In [1]:

```
# Once you have loaded data into your Pandas dataframe, you
# might need to further manipulate the data and perform a
# variety of functions such as filtering certain columns, dropping
# the others, selecting a subset of rows or columns, sorting the
# data, finding unique values, and so on.
# Indexing refers to fetching data using index or column
#information of a Pandas dataframe. Slicing, on the other hand,
# refers to slicing a Pandas dataframe using indexing
# techniques.
```

In [2]:

```
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[2]:

	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 [3]:

```
# One of the simplest ways to select data from various columns
# is by using square brackets. To get column data in the form of
# a series from a Pandas dataframe, you need to pass the
# column name inside square brackets that follow the Pandas
# dataframe name.
# The following script selects records from the class column of
# the Titanic dataset.

print(titanic_data["class"])
type(titanic_data["class"])
```

```
0       Third
1       First
2       Third
3       First
4       Third
        ...
886    Second
887     First
888     Third
889     First
890     Third
Name: class, Length: 891, dtype: category
Categories (3, object): ['First', 'Second', 'Third']
```

Out[3]:

```
pandas.core.series.Series
```

In [4]:

```
# You can select multiple columns by passing a list of column
# names inside a string to the square brackets. You will then get
# a Pandas dataframe with the specified columns, as shown
# below.

print(type(titanic_data[["class", "sex", "age"]]))
titanic_data[["class", "sex", "age"]]
```

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

Out[4]:

	class	sex	age
0	Third	male	22.0
1	First	female	38.0
2	Third	female	26.0
3	First	female	35.0
4	Third	male	35.0
...	...	...	...
886	Second	male	27.0
887	First	female	19.0
888	Third	female	NaN
889	First	male	26.0
890	Third	male	32.0

891 rows × 3 columns

In [5]:

```
# You can also filter rows based on some column values. For
# doing this, you need to pass the condition to the filter inside
# the square brackets. For instance, the script below returns all
# records from the Titanic dataset where the sex column
# contains the value “male.”

my_df = titanic_data[titanic_data["sex"] == "male"]
my_df.head()
```

Out[5]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	dec
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	Na
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	Na
5	0	3	male	NaN	0	0	8.4583	Q	Third	man	True	Na
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	
7	0	3	male	2.0	3	1	21.0750	S	Third	child	False	Na

In [6]:

```
# You can specify multiple conditions inside the square
# brackets. The following script returns those records where the
# sex column contains the string “male,” while the class column
# contains the string “First.”

my_df = titanic_data[(titanic_data["sex"] == "male") &
(titanic_data["class"] == "First")]
my_df.head()
```

Out[6]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	d
6	0	1	male	54.0	0	0	51.8625	S	First	man	True	
23	1	1	male	28.0	0	0	35.5000	S	First	man	True	
27	0	1	male	19.0	3	2	263.0000	S	First	man	True	
30	0	1	male	40.0	0	0	27.7208	C	First	man	True	N
34	0	1	male	28.0	1	0	82.1708	C	First	man	True	N

In [7]:

```
# You can also use the isin() function to specify a range of
# values to filter records. For instance, the script below filters all
# records where the age column contains the values 20, 21, or
# 22.

ages = [20,21,22]
age_dataset = titanic_data[titanic_data["age"].isin(ages)]
age_dataset.head()
```

Out[7]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male
0	0	3	male	22.0	1	0	7.25	S	Third	man	True
12	0	3	male	20.0	0	0	8.05	S	Third	man	True
37	0	3	male	21.0	0	0	8.05	S	Third	man	True
51	0	3	male	21.0	0	0	7.80	S	Third	man	True
56	1	2	female	21.0	0	0	10.50	S	Second	woman	False

In [8]:

```
# Indexing and Slicing Using loc Function

# The loc function from the Pandas dataframe can also be used
# to filter records in the Pandas dataframe.
# To create a dummy dataframe used as an example in this
# section, run the following script:

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[8]:

	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 [9]:

```
# Let's now see how to filter records. To filter the row at the
# second index in the my_dfdataframe, you need to pass 2
# inside the square brackets that follow the loc function. Here is
# an example:

print(my_df.loc[2])
type(my_df.loc[2])

# In the output below, you can see data from the row at the
# second index (row 3) in the form of a series.
```

```
Subject    English
Score           76
Grade            C
Remarks       Fair
Name: 2, dtype: object
```

Out[9]:

```
pandas.core.series.Series
```

In [10]:

```
# You can also specify the range of indexes to filter records
# using the loc function. For instance, the following script filters
# records from index 2 to 4.

my_df.loc[2:4]
```

Out[10]:

	Subject	Score	Grade	Remarks
2	English	76	C	Fair
3	Science	72	C	Fair
4	Arts	95	A	Excellent

In [11]:

```
# Along with filtering rows, you can also specify which columns
# to filter with the loc function.
# The following script filters the values in columns Grade and
# Score in the rows from index 2 to 4.

my_df.loc[2:4, ["Grade", "Score"]]
```

Out[11]:

	Grade	Score
2	C	76
3	C	72
4	A	95