeria	Africa

In [56]:

```
# rows 0 through 2 (inclusive), columns 'country' through 'continent' (inclusive)
df.loc[0 : 2 , "country" : "continent"]
```

Out[56]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa

In [57]:

```
df.loc[df.country == "Brazil" , "continent"]  # rows in which the 'country' is 'Brazil',
```

```
column 'continent'
```

Out[57]:

```
23    South America
Name: continent, dtype: object
```

In [110]:

```
# 'iloc' is simply following NumPy's slicing convention...
df.values[0 : 4 , :]
```

Out[110]:

```
array([['Afghanistan', 0, 0, 0.0, 'Asia'],
       ['Albania', 89, 54, 4.9, 'Europe'],
       ['Algeria', 25, 14, 0.7, 'Africa'],
       ['Andorra', 245, 312, 12.4, 'Europe']], dtype=object)
```

The iloc method is used to select rows and columns by integer position. You can pass it:

- **A single integer position**
- **A list of integer positions**
- **A slice of integer positions**
- **A colon (which indicates "all integer positions")**

In [58]:

```
# rows in positions 0 and 1, columns in positions 0 and 3
df.iloc[[0, 1], [0, 3]]
```

Out[58]:

	country	wine_servings
0	Afghanistan	0
1	Albania	54

In [59]:

```
# rows in positions 0 through 2 (exclusive), columns in positions 0 through 4 (exclusive)
df.iloc[0:2, 0:4]
```

Out[59]:

	country	beer_servings	spirit_servings	wine_servings
0	Afghanistan	0	0	0
1	Albania	89	132	54

In [60]:

```
# rows in positions 0 through 2 (exclusive), all columns
df.iloc[0:2, :]
```

Out[60]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe

In [61]:

```
# accomplish the same thing - but using 'iloc' is preferred since it's more explicit
```

```
df[0:2]
```

Out[61]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia
1	Albania	89	132	54	4.9	Europe

14. When should I use the "inplace" parameter in pandas?

In [69]:

```
# remove the 'spirit_servings' column (does affect the DataFrame since inplace=True)
df.drop("spirit_servings" , axis=1 , inplace=True)
df.tail()
```

Out[69]:

	country	beer_servings	wine_servings	total_litres_of_pure_alcohol	continent
188	Venezuela	333	3	7.7	South America
189	Vietnam	111	1	2.0	Asia
190	Yemen	6	0	0.1	Asia
191	Zambia	32	4	2.5	Africa
192	Zimbabwe	64	4	4.7	Africa

In [70]:

```
# drop a row if any value is missing from that row (doesn't affect the DataFrame since in
place=False)
df.dropna(how='any').shape
```

Out[70]:

```
(193, 5)
```

In [71]:

```
# fill missing values using "backward fill" strategy (doesn't affect the DataFrame since
inplace=False)
df.fillna(method='bfill').tail()
```

Out[71]:

	country	beer_servings	wine_servings	total_litres_of_pure_alcohol	continent
188	Venezuela	333	3	7.7	South America
189	Vietnam	111	1	2.0	Asia
190	Yemen	6	0	0.1	Asia
191	Zambia	32	4	2.5	Africa
192	Zimbabwe	64	4	4.7	Africa

15. How do I make my pandas DataFrame smaller and faster?

In [72]:

```
# exact memory usage is unknown because object columns are references elsewhere
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193 entries, 0 to 192
Data columns (total 5 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   country                       193 non-null    object
 1   beer_servings                 193 non-null    int64
 2   wine_servings                 193 non-null    int64
 3   total_litres_of_pure_alcohol  193 non-null    float64
 4   continent                     193 non-null    object
dtypes: float64(1), int64(2), object(2)
memory usage: 7.7+ KB
```

In [73]:

```
df.info(memory_usage="deep")    # force pandas to calculate the true memory usage
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 193 entries, 0 to 192
Data columns (total 5 columns):
 #   Column                        Non-Null Count  Dtype
---  ------                        --------------  -----
 0   country                       193 non-null    object
 1   beer_servings                 193 non-null    int64
 2   wine_servings                 193 non-null    int64
 3   total_litres_of_pure_alcohol  193 non-null    float64
 4   continent                     193 non-null    object
dtypes: float64(1), int64(2), object(2)
memory usage: 29.0 KB
```

In [74]:

```
# calculate the memory usage for each Series (in bytes)
df.memory_usage(deep=True)
```

