Chapter 2 Subsetting and Sorting

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

# Read titanic dataset
tnc = pd.read_csv("./datasets/titanic.csv")

# Print dataframe
tnc.head()
```

Out[1]:

	pclass	survived	name	gender	age	sibsp	parch	ticket	fare	cabin	em
0	1	1	Allen, Miss. Elisabeth Walton	female	29	0	0	24160	211.3375	B5	
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.55	C22 C26	
2	1	0	Allison, Miss. Helen Loraine	female	2	1	2	113781	151.55	C22 C26	
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30	1	2	113781	151.55	C22 C26	
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25	1	2	113781	151.55	C22 C26	

In [2]:
```
# Info about columns, datatypes, non-null columns, and total size
tnc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   pclass     1309 non-null   int64
 1   survived   1309 non-null   int64
 2   name       1309 non-null   object
 3   gender     1309 non-null   object
 4   age        1309 non-null   object
 5   sibsp      1309 non-null   int64
 6   parch      1309 non-null   int64
 7   ticket     1309 non-null   object
 8   fare       1309 non-null   object
 9   cabin      1309 non-null   object
 10  embarked   1309 non-null   object
 11  boat       1309 non-null   object
 12  body       1309 non-null   object
 13  home.dest  1309 non-null   object
dtypes: int64(4), object(10)
memory usage: 143.3+ KB
```

In [3]:
```
# Shape of the dataframe
tnc.shape
```

Out[3]:
```
(1309, 14)
```

Selecting column

We can select one column from a Dataframe using the following two syntax:

1. **Dataframe.column_name**
2. **Dataframe["column_name"]** or **Dataframe["column_name"]**

In [4]:
```
# Extracting age column from the data
tnc.age
```

Out[4]:
```
0          29
1      0.9167
2           2
3          30
4          25
         ...
1304     14.5
1305        ?
1306     26.5
1307       27
1308       29
Name: age, Length: 1309, dtype: object
```

In [5]:
```
# Extracting name column from the dataframe
tnc["name"]
```

Out[5]:
```
0                          Allen, Miss. Elisabeth Walton
1                         Allison, Master. Hudson Trevor
2                           Allison, Miss. Helen Loraine
3                   Allison, Mr. Hudson Joshua Creighton
4         Allison, Mrs. Hudson J C (Bessie Waldo Daniels)
                               ...
1304                             Zabour, Miss. Hileni
1305                            Zabour, Miss. Thamine
1306                         Zakarian, Mr. Mapriededer
1307                               Zakarian, Mr. Ortin
1308                                Zimmerman, Mr. Leo
Name: name, Length: 1309, dtype: object
```

Why second syntax is better than first?

First syntax: **Dataframe.column_name**

Second syntax: **Dataframe["column_name"]**

Because there can be many instances where the dot operator syntax may not work.

One such example is that we cannot access the column "home.dest" using the dot operator.

In [6]:
```
# Using first syntax
# tnc.home.dest

