In [2]:

```
# You can also drop columns using the drop() method.

import pandas as pd
scores = [
{'Subject':'Mathematics', 'Score':85, 'Grade': 'B', 'Remarks': 'Good',
},
{'Subject':'History', 'Score':98, 'Grade': 'A','Remarks':
'Excellent'},
{'Subject':'English', 'Score':76, 'Grade': 'C','Remarks': 'Fair'},
{'Subject':'Science', 'Score':72, 'Grade': 'C','Remarks': 'Fair'},
{'Subject':'Arts', 'Score':95, 'Grade': 'A','Remarks': 'Excellent'},
]
my_df = pd.DataFrame(scores)
my_df.head()
```

Out[2]:

	Subject	Score	Grade	Remarks
0	Mathematics	85	B	Good
1	History	98	A	Excellent
2	English	76	C	Fair
3	Science	72	C	Fair
4	Arts	95	A	Excellent

In [3]:

```
# To drop columns via the drop() method, you need to pass the
# list of columns to the drop() method, along with 1 as the value
# for the axis parameter of the drop method.
# The following script drops the columns Subject and Grade
# from our dummy dataframe.

my_df2 = my_df.drop(["Subject", "Grade"], axis = 1)
my_df2.head()
```

Out[3]:

	Score	Remarks
0	85	Good
1	98	Excellent
2	76	Fair
3	72	Fair
4	95	Excellent

In [4]:

```
# You can also drop the columns inplace from a dataframe using
# the inplace = True parameter value, as shown in the script
# below.

my_df.drop(["Subject", "Grade"], axis = 1, inplace = True)
my_df.head()
```

Out[4]:

	Score	Remarks
0	85	Good
1	98	Excellent
2	76	Fair
3	72	Fair
4	95	Excellent

In [5]:

```
# Filtering Rows and Columns with Filter Method

# The drop() method drops the unwanted records, and the
# filter() method performs the reverse tasks. It keeps the
# desired records from a set of records in a Pandas dataframe.
```

In [6]:

```
# To filter rows using the filter() method, you need to pass the
# list of row indexes to filter to the filter() method of the Pandas
# dataframe. Along with that, you need to pass 0 as the value
# for the axis attribute of the filter() method. Here is an
# example. The script below filters rows with indexes 1, 3, and 4
# from the Pandas dataframe.

my_df2 = my_df.filter([1,3,4], axis = 0)
my_df2.head()
```

Out[6]:

	Score	Remarks
1	98	Excellent
3	72	Fair
4	95	Excellent

In [7]:

```
# You can also reset indexes after filtering data using the reset_
# index() method, as shown in the following script:

my_df2 = my_df2.reset_index(drop=True)
my_df2.head()
```

Out[7]:

	Score	Remarks
0	98	Excellent
1	72	Fair
2	95	Excellent

In [8]:

```
# To filter columns using the filter() method, you need to pass
# the list of column names to the filter method. Furthermore,
# you need to set 1 as the value for the axis attribute.
# The script below filters the Score and Grade columns from
# your dummy dataframe.

my_df2 = my_df.filter(["Score","Grade"], axis = 1)
my_df2.head()
```

Out[8]:

	Score
0	85
1	98
2	76
3	72
4	95

In [9]:

```
# You can also sort records in your Pandas dataframe based on
# values in a particular column.
# you will be using the Titanic dataset, which
# you can import using the Seaborn library using the following
# script:

import matplotlib.pyplot as plt
import seaborn as sns
# sets the default style for plotting
sns.set_style("darkgrid")
titanic_data = sns.load_dataset('titanic')
titanic_data.head()
```

Out[9]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True

In [10]:

```
# To sort the Pandas dataframe, you can use the sort_values()
# function of the Pandas dataframe. The list of columns used for
# sorting needs to be passed to the by attribute of the sort_
# values() method.
# The following script sorts the Titanic dataset in ascending
# order of the passenger's age.

age_sorted_data = titanic_data.sort_values(by=['age'])
age_sorted_data.head()
```

Out[10]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
803	1	3	male	0.42	0	1	8.5167	C	Third	child	False
755	1	2	male	0.67	1	1	14.5000	S	Second	child	False
644	1	3	female	0.75	2	1	19.2583	C	Third	child	False
469	1	3	female	0.75	2	1	19.2583	C	Third	child	False
78	1	2	male	0.83	0	2	29.0000	S	Second	child	False

In [11]:

```
# To sort by descending order, you need to pass False as the
# value for the ascending attribute of the sort_values()
# function.
# The following script sorts the dataset by descending order of
# age.

age_sorted_data = titanic_data.sort_values(by=['age'], ascending = False)
age_sorted_data.head()
```

Out[11]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	d
630	1	1	male	80.0	0	0	30.0000	S	First	man	True	
851	0	3	male	74.0	0	0	7.7750	S	Third	man	True	N
493	0	1	male	71.0	0	0	49.5042	C	First	man	True	N
96	0	1	male	71.0	0	0	34.6542	C	First	man	True	
116	0	3	male	70.5	0	0	7.7500	Q	Third	man	True	N

In [12]:

```
# You can also pass multiple columns to the by attribute of the
# sort_values() function. In such a case, the dataset will be
# sorted by the first column, and in the case of equal values for
# two or more records, the dataset will be sorted by the second
# column and so on.
# The following script first sorts the data by Age and then by
# Fare, both by descending orders.

age_sorted_data = titanic_data.sort_values(by=['age','fare'], ascending =
False)
age_sorted_data.head()
```

Out[12]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	d
630	1	1	male	80.0	0	0	30.0000	S	First	man	True	
851	0	3	male	74.0	0	0	7.7750	S	Third	man	True	N
493	0	1	male	71.0	0	0	49.5042	C	First	man	True	N
96	0	1	male	71.0	0	0	34.6542	C	First	man	True	
116	0	3	male	70.5	0	0	7.7500	Q	Third	man	True	N