Out[74]:

```
Index                           128
country                       12588
beer_servings                  1544
wine_servings                  1544
total_litres_of_pure_alcohol   1544
continent                     12332
dtype: int64
```

In [75]:

```
df.continent
```

Out[75]:

```
0                Asia
1              Europe
2              Africa
3              Europe
4              Africa
            ...
188    South America
189              Asia
190              Asia
191            Africa
192            Africa
Name: continent, Length: 193, dtype: object
```

In [79]:

```
df.continent = df.continent.astype("category")
df.continent
```

Out[79]:

```
0                Asia
1              Europe
2              Africa
```

```
3              Europe
4              Africa
           ...
188    South America
189             Asia
190             Asia
191           Africa
192           Africa
Name: continent, Length: 193, dtype: category
Categories (6, object): ['Africa', 'Asia', 'Europe', 'North America', 'Oceania', 'South A
merica']
```

In [81]:

```
# strings are now encoded (0 means 'Africa', 1 means 'Asia', 2 means 'Europe', etc.)
df.continent.cat.codes.head(10)
```

Out[81]:

```
0    1
1    2
2    0
3    2
4    0
5    3
6    5
7    2
8    4
9    2
dtype: int8
```

In [83]:

```
df.memory_usage(deep=True)   # memory usage has been drastically reduced
```

Out[83]:

```
Index                            128
country                        12588
beer_servings                   1544
wine_servings                   1544
total_litres_of_pure_alcohol    1544
continent                        756
dtype: int64
```

In [84]:

```
# create a small DataFrame from a dictionary
df1 = pd.DataFrame({'ID':[100, 101, 102, 103], 'quality':['good', 'very good', 'good', '
excellent']})
df1
```

Out[84]:

	ID	quality
0	**100**	**good**
1	**101**	**very good**
2	**102**	**good**
3	**103**	**excellent**

In [85]:

```
df1.sort_values("quality")
```

Out[85]:

	ID	quality

	ID	quality
3	103	excellent
0	100	good
2	102	good
1	101	very good

In [107]:

```
# define a logical ordering for the categories
df1["quality"] = df1.quality.astype(pd.api.types.CategoricalDtype(categories=['good', 'very good', 'excellent'], ordered=True))
df1.quality
```

Out[107]:

```
0         good
1    very good
2         good
3    excellent
Name: quality, dtype: category
Categories (3, object): ['good' < 'very good' < 'excellent']
```

In [108]:

```
# sort the DataFrame by the 'quality' Series (logical order)
df1.sort_values("quality")
```

Out[108]:

	ID	quality
0	100	good
2	102	good
1	101	very good
3	103	excellent

In [109]:

```
# comparison operators work with ordered categories
df1.loc[df1.quality > "good" , :]
```

Out[109]:

	ID	quality
1	101	very good
3	103	excellent

In [111]:

```
df.sample(n=5)  # sample 5 rows from the DataFrame without replacement
```

Out[111]:

	country	beer_servings	wine_servings	total_litres_of_pure_alcohol	continent
186	Uzbekistan	25	8	2.4	Asia
60	Finland	263	97	10.0	Europe
54	El Salvador	52	2	2.2	North America
138	South Korea	140	9	9.8	Asia
101	Malawi	8	1	1.5	Africa

In [113]:

```
# use the 'random_state' parameter for reproducibility
df.sample(n=5, random_state=42)
```

Out[113]:

	country	beer_servings	wine_servings	total_litres_of_pure_alcohol	continent
45	Czech Republic	361	134	11.8	Europe
137	Qatar	1	7	0.9	Asia
76	Iceland	233	78	6.6	Europe
144	St. Lucia	171	71	10.1	North America
113	Montenegro	31	128	4.9	Europe

17. How do I create dummy variables in pandas?

In [117]:

```
pd.get_dummies(df.continent).iloc[: , 1:].head()  # alternative: use 'get_dummies' to cr
eate one column for every possible value
```

Out[117]:

	Asia	Europe	North America	Oceania	South America
0	1	0	0	0	0
1	0	1	0	0	0
2	0	0	0	0	0
3	0	1	0	0	0
4	0	0	0	0	0

18. How do I work with dates and times in pandas?

In [118]:

```
# read a dataset of UFO reports into a DataFrame
df = pd.read_csv('http://bit.ly/uforeports')
df.head()
```

Out[118]:

	City	Colors Reported	Shape Reported	State	Time
0	Ithaca	NaN	TRIANGLE	NY	6/1/1930 22:00
1	Willingboro	NaN	OTHER	NJ	6/30/1930 20:00
2	Holyoke	NaN	OVAL	CO	2/15/1931 14:00
3	Abilene	NaN	DISK	KS	6/1/1931 13:00
4	New York Worlds Fair	NaN	LIGHT	NY	4/18/1933 19:00

In [119]:

```
df.dtypes
```

Out[119]:

```
City               object
Colors Reported    object
Shape Reported     object
```

```
Shape Reported      object
State               object
Time                object
dtype: object
```

In [120]:

```
# convert 'Time' to datetime format
df['Time'] = pd.to_datetime(df.Time)
df.head()
```

Out[120]:

	City	Colors Reported	Shape Reported	State	Time
0	**Ithaca**	**NaN**	**TRIANGLE**	**NY**	**1930-06-01 22:00:00**
1	**Willingboro**	**NaN**	**OTHER**	**NJ**	**1930-06-30 20:00:00**
2	**Holyoke**	**NaN**	**OVAL**	**CO**	**1931-02-15 14:00:00**
3	**Abilene**	**NaN**	**DISK**	**KS**	**1931-06-01 13:00:00**
4	**New York Worlds Fair**	**NaN**	**LIGHT**	**NY**	**1933-04-18 19:00:00**

In [121]:

```
df.dtypes
```

Out[121]:

```
City                        object
Colors Reported             object
Shape Reported              object
State                       object
Time                datetime64[ns]
dtype: object
```

In [123]:

```
df.Time.dt.hour.head()    # convenient Series attributes are now available
```

Out[123]:

```
0    22
1    20
2    14
3    13
4    19
Name: Time, dtype: int64
```

In [125]:

```
df.Time.dt.weekday.head()
```

Out[125]:

```
0    6
1    0
2    6
3    0
4    1
Name: Time, dtype: int64
```

In [132]:

```
df.Time.dt.day_name().head()
```

Out[132]:

```
0      Sunday
1      Monday
2      Sunday
3      Monday
```

```
4    Tuesday
Name: Time, dtype: object
```

In [133]:

```
df.Time.dt.dayofyear.head()
```

Out[133]:

```
0    152
1    181
2     46
3    152
4    108
Name: Time, dtype: int64
```

19. How do I find and remove duplicate rows in pandas?

In [141]:

```
df.duplicated().sum()  # count the duplicate rows
```

Out[141]:

```
109
```

In [142]:

```
df.duplicated().value_counts()
```

Out[142]:

```
False    18132
True       109
dtype: int64
```

In [143]:

```
df.City.duplicated().sum()  # count the duplicate items (True becomes 1, False becomes 0)
```

Out[143]:

```
11764
```

In [147]:

```
df.loc[df.duplicated(keep="first") , :]
```

Out[147]:

	City	Colors Reported	Shape Reported	State	Time
195	Miami	NaN	DISK	FL	1952-06-30 21:00:00
469	Madison	NaN	CIGAR	WI	1957-12-28 00:00:00
473	Winooski	NaN	OVAL	VT	1958-04-17 21:30:00
869	Covina	NaN	CIGAR	CA	1964-05-15 15:00:00
943	Mt. Prospect	NaN	DISK	IL	1964-09-25 19:00:00
...	...	...	...	...	...
17665	Mohawk Valley	NaN	OTHER	AZ	2000-10-13 01:00:00
17843	Cygnet	NaN	EGG	OH	2000-10-31 19:15:00
17871	Dover	NaN	TRIANGLE	DE	2000-11-03 19:40:00
18195	Walpole	GREEN	FIREBALL	NH	2000-12-26 18:20:00
18231	Pismo Beach	NaN	OVAL	CA	2000-12-31 20:00:00

109 rows × 5 columns

In [148]:

```
# drop the duplicate rows (inplace=False by default)
df.drop_duplicates(keep='first').shape
```

Out[148]:

```
(18132, 5)
```

In [149]:

```
df.drop_duplicates(keep='last').shape
```

Out[149]:

```
(18132, 5)
```

some tricks in pandas

1. Create a datetime column from a DataFrame

In [2]:

```
df = pd.DataFrame([[12 , 25 , 2017 , 10] , [1 , 15 , 2018 , 11]] , columns=["month" , "day" , "year" , "hour"])
df
```

Out[2]:

	month	day	year	hour
0	12	25	2017	10
1	1	15	2018	11

In [3]:

```
# new: create a datetime column from the entire DataFrame
pd.to_datetime(df)
```

