In [1]:

```
# Indexing and Slicing Using iloc Function

# You can also use the iloc function for selecting and slicing
# records using index values. However, unlike the loc function,
# where you can pass both the string indexes and integer
# indexes, you can only pass the integer index values to the iloc
# function.

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[1]:

	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 [2]:

```
# Let's filter the record at index 3 (row 4).

my_df.iloc[3]

# The script below returns a series.
```

Out[2]:

```
Subject    Science
Score           72
Grade            C
Remarks       Fair
Name: 3, dtype: object
```

In [3]:

```
# If you want to select records from a single column as a
# dataframe, you need to specify the index inside the square
# brackets and then those square brackets inside the square
# brackets that follow the iloc function, as shown below.

my_df.iloc[[3]]
```

Out[3]:

	Subject	Score	Grade	Remarks
3	Science	72	C	Fair

In [4]:

```
# You can pass multiple indexes to the iloc function to select
# multiple records. Here is an example:

my_df.iloc[[2,3]]
```

Out[4]:

	Subject	Score	Grade	Remarks
2	English	76	C	Fair
3	Science	72	C	Fair

In [5]:

```
# You can also pass a range of indexes. In this case, the records
# from the lower range to 1 less than the upper range will be
# selected.
# For instance, the script below returns records from index 2 to
# index 3 (1 less than 4).

my_df.iloc[2:4]
```

Out[5]:

	Subject	Score	Grade	Remarks
2	English	76	C	Fair
3	Science	72	C	Fair

In [6]:

```
# In addition to specifying indexes, you can also pass column
# numbers (starting from 0) to the iloc method.
# The following script returns values from columns number 0
# and 1 for the records at indexes 2 and 3.

my_df.iloc[[2,3], [0,1]]
```

Out[6]:

	Subject	Score
2	English	76
3	Science	72

In [7]:

```
# You can also pass a range of indexes and columns to select.
# The script below selects columns 1 and 2 and rows 2 and 3.

my_df.iloc[2:4, 0:2]
```

Out[7]:

	Subject	Score
2	English	76
3	Science	72

In [8]:

```
# Dropping Rows and Columns with the drop() Method

# Apart from selecting columns using the loc and iloc functions,
# you can also use the drop() method to drop unwanted rows
# and columns from your dataframe while keeping the rest of
# the rows and columns.

# The following script drops records at indexes 1 and 4.

my_df2 = my_df.drop([1,4])
my_df2.head()
```

Out[8]:

	Subject	Score	Grade	Remarks
0	Mathematics	85	B	Good
2	English	76	C	Fair
3	Science	72	C	Fair

In [9]:

```
# From the output above, you can see that the indexes are not
# in sequence since you have dropped indexes 1 and 4.
# You can reset dataframe indexes starting from 0, using the
# reset_index().
# Let's call the reset_index() method on the my_df2 dataframe.
# Here, the value True for the inplace parameter specifies that
# you want to remove the records in place without assigning the
# result to any new variable.

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

Out[9]:

	index	Subject	Score	Grade	Remarks
0	0	Mathematics	85	B	Good
1	2	English	76	C	Fair
2	3	Science	72	C	Fair

In [10]:

```
# The above output shows that the indexes have been reset.
# Also, you can see that a new column index has been added,
# which contains the original index. If you only want to reset
# new indexes without creating a new column named index, you
# can do so by passing True as the value for the drop parameter
# of the reset_index method.
# Let's again drop some rows and reset the index using the
# reset_index() method by passing True as the value for the
# drop attribute. See the following two scripts:

my_df2 = my_df.drop([1,4])
my_df2.head()
```

Out[10]:

	Subject	Score	Grade	Remarks
0	Mathematics	85	B	Good
2	English	76	C	Fair
3	Science	72	C	Fair

In [11]:

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

Out[11]:

	Subject	Score	Grade	Remarks
0	Mathematics	85	B	Good
1	English	76	C	Fair
2	Science	72	C	Fair

In [12]:

```
# By default, the drop method doesn't drop rows in place.
# Instead, you have to assign the result of the drop() method to
# another variable that contains the records with dropped
# results.
# For instance, if you drop the records at indexes 1, 3, and 4
# using the following script and then print the dataframe
# header, you will see that the rows are not removed from the
# original dataframe.

my_df.drop([1,3,4])
my_df.head()
```

Out[12]:

	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 [13]:

```
# If you want to drop rows in place, you need to pass True as
# the value for the inplace attribute, as shown in the script
# below:

my_df.drop([1,3,4], inplace = True)
my_df.head()
```

Out[13]:

	Subject	Score	Grade	Remarks
0	Mathematics	85	B	Good
2	English	76	C	Fair