# Python looks for 'dest' attribute in 'home' attribute of 'tnc'.
# It first looks for 'home' which is not an attribute and hence results in error
```

In [7]:
```
# Extracting "home.dest" column using the second syntax
tnc["home.dest"]
```

Out[7]:
```
0                          St Louis, MO
1       Montreal, PQ / Chesterville, ON
2       Montreal, PQ / Chesterville, ON
3       Montreal, PQ / Chesterville, ON
4       Montreal, PQ / Chesterville, ON
                       ...
1304                                 ?
1305                                 ?
1306                                 ?
1307                                 ?
1308                                 ?
Name: home.dest, Length: 1309, dtype: object
```

Selecting multiple columns

We can select multiple columns using the square bracket syntax:

Dataframe[["col1_name", "col2_name", ..., "colN_name"]]

In [8]:
```
# Describe the statistical information about the columns name, age and home.dest of
```

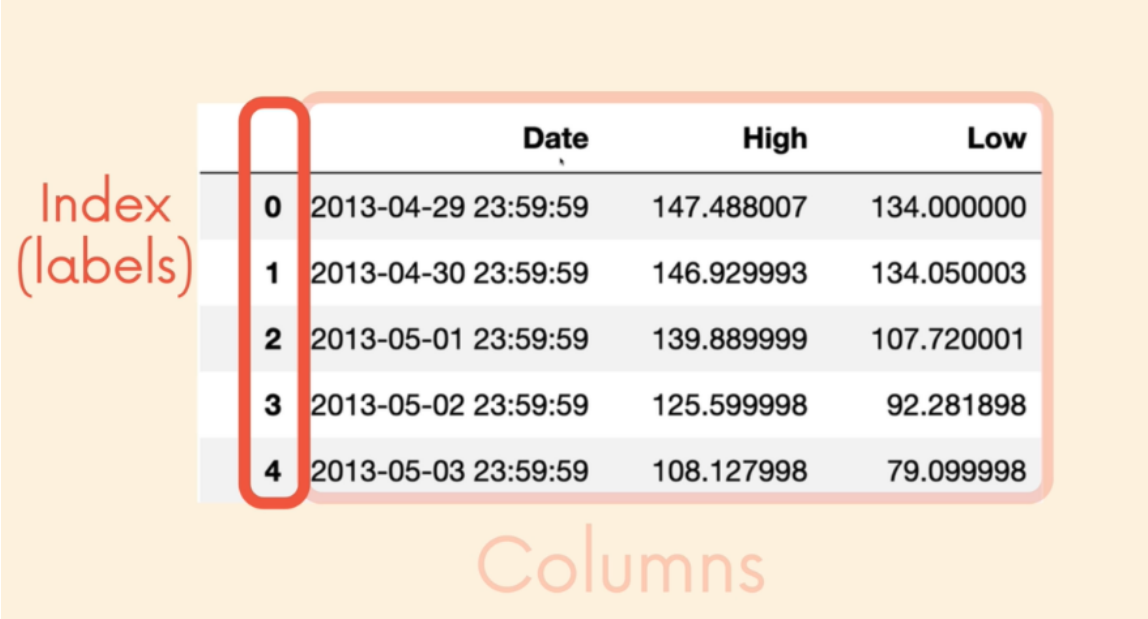
```
# Extract first 25 members
tnc_25 = tnc.head(25)

# Select the columns name, age and home.dest using square bracket syntax
tnc_cols = tnc_25[["name", "age", "home.dest"]]

# Describe the columns
tnc_cols.describe()

# This can also be achieved in a single line using Method Chaining
# tnc.head(25)[["name", "age", "home.dest"]].describe()
```

Out[8]:

	name	age	home.dest
count	25	25	25
unique	25	21	12
top	Allen, Miss. Elisabeth Walton	29	New York, NY
freq	1	2	7

Index (or) Label

Index or label are unique labels given to records in a Dataframe. It is given by Pandas by default to uniquely indentify the records.



To know the current index of the Dataframe we can use the **index property upon the Dataframe**.

Ex: df.index

In [9]:
```
# Print titanic dataframe
```

```
tnc
```

Out[9]:

	pclass	survived	name	gender	age	sibsp	parch	ticket	fare	cabi
0	1	1	Allen, Miss. Elisabeth Walton	female	29	0	0	24160	211.3375	B
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.55	C2 C2
2	1	0	Allison, Miss. Helen Loraine	female	2	1	2	113781	151.55	C2 C2
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30	1	2	113781	151.55	C2 C2
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25	1	2	113781	151.55	C2 C2
...	...	...	...	...	...	...	...	...	...	
1304	3	0	Zabour, Miss. Hileni	female	14.5	1	0	2665	14.4542	
1305	3	0	Zabour, Miss. Thamine	female	?	1	0	2665	14.4542	
1306	3	0	Zakarian, Mr. Mapriededer	male	26.5	0	0	2656	7.225	
1307	3	0	Zakarian, Mr. Ortin	male	27	0	0	2670	7.225	
1308	3	0	Zimmerman, Mr. Leo	male	29	0	0	315082	7.875	

1309 rows × 14 columns

In [10]:
```
# Print index of titanic
tnc.index
```

Out[10]:
```
RangeIndex(start=0, stop=1309, step=1)
```

Set index

We can create our own index using the method:

Dataframe.set_index(col_name)

Ex: df.set_index("name")

Note that when a column is made as an index, it is no longer a column of the dataframe and rather acts as an index to the dataframe

Also note that this is not an in place operation and hence returns a copy of the resultant Dataframe.

In [11]:
```
# Setting an index
tnc_new = tnc.set_index("name")
```

In [12]:
```
# Print new titanic dataframe
tnc_new
```

Out[12]:

name	pclass	survived	gender	age	sibsp	parch	ticket	fare	cabin	em
Allen, Miss. Elisabeth Walton	1	1	female	29	0	0	24160	211.3375	B5	
Allison, Master. Hudson Trevor	1	1	male	0.9167	1	2	113781	151.55	C22 C26	
Allison, Miss. Helen Loraine	1	0	female	2	1	2	113781	151.55	C22 C26	
Allison, Mr. Hudson Joshua Creighton	1	0	male	30	1	2	113781	151.55	C22 C26	
Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	1	0	female	25	1	2	113781	151.55	C22 C26	
...	...	...	...	...	...	...	...	...	...	
Zabour, Miss. Hileni	3	0	female	14.5	1	0	2665	14.4542	?	
Zabour, Miss. Thamine	3	0	female	?	1	0	2665	14.4542	?	
Zakarian, Mr. Mapriededer	3	0	male	26.5	0	0	2656	7.225	?	
Zakarian, Mr. Ortin	3	0	male	27	0	0	2670	7.225	?	
Zimmerman, Mr. Leo	3	0	male	29	0	0	315082	7.875	?	

1309 rows × 13 columns

In [13]:
```
# Print index of new titanic
tnc_new.index
```

Out[13]:
```
Index(['Allen, Miss. Elisabeth Walton', 'Allison, Master. Hudson Trevor',
       'Allison, Miss. Helen Loraine', 'Allison, Mr. Hudson Joshua Creighton',
       'Allison, Mrs. Hudson J C (Bessie Waldo Daniels)',
       'Anderson, Mr. Harry', 'Andrews, Miss. Kornelia Theodosia',
       'Andrews, Mr. Thomas Jr',
       'Appleton, Mrs. Edward Dale (Charlotte Lamson)',
       'Artagaveytia, Mr. Ramon',
       ...
       'Yasbeck, Mr. Antoni', 'Yasbeck, Mrs. Antoni (Selini Alexander)',
       'Youseff, Mr. Gerious', 'Yousif, Mr. Wazli', 'Yousseff, Mr. Gerious',
       'Zabour, Miss. Hileni', 'Zabour, Miss. Thamine',
       'Zakarian, Mr. Mapriededer', 'Zakarian, Mr. Ortin',
       'Zimmerman, Mr. Leo'],
      dtype='object', name='name', length=1309)
```

To set an index in place

To set an index in place we need to **set the argument 'inplace' of set_index() method to True.**

Ex: df.set_index(col, inplace=True)

In [14]:
```
# Read a new dataset 'world happiness report 2021'
countries = pd.read_csv("./datasets/world-happiness-report-2021.csv")

# Print the dataframe
countries.head()
```

Out[14]:

	Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Socia suppor
0	Finland	Western Europe	7.842	0.032	7.904	7.780	10.775	0.954
1	Denmark	Western Europe	7.620	0.035	7.687	7.552	10.933	0.954
2	Switzerland	Western Europe	7.571	0.036	7.643	7.500	11.117	0.942
3	Iceland	Western Europe	7.554	0.059	7.670	7.438	10.878	0.983
4	Netherlands	Western Europe	7.464	0.027	7.518	7.410	10.932	0.942

In [15]:
```
# Select 'Healthy life expectancy' column of dataframe
countries["Healthy life expectancy"]
```

```
# It prints 'Healthy life expectancy', but I cannot know corresponding to which cou
# Setting an index plays a key role here.
```

Out[15]:
```
0      72.000
1      72.700
2      74.400
3      73.000
4      72.400
        ...
144    48.700
145    59.269
146    61.400
147    56.201
148    52.493
Name: Healthy life expectancy, Length: 149, dtype: float64
```

In [16]:
```
# Set Country name column as index in place
countries.set_index("Country name", inplace=True)

# Print the dataframe
countries
```

Out[16]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Social support
Finland	Western Europe	7.842	0.032	7.904	7.780	10.775	0.954
Denmark	Western Europe	7.620	0.035	7.687	7.552	10.933	0.954
Switzerland	Western Europe	7.571	0.036	7.643	7.500	11.117	0.942
Iceland	Western Europe	7.554	0.059	7.670	7.438	10.878	0.983
Netherlands	Western Europe	7.464	0.027	7.518	7.410	10.932	0.942
...	...	...	...	...	...	...	...
Lesotho	Sub-Saharan Africa	3.512	0.120	3.748	3.276	7.926	0.787
Botswana	Sub-Saharan Africa	3.467	0.074	3.611	3.322	9.782	0.784
Rwanda	Sub-Saharan Africa	3.415	0.068	3.548	3.282	7.676	0.552
Zimbabwe	Sub-Saharan Africa	3.145	0.058	3.259	3.030	7.943	0.750
Afghanistan	South Asia	2.523	0.038	2.596	2.449	7.695	0.463

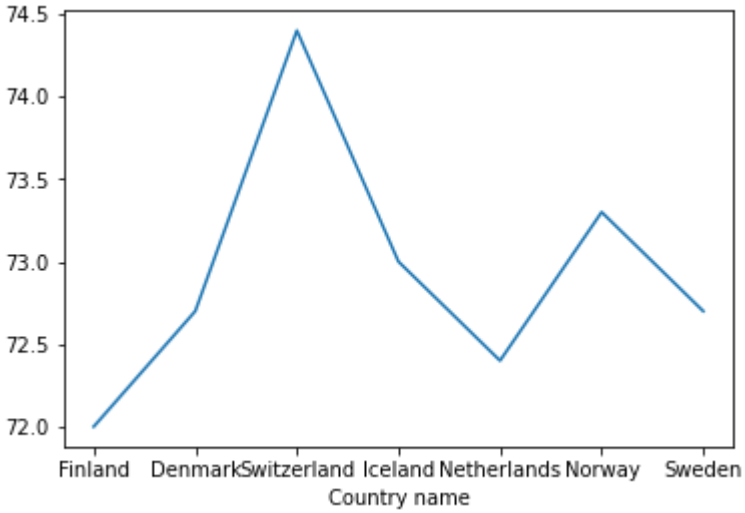
149 rows × 19 columns

In [17]:
```
# Select 'Healthy life expectancy' column of dataframe
countries["Healthy life expectancy"]
```

Out[17]:
```
Country name
Finland        72.000
Denmark        72.700
Switzerland    74.400
Iceland        73.000
Netherlands    72.400
                ...
Lesotho        48.700
Botswana       59.269
Rwanda         61.400
Zimbabwe       56.201
Afghanistan    52.493
Name: Healthy life expectancy, Length: 149, dtype: float64
```

In [18]:
```
# Plot the 'Healthy life expectancy' of first seven countries
countries.head(7)["Healthy life expectancy"].plot()
```

Out[18]:
```
<AxesSubplot:xlabel='Country name'>
```



Sorting

Sort the dataframe records in the ascending or descending order of the specified column. To sort the records in ascending order we use the method:

Dataframe.sort_values(col_name)

Ex: df.sort_values("name")

To sort the records in descending order we need to set the argument 'ascending' of sort_values() method to False.

Ex: df.sort_values("name", ascending=False)

Note that this is not an in place operation and hence returns a copy of the resultant Dataframe.

To sort the records in ascending/descending order in place we need to set the argument 'inplace' of set_index() method to True.

Ex: df.sort_values("name", ascending=False, inplace=True)

In [19]:
```
# Sort the countries dataframe in ascending order of Social support
countries.sort_values("Social support")
```

Out[19]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	su
Afghanistan	South Asia	2.523	0.038	2.596	2.449	7.695	
Benin	Sub-Saharan Africa	5.045	0.073	5.189	4.901	8.087	
Burundi	Sub-Saharan Africa	3.775	0.107	3.985	3.565	6.635	
Malawi	Sub-Saharan Africa	3.600	0.092	3.781	3.419	6.958	
Haiti	Latin America and Caribbean	3.615	0.173	3.953	3.276	7.477	
...	...	...	...	...	...	...	
Norway	Western Europe	7.392	0.035	7.462	7.323	11.053	
Denmark	Western Europe	7.620	0.035	7.687	7.552	10.933	
Finland	Western Europe	7.842	0.032	7.904	7.780	10.775	
Turkmenistan	Commonwealth of Independent States	5.066	0.036	5.136	4.996	9.629	
Iceland	Western Europe	7.554	0.059	7.670	7.438	10.878	

149 rows × 19 columns

In [20]:
```
# Sort the countries dataframe in descending order of Freedom to make life choices
countries.sort_values("Freedom to make life choices", ascending=False)
```

Out[20]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Sc supp
Uzbekistan	Commonwealth of Independent States	6.179	0.068	6.312	6.045	8.836	0
Norway	Western Europe	7.392	0.035	7.462	7.323	11.053	0
Cambodia	Southeast Asia	4.830	0.067	4.963	4.698	8.360	0
Iceland	Western Europe	7.554	0.059	7.670	7.438	10.878	0
Finland	Western Europe	7.842	0.032	7.904	7.780	10.775	0
...	...	...	...	...	...	...	
Madagascar	Sub-Saharan Africa	4.208	0.072	4.349	4.068	7.396	0
Comoros	Sub-Saharan Africa	4.289	0.084	4.454	4.123	8.031	0
Lebanon	Middle East and North Africa	4.584	0.055	4.691	4.477	9.626	0
Algeria	Middle East and North Africa	4.887	0.053	4.991	4.783	9.342	0
Afghanistan	South Asia	2.523	0.038	2.596	2.449	7.695	0

149 rows × 19 columns

In [21]:
```
# Sort the countries dataframe in descending order of Generosity in place
countries.sort_values("Generosity", ascending=False, inplace=True)

# Print the dataframe
countries
```

Out[21]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Soc supp
Indonesia	Southeast Asia	5.345	0.056	5.454	5.235	9.365	0.8
Myanmar	Southeast Asia	4.426	0.052	4.527	4.324	8.541	0.7
Gambia	Sub-Saharan Africa	5.051	0.089	5.225	4.877	7.686	0.6
Haiti	Latin America and Caribbean	3.615	0.173	3.953	3.276	7.477	0.5
Uzbekistan	Commonwealth of Independent States	6.179	0.068	6.312	6.045	8.836	0.9
...	...	...	...	...	...	...	
Georgia	Commonwealth of Independent States	4.891	0.054	4.998	4.785	9.585	0.6
Portugal	Western Europe	5.929	0.055	6.037	5.821	10.421	0.8
Botswana	Sub-Saharan Africa	3.467	0.074	3.611	3.322	9.782	0.7
Japan	East Asia	5.940	0.040	6.020	5.861	10.611	0.8
Greece	Western Europe	5.723	0.046	5.813	5.632	10.279	0.8

149 rows × 19 columns

Sorting by index

Since index is not a column of a dataframe we cannot directly use sort_values() method on the index. Instead we use the method:

Dataframe.sort_index()

Ex: df.sort_index()

Note: As usual by default the values of the arguments 'inplace' and 'ascending' are False

In [22]:
```
# Sort the countries by its index
countries.sort_index()
```

Out[22]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Sc supp
Afghanistan	South Asia	2.523	0.038	2.596	2.449	7.695	0
Albania	Central and Eastern Europe	5.117	0.059	5.234	5.001	9.520	0
Algeria	Middle East and North Africa	4.887	0.053	4.991	4.783	9.342	0
Argentina	Latin America and Caribbean	5.929	0.056	6.040	5.819	9.962	0
Armenia	Commonwealth of Independent States	5.283	0.058	5.397	5.168	9.487	0
...	...	...	...	...	...	...	
Venezuela	Latin America and Caribbean	4.892	0.064	5.017	4.767	9.073	0
Vietnam	Southeast Asia	5.411	0.039	5.488	5.334	8.973	0
Yemen	Middle East and North Africa	3.658	0.070	3.794	3.521	7.578	0
Zambia	Sub-Saharan Africa	4.073	0.069	4.209	3.938	8.145	0
Zimbabwe	Sub-Saharan Africa	3.145	0.058	3.259	3.030	7.943	0

149 rows × 19 columns

Sorting by multiple columns

We can sort the dataframe by multiple columns using the same syntax. Instead of passing one column, we pass a list of columns.

Syntax: Dataframe.sort_values([col1_name, col2_name, ..., colN_name])

Ex: df.sort_values(["name", "age"])

In [23]:
```
# Read houses dataset
houses = pd.read_csv("./datasets/kc_house_data.csv")

# Print the dataframe
houses.head()
```

Out[23]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	fl
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	

5 rows × 21 columns

In [24]:
```
# Set in place id as index of dataframe
houses.set_index("id", inplace=True)

# Print the dataframe
houses
```

Out[24]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floor
7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.
6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.
5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.
2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.
1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.
...	...	...	...	...	...	...	.
263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.
6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.
1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.
291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.
1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.

21613 rows × 20 columns

In [25]:
```
# Sort the dataframe in descending order of price
houses.sort_values("price", ascending=False)
```

Out[25]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floc
6762700020	20141013T000000	7700000.0	6	8.00	12050	27600	2
9808700762	20140611T000000	7062500.0	5	4.50	10040	37325	2
9208900037	20140919T000000	6885000.0	6	7.75	9890	31374	2
2470100110	20140804T000000	5570000.0	5	5.75	9200	35069	2
8907500070	20150413T000000	5350000.0	5	5.00	8000	23985	2
...	...	...	...	...	...	...	
3883800011	20141105T000000	82000.0	3	1.00	860	10426	1
3028200080	20150324T000000	81000.0	2	1.00	730	9975	1
8658300340	20140523T000000	80000.0	1	0.75	430	5050	1
40000362	20140506T000000	78000.0	2	1.00	780	16344	1
3421079032	20150217T000000	75000.0	1	0.00	670	43377	1

21613 rows × 20 columns

In [26]:
```
# Sort the dataframe in descending order of bedrooms
houses.sort_values("bedrooms", ascending=False)
```

Out[26]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floc
2402100895	20140625T000000	640000.0	33	1.75	1620	6000	1
1773100755	20140821T000000	520000.0	11	3.00	3000	4960	2
5566100170	20141029T000000	650000.0	10	2.00	3610	11914	2
627300145	20140814T000000	1148000.0	10	5.25	4590	10920	1
8812401450	20141229T000000	660000.0	10	3.00	2920	3745	2
...	...	...	...	...	...	...	
3980300371	20140926T000000	142000.0	0	0.00	290	20875	1
2310060040	20140925T000000	240000.0	0	2.50	1810	5669	2
6306400140	20140612T000000	1095000.0	0	0.00	3064	4764	3
2569500210	20141117T000000	339950.0	0	2.50	2290	8319	2
3374500520	20150429T000000	355000.0	0	0.00	2460	8049	2

21613 rows × 20 columns

In [27]:
```
# Sort the dataframe in descending order of bedrooms and if there is a tie sort the
```

```
houses.sort_values(["bedrooms", "price"], ascending=False)
```

Out[27]:

id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floc
2402100895	20140625T000000	640000.0	33	1.75	1620	6000	1
1773100755	20140821T000000	520000.0	11	3.00	3000	4960	2
627300145	20140814T000000	1148000.0	10	5.25	4590	10920	1
8812401450	20141229T000000	660000.0	10	3.00	2920	3745	2
5566100170	20141029T000000	650000.0	10	2.00	3610	11914	2
...	...	...	...	...	...	...	
2310060040	20140925T000000	240000.0	0	2.50	1810	5669	2
7849202190	20141223T000000	235000.0	0	0.00	1470	4800	2
6896300380	20141002T000000	228000.0	0	1.00	390	5900	1
3980300371	20140926T000000	142000.0	0	0.00	290	20875	1
9543000205	20150413T000000	139950.0	0	0.00	844	4269	1

21613 rows × 20 columns

Selecting rows

We can select rows from a Dataframe based on their label/index using the following methods:

1. **Dataframe.loc["label"]**: Access a group of records based on their label.

Ex: df.loc["group1"], returns all the records with label "group1".

2. **Dataframe.iloc[position]**: Access a group of records based on their position

Ex: df.iloc[0], returns first row (index 0).

Note: loc looks for values whereas iloc looks for position

We can also make use of Python slicing to extract records within a range

3. **Dataframe.loc[start:end] (or) Dataframe.iloc[start:end]**

Ex: df.loc["group1":"group3"], returns all the records of dataframe starting with label "group1" upto label "group3".

Ex: df.loc[0:9], returns all the records of dataframe whose position are within the range 0 to 9.

In [28]:
```
# Extract details about country 'India'
countries.loc["India"]
```

Out[28]:
```
Regional indicator                          South Asia
Ladder score                                     3.819
Standard error of ladder score                   0.026
upperwhisker                                     3.869
lowerwhisker                                     3.769
Logged GDP per capita                            8.755
Social support                                   0.603
Healthy life expectancy                         60.633
Freedom to make life choices                     0.893
Generosity                                       0.089
Perceptions of corruption                        0.774
Ladder score in Dystopia                          2.43
Explained by: Log GDP per capita                 0.741
Explained by: Social support                     0.316
Explained by: Healthy life expectancy            0.383
Explained by: Freedom to make life choices       0.622
Explained by: Generosity                         0.246
Explained by: Perceptions of corruption          0.106
Dystopia + residual                              1.405
Name: India, dtype: object
```

In [29]:
```
# Extract details about country 'India' horizontally
countries.loc[["India"]]
```

Out[29]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Social support	exp
India	South Asia	3.819	0.026	3.869	3.769	8.755	0.603	

In [30]:
```
# Extract details about any 5 Asian countries
countries.loc[["India", "Pakistan", "Bangladesh", "Sri Lanka", "Nepal"]]
```

Out[30]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Social support
India	South Asia	3.819	0.026	3.869	3.769	8.755	0.603
Pakistan	South Asia	4.934	0.068	5.066	4.802	8.458	0.651
Bangladesh	South Asia	5.025	0.046	5.115	4.934	8.454	0.693
Sri Lanka	South Asia	4.325	0.066	4.454	4.196	9.470	0.827
Nepal	South Asia	5.269	0.070	5.406	5.132	8.120	0.774

In [31]:
```
# Extract details within a range
tnc.loc[0:5]
```

Out[31]:

	pclass	survived	name	gender	age	sibsp	parch	ticket	fare	cabin	en
0	1	1	Allen, Miss. Elisabeth Walton	female	29	0	0	24160	211.3375	B5	
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.55	C22 C26	
2	1	0	Allison, Miss. Helen Loraine	female	2	1	2	113781	151.55	C22 C26	
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30	1	2	113781	151.55	C22 C26	
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25	1	2	113781	151.55	C22 C26	
5	1	1	Anderson, Mr. Harry	male	48	0	0	19952	26.55	E12	

In [32]:
```
# Extract details of countries from 'America' to 'India' when sorted alphabetically
countries.sort_index().loc["America":"India"]
```

Out[32]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	S sup
Argentina	Latin America and Caribbean	5.929	0.056	6.040	5.819	9.962	0
Armenia	Commonwealth of Independent States	5.283	0.058	5.397	5.168	9.487	0
Australia	North America and ANZ	7.183	0.041	7.265	7.102	10.796	0
Austria	Western Europe	7.268	0.036	7.337	7.198	10.906	0
Azerbaijan	Commonwealth of Independent States	5.171	0.040	5.250	5.091	9.569	0
Bahrain	Middle East and North Africa	6.647	0.068	6.779	6.514	10.669	0
Bangladesh	South Asia	5.025	0.046	5.115	4.934	8.454	0
Belarus	Commonwealth of Independent States	5.534	0.047	5.625	5.442	9.853	0
Belgium	Western Europe	6.834	0.034	6.901	6.767	10.823	0
Benin	Sub-Saharan Africa	5.045	0.073	5.189	4.901	8.087	0
Bolivia	Latin America and Caribbean	5.716	0.053	5.819	5.613	9.046	0
Bosnia and Herzegovina	Central and Eastern Europe	5.813	0.050	5.911	5.715	9.590	0
Botswana	Sub-Saharan Africa	3.467	0.074	3.611	3.322	9.782	0
Brazil	Latin America and Caribbean	6.330	0.043	6.415	6.245	9.577	0
Bulgaria	Central and Eastern Europe	5.266	0.054	5.371	5.160	10.016	0
Burkina Faso	Sub-Saharan Africa	4.834	0.081	4.993	4.675	7.678	0

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	S sup
Burundi	Sub-Saharan Africa	3.775	0.107	3.985	3.565	6.635	0
Cambodia	Southeast Asia	4.830	0.067	4.963	4.698	8.360	0
Cameroon	Sub-Saharan Africa	5.142	0.074	5.288	4.996	8.189	0
Canada	North America and ANZ	7.103	0.042	7.185	7.021	10.776	0
Chad	Sub-Saharan Africa	4.355	0.094	4.540	4.171	7.364	0
Chile	Latin America and Caribbean	6.172	0.046	6.262	6.081	10.071	0
China	East Asia	5.339	0.029	5.397	5.281	9.673	0
Colombia	Latin America and Caribbean	6.012	0.061	6.132	5.892	9.557	0
Comoros	Sub-Saharan Africa	4.289	0.084	4.454	4.123	8.031	0
Congo (Brazzaville)	Sub-Saharan Africa	5.342	0.097	5.533	5.151	8.117	0
Costa Rica	Latin America and Caribbean	7.069	0.056	7.179	6.960	9.880	0
Croatia	Central and Eastern Europe	5.882	0.048	5.975	5.788	10.217	0
Cyprus	Western Europe	6.223	0.049	6.319	6.128	10.576	0
Czech Republic	Central and Eastern Europe	6.965	0.049	7.062	6.868	10.556	0
Denmark	Western Europe	7.620	0.035	7.687	7.552	10.933	0
Dominican Republic	Latin America and Caribbean	5.545	0.071	5.685	5.405	9.802	0
Ecuador	Latin America and Caribbean	5.764	0.057	5.875	5.653	9.313	0

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	S sup
Egypt	Middle East and North Africa	4.283	0.045	4.371	4.195	9.367	0
El Salvador	Latin America and Caribbean	6.061	0.065	6.188	5.933	9.054	0
Estonia	Central and Eastern Europe	6.189	0.038	6.263	6.115	10.481	0
Ethiopia	Sub-Saharan Africa	4.275	0.051	4.374	4.175	7.694	0
Finland	Western Europe	7.842	0.032	7.904	7.780	10.775	0
France	Western Europe	6.690	0.037	6.762	6.618	10.704	0
Gabon	Sub-Saharan Africa	4.852	0.075	4.998	4.706	9.603	0
Gambia	Sub-Saharan Africa	5.051	0.089	5.225	4.877	7.686	0
Georgia	Commonwealth of Independent States	4.891	0.054	4.998	4.785	9.585	0
Germany	Western Europe	7.155	0.040	7.232	7.077	10.873	0
Ghana	Sub-Saharan Africa	5.088	0.067	5.219	4.958	8.580	0
Greece	Western Europe	5.723	0.046	5.813	5.632	10.279	0
Guatemala	Latin America and Caribbean	6.435	0.073	6.577	6.292	9.053	0
Guinea	Sub-Saharan Africa	4.984	0.090	5.160	4.808	7.838	0
Haiti	Latin America and Caribbean	3.615	0.173	3.953	3.276	7.477	0
Honduras	Latin America and Caribbean	5.919	0.082	6.081	5.758	8.648	0

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	S sup
Hong Kong S.A.R. of China	East Asia	5.477	0.049	5.573	5.380	11.000	0
Hungary	Central and Eastern Europe	5.992	0.047	6.085	5.899	10.358	0
Iceland	Western Europe	7.554	0.059	7.670	7.438	10.878	0
India	South Asia	3.819	0.026	3.869	3.769	8.755	0

In [33]:
```
# Extract details about the 20th country from the last when sorted alphabetically
countries.sort_index(ascending=False).iloc[20]
```

Out[33]:
```
Regional indicator                          Western Europe
Ladder score                                         7.571
Standard error of ladder score                       0.036
upperwhisker                                         7.643
lowerwhisker                                           7.5
Logged GDP per capita                                11.117
Social support                                        0.942
Healthy life expectancy                                74.4
Freedom to make life choices                          0.919
Generosity                                            0.025
Perceptions of corruption                             0.292
Ladder score in Dystopia                               2.43
Explained by: Log GDP per capita                      1.566
Explained by: Social support                          1.079
Explained by: Healthy life expectancy                 0.816
Explained by: Freedom to make life choices            0.653
Explained by: Generosity                              0.204
Explained by: Perceptions of corruption               0.413
Dystopia + residual                                   2.839
Name: Switzerland, dtype: object
```

In [34]:
```
# Extract details about the 20th country from the last when sorted alphabetically,
countries.sort_index(ascending=False).iloc[[20]]
```

Out[34]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Social support
Switzerland	Western Europe	7.571	0.036	7.643	7.5	11.117	0.942

In [35]:
```
# Extract details about the countries that ranked between 30-40 when sorted by 'Log
countries.sort_values("Logged GDP per capita").iloc[29:39]
```

Out[35]:

Country name	Regional indicator	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	So supp
Cameroon	Sub-Saharan Africa	5.142	0.074	5.288	4.996	8.189	0.
Cambodia	Southeast Asia	4.830	0.067	4.963	4.698	8.360	0.
Kenya	Sub-Saharan Africa	4.607	0.072	4.747	4.466	8.361	0.
Bangladesh	South Asia	5.025	0.046	5.115	4.934	8.454	0.
Pakistan	South Asia	4.934	0.068	5.066	4.802	8.458	0.
Palestinian Territories	Middle East and North Africa	4.517	0.067	4.649	4.384	8.485	0.
Nigeria	Sub-Saharan Africa	4.759	0.052	4.861	4.658	8.533	0.
Kyrgyzstan	Commonwealth of Independent States	5.744	0.046	5.834	5.653	8.538	0.
Myanmar	Southeast Asia	4.426	0.052	4.527	4.324	8.541	0.
Mauritania	Sub-Saharan Africa	4.227	0.070	4.365	4.090	8.542	0.