Out[3]:

```
0   2017-12-25 10:00:00
1   2018-01-15 11:00:00
dtype: datetime64[ns]
```

In [13]:

```
df.index = pd.to_datetime(df[["month" , "day" , "year"]])  # overwrite the index
df
```

Out[13]:

	month	day	year	hour
2017-12-25	12	25	2017	10
2018-01-15	1	15	2018	11

2. Create a category column during file reading

In [23]:

```
df = pd.DataFrame({"name" : ["one" , "two"] , "age" : [21 , 39] } , dtype="category"  )
df
```

Out[23]:

	name	age
0	one	21
1	two	39

In [19]:

```
df.dtypes
```

Out[19]:

```
name    category
age     category
dtype: object
```

3. Convert the data type of multiple columns at once

In [24]:

```
# read the drinks dataset into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks.dtypes
```

Out[24]:

```
country                          object
beer_servings                     int64
spirit_servings                   int64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

In [26]:

```
drinks["beer_servings"] = drinks.beer_servings.astype("float")
drinks["spirit_servings"] = drinks.spirit_servings.astype("float")
drinks.dtypes
```

Out[26]:

```
country                          object
beer_servings                   float64
spirit_servings                 float64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

In [27]:

```
# new way to convert data types (all at once)
drinks = drinks.astype({'beer_servings':'float', 'spirit_servings':'float'})
drinks.dtypes
```

Out[27]:

```
country                          object
beer_servings                   float64
spirit_servings                 float64
wine_servings                     int64
total_litres_of_pure_alcohol    float64
continent                        object
dtype: object
```

In [28]:

```
# new: apply the same aggregations to a DataFrame
drinks.agg(['mean', 'min', 'max'])
```

Out[28]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
min	Afghanistan	0.000000	0.000000	0.000000	0.000000	Africa
max	Zimbabwe	376.000000	438.000000	370.000000	14.400000	South America
mean	NaN	106.160622	80.994819	49.450777	4.717098	NaN

4. Rename columns

In [31]:

```
df.rename({"name" : "Names" , "ange" :"Age"} , axis="columns")
```

Out[31]:

	names	age
0	one	21
1	two	39

In [33]:

```
df.columns = ['NAMES', 'AGE']
df
```

Out[33]:

	NAMES	AGE
0	one	21
1	two	39

In [37]:

```
# Finally, if you just need to add a prefix or suffix to all of your column names, you ca
n use the add_prefix() method.
df.add_prefix('x_')
```

Out[37]:

	x_NAMES	x_AGE
0	one	21
1	two	39

In [40]:

```
df.add_suffix('_Y')
```

Out[40]:

	NAMES_Y	AGE_Y
0	one	21
1	two	39

In [44]:

```
# Let's say you need to select only the numeric columns. You can use the select_dtypes()
```

```
method:
drinks.select_dtypes(include='number').head()
```

Out[44]:

	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol
0	0.0	0.0	0	0.0
1	89.0	132.0	54	4.9
2	25.0	0.0	14	0.7
3	245.0	138.0	312	12.4
4	217.0	57.0	45	5.9

In [45]:

```
drinks.select_dtypes(include='float').head()
```

Out[45]:

	beer_servings	spirit_servings	total_litres_of_pure_alcohol
0	0.0	0.0	0.0
1	89.0	132.0	4.9
2	25.0	0.0	0.7
3	245.0	138.0	12.4
4	217.0	57.0	5.9

In [46]:

```
drinks.select_dtypes(include='object').head()
```

Out[46]:

	country	continent
0	Afghanistan	Asia
1	Albania	Europe
2	Algeria	Africa
3	Andorra	Europe
4	Angola	Africa

In [52]:

```
# You can also tell it to exclude certain data types
drinks.select_dtypes(exclude='number').head()
```

Out[52]:

	country	continent
0	Afghanistan	Asia
1	Albania	Europe
2	Algeria	Africa
3	Andorra	Europe
4	Angola	Africa

5. Convert strings to numbers

In [53]:

```
df = pd.DataFrame({'col_one':['1.1', '2.2', '3.3'],
                   'col_two':['4.4', '5.5', '6.6'],
                   'col_three':['7.7', '8.8', '-']})
df
```

Out[53]:

	col_one	col_two	col_three
0	**1.1**	**4.4**	**7.7**
1	**2.2**	**5.5**	**8.8**
2	**3.3**	**6.6**	**-**

In [54]:

```
df.dtypes
```

Out[54]:

```
col_one      object
col_two      object
col_three    object
dtype: object
```

In [55]:

```
df.astype({'col_one':'float', 'col_two':'float'}).dtypes
```

Out[55]:

```
col_one      float64
col_two      float64
col_three     object
dtype: object
```

In [56]:

```
# Instead, you can use the to_numeric() function on the third column and tell it to conve
rt any invalid input into NaN values:
pd.to_numeric(df.col_three, errors='coerce')
```

Out[56]:

```
0    7.7
1    8.8
2    NaN
Name: col_three, dtype: float64
```

In [57]:

```
# If you know that the NaN values actually represent zeros, you can fill them with zeros
using the fillna() method:
pd.to_numeric(df.col_three, errors='coerce').fillna(0)
```

Out[57]:

```
0    7.7
1    8.8
2    0.0
Name: col_three, dtype: float64
```

In [58]:

```
# Finally, you can apply this function to the entire DataFrame all at once by using the a
pply() method:
df = df.apply(pd.to_numeric, errors='coerce').fillna(0)
df
```

Out[58]:

	col_one	col_two	col_three
0	1.1	4.4	7.7
1	2.2	5.5	8.8
2	3.3	6.6	0.0

In [59]:

```
df.dtypes
```

Out[59]:

```
col_one       float64
col_two       float64
col_three     float64
dtype: object
```

6. Split a string into multiple columns

In [60]:

```
df = pd.DataFrame({'name':['John Arthur Doe', 'Jane Ann Smith'],
                   'location':['Los Angeles, CA', 'Washington, DC']})
df
```

Out[60]:

	name	location
0	John Arthur Doe	Los Angeles, CA
1	Jane Ann Smith	Washington, DC

In [61]:

```
df.name.str.split(' ', expand=True)
```

Out[61]:

	0	1	2
0	John	Arthur	Doe
1	Jane	Ann	Smith

In [62]:

```
df[['first', 'middle', 'last']] = df.name.str.split(' ', expand=True)
df
```

Out[62]:

	name	location	first	middle	last
0	John Arthur Doe	Los Angeles, CA	John	Arthur	Doe
1	Jane Ann Smith	Washington, DC	Jane	Ann	Smith

In [63]:

```
df['city'] = df.location.str.split(', ', expand=True)[0]
df
```

Out[63]:

	name	location	first	middle	last	city

	name	location	first	middle	last	city
0	John Arthur Doe	Los Angeles, CA	John	Arthur	Doe	Los Angeles
1	Jane Ann Smith	Washington, DC	Jane	Ann	Smith	Washington

7. Expand a Series of lists into a DataFrame

In [64]:

```
df = pd.DataFrame({'col_one':['a', 'b', 'c'], 'col_two':[[10, 40], [20, 50], [30, 60]]})
df
```

Out[64]:

	col_one	col_two
0	a	[10, 40]
1	b	[20, 50]
2	c	[30, 60]

In [65]:

```
# If we wanted to expand the second column into its own DataFrame, we can use the apply() method on that column and pass it the Series constructor:
df_new = df.col_two.apply(pd.Series)
df_new
```

Out[65]:

	0	1
0	10	40
1	20	50
2	30	60

In [66]:

```
# And by using the concat() function, you can combine the original DataFrame with the new DataFrame:
pd.concat([df, df_new], axis='columns')
```

Out[66]:

	col_one	col_two	0	1
0	a	[10, 40]	10	40
1	b	[20, 50]	20	50
2	c	[30, 60]	30	60

Bonus: Profile a DataFrame

In [4]:

```
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
drinks
```

Out[4]:

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
0	Afghanistan	0	0	0	0.0	Asia

	country	beer_servings	spirit_servings	wine_servings	total_litres_of_pure_alcohol	continent
1	Albania	89	132	54	4.9	Europe
2	Algeria	25	0	14	0.7	Africa
3	Andorra	245	138	312	12.4	Europe
4	Angola	217	57	45	5.9	Africa
...	...	...	...	...	...	...
188	Venezuela	333	100	3	7.7	South America
189	Vietnam	111	2	1	2.0	Asia
190	Yemen	6	0	0	0.1	Asia
191	Zambia	32	19	4	2.5	Africa
192	Zimbabwe	64	18	4	4.7	Africa

193 rows × 6 columns

In [2]:

```
import pandas_profiling
```

In [6]:

```
over_view = pandas_profiling.ProfileReport(drinks)
```

In [9]:

```
over_view
```

Out[9]:

In [ ]: