

## Note:

To learn to apply `loc[ ]` and `iloc[ ]` properties on DataFrame and CSV files -

Refer Lecture Notes 3.a , 3.b , 3.c , 3.d as demo files

*Then come back to Lecture Notes 3 to learn Indexing and Selection*

## Indexing/Selection in Pandas

- Indexing in pandas means simply selecting particular rows and columns of data from a DataFrame.
- Indexing could mean selecting all the rows and some of the columns, some of the rows and all of the columns, or some of each of the rows and columns.
- Indexing can also be called as Subset Selection.

## Pandas support three types of indexing operators they are:

**Dataframe.[ ]**: indexing operator

**Dataframe.loc[ ]** : (label-based) To select rows or columns using labels.

**Dataframe.iloc[ ]** : (integer-based) To select rows or columns using indices.

Collectively, they are called the indexers.

## Methods of Indexing/Selection using [ ], .loc[ ], .iloc[ ]

- There are a lot of ways to pull the elements, rows, and columns from a DataFrame.
- There above mentioned are some indexing method in Pandas which help in getting an element from a DataFrame.
- These indexing methods appear very similar but behave very differently.

## A) Indexing/Selecting a Dataframe using indexing operator [ ] :

Indexing operator is used to refer to the square brackets following an object. The `.loc` and `.iloc` indexers also use the indexing operator to make selections. In this indexing operator to refer to `df[ ]`.

### ▼ Selecting a single columns

In order to select a single column, we simply put the name of the column in-between the brackets

```
# importing pandas package
import pandas as pd

# making data frame from csv file
df = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# selecting single column 'Team' by indexing operator
a = df['Team']
```

```
print(a)
```

```
Name
Aaron Brooks      Chicago Bulls
Aaron Gordon      Orlando Magic
Aaron Harrison    Charlotte Hornets
Adreian Payne     Minnesota Timberwolves
Al Horford        Atlanta Hawks
...
Xavier Munford    Memphis Grizzlies
Zach LaVine       Minnesota Timberwolves
Zach Randolph     Memphis Grizzlies
Zaza Pachulia     Dallas Mavericks
NaN              NaN
Name: Team, Length: 458, dtype: object
```

## ▼ Selecting multiple columns

In order to select multiple columns, we have to pass a list of columns in an indexing operator.

```
# importing pandas package
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving multiple columns by indexing operator
first = data[["Age", "College", "Salary"]]

print(first)
```

	Age	College	Salary
Name			
Aaron Brooks	31.0	Oregon	2250000.0
Aaron Gordon	20.0	Arizona	4171680.0
Aaron Harrison	21.0	Kentucky	525093.0
Adreian Payne	25.0	Michigan State	1938840.0
Al Horford	30.0	Florida	12000000.0
...	...	...	...
Xavier Munford	24.0	Rhode Island	NaN
Zach LaVine	21.0	UCLA	2148360.0
Zach Randolph	34.0	Michigan State	9638555.0
Zaza Pachulia	32.0	NaN	5200000.0
NaN	NaN	NaN	NaN

```
[458 rows x 3 columns]
```

## B) Indexing/Selection a DataFrame using .loc[ ] :

This function selects data by the label of the rows and columns.

The df.loc indexer selects data in a different way than just the indexing operator. It can select subsets of rows or columns. It can also simultaneously select subsets of rows and columns.

## ▼ Selecting a single row

In order to select a single row using .loc[ ], we put a single row label in a .loc function.

```
# importing pandas package
import pandas as pd
```

```
# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving row by loc method
first = data.loc["Avery Bradley"]
second = data.loc["R.J. Hunter"]

print(first, "\n\n", second)
```

```
Unnamed: 0      0
Team      Boston Celtics
Number      0.0
Position    PG
Age         25.0
Height      6-2
Weight      180.0
College     Texas
Salary      7730337.0
Name: Avery Bradley, dtype: object
```

```
Unnamed: 0      3
Team      Boston Celtics
Number      28.0
Position    SG
Age         22.0
Height      6-5
Weight      185.0
College     Georgia State
Salary      1148640.0
Name: R.J. Hunter, dtype: object
```

## ▼ Selecting multiple rows

In order to select multiple rows, we put all the row labels in a list and pass that to .loc function.

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving multiple rows by loc method
first = data.loc[["Avery Bradley", "R.J. Hunter"]]

print(first)
```

```
      Unnamed: 0      Team  Number  Position  Age  Height  \
Name
Avery Bradley      0  Boston Celtics      0.0      PG  25.0    6-2
R.J. Hunter        3  Boston Celtics      28.0      SG  22.0    6-5

      Weight      College      Salary
Name
Avery Bradley  180.0      Texas  7730337.0
R.J. Hunter    185.0  Georgia State  1148640.0
```

## ▼ Selecting two rows and three columns

In order to select two rows and three columns, we select a two rows which we want to select and three columns and put it in a separate list like this:

```
Dataframe.loc[["row1", "row2"], ["column1", "column2", "column3"]]
```

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving two rows and three columns by loc method
first = data.loc[["Avery Bradley", "R.J. Hunter"], ["Team", "Number", "Position"]]

print(first)
```

	Team	Number	Position
Name			
Avery Bradley	Boston Celtics	0.0	PG
R.J. Hunter	Boston Celtics	28.0	SG

## ▼ Selecting all of the rows and some columns

In order to select all of the rows and some columns, we use single colon [:] to select all of rows and list of some columns which we want to select like this:

```
Dataframe.loc[:, ["column1", "column2", "column3"]]
```

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving all rows and some columns by loc method
first = data.loc[:, ["Team", "Number", "Position"]]

print(first)
```

	Team	Number	Position
Name			
Aaron Brooks	Chicago Bulls	0.0	PG
Aaron Gordon	Orlando Magic	0.0	PF
Aaron Harrison	Charlotte Hornets	9.0	SG
Adreian Payne	Minnesota Timberwolves	33.0	PF
Al Horford	Atlanta Hawks	15.0	C
...	...	...	...
Xavier Munford	Memphis Grizzlies	14.0	PG
Zach LaVine	Minnesota Timberwolves	8.0	PG
Zach Randolph	Memphis Grizzlies	50.0	PF
Zaza Pachulia	Dallas Mavericks	27.0	C
NaN	NaN	NaN	NaN

[458 rows x 3 columns]

## C) Indexing/Selection a DataFrame using .iloc[ ] :

This function allows us to retrieve rows and columns by position.

In order to do that, we'll need to specify the positions of the rows that we want, and the positions of the columns that we want as well.

The df.iloc indexer is very similar to df.loc but only uses integer locations to make its selections.

## ▼ Selecting a single row

In order to select a single row using .iloc[], we can pass a single integer to .iloc[] function.

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving rows by iloc method
row2 = data.iloc[3]

print(row2)
```

```
Unnamed: 0      404
Team      Minnesota Timberwolves
Number      33.0
Position     PF
Age          25.0
Height       6-10
Weight       237.0
College      Michigan State
Salary      1938840.0
Name: Adreian Payne, dtype: object
```

## ▼ Selecting multiple rows

In order to select multiple rows, we can pass a list of integer to .iloc[] function.

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving multiple rows by iloc method
row2 = data.iloc [[3, 5, 7]]
```

row2



Unnamed:  
0

Team Number Position Age Height Weight College Salary

Name

Adreian Payne	404	Minnesota Timberwolves	33.0	PF	25.0	6-10	237.0	Michigan State	1938840.0
Al Jefferson	330	Charlotte Hornets	25.0	C	31.0	6-10	289.0	NaN	13500000.0

## ▼ Selecting two rows and two columns

In order to select two rows and two columns, we create a list of 2 integer for rows and list of 2 integer for columns then pass to a .iloc[] function.

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving two rows and two columns by iloc method
row2 = data.iloc [[3, 4], [1, 2]]
```

```
print(row2)
```

Name	Team	Number
Adreian Payne	Minnesota Timberwolves	33.0
Al Horford	Atlanta Hawks	15.0

## ▼ Selecting all the rows and some columns

In order to select all rows and some columns, we use single colon [:] to select all of rows and for columns we make a list of integer then pass to a .iloc[] function.

```
import pandas as pd

# making data frame from csv file
data = pd.read_csv("/content/drive/MyDrive/nba.csv", index_col = "Name")

# retrieving all rows and some columns by iloc method
row2 = data.iloc[:, [1, 2]]

print(row2)
```

Name	Team	Number
Aaron Brooks	Chicago Bulls	0.0
Aaron Gordon	Orlando Magic	0.0
Aaron Harrison	Charlotte Hornets	9.0
Adreian Payne	Minnesota Timberwolves	33.0
Al Horford	Atlanta Hawks	15.0
...	...	...
Xavier Munford	Memphis Grizzlies	14.0
Zach LaVine	Minnesota Timberwolves	8.0
Zach Randolph	Memphis Grizzlies	50.0
Zaza Pachulia	Dallas Mavericks	27.0
NaN	NaN	NaN

[458 rows x 2 columns]

## ▼ QUICK REVISION (ADDITIONAL POINTS)

### ▼ How to Select Rows by Index (Position/Label)

In order to deal with rows, we can perform basic operations on rows like selecting, deleting, adding and renaming.

- Use `pandas.DataFrame.iloc[]` & `pandas.DataFrame.loc[]` to select a single row or multiple rows from DataFrame by integer Index and by row indices respectively.
- `iloc[]` operator can accept single index, multiple indexes from the list, indexes by a range, and many more.
- `loc[]` operator is explicitly used with labels that can accept single index labels, multiple index labels from the list, indexes by a range (between two indexes labels), and many more.
- When using `.iloc[]` or `loc[]` with an index that doesn't exist it returns an error.

## Let's create a DataFrame with a few rows and columns and execute some examples using both loc and iloc.

```
import pandas as pd
import numpy as np

technologies = {
    'Courses':["Spark","PySpark","Hadoop","Python","pandas","Oracle","Java"],
    'Fee' :[20000,25000,26000,22000,24000,21000,22000],
    'Duration':['30days','40days','35days','40days',np.nan,None,'55days'],
    'Discount':[1000,2300,1500,1200,2500,2100,2000]
}

index_labels=['r1','r2','r3','r4','r5','r6','r7']

df = pd.DataFrame(technologies,index=index_labels)

print(df)
```

	Courses	Fee	Duration	Discount
r1	Spark	20000	30days	1000
r2	PySpark	25000	40days	2300
r3	Hadoop	26000	35days	1500
r4	Python	22000	40days	1200
r5	pandas	24000	NaN	2500
r6	Oracle	21000	None	2100
r7	Java	22000	55days	2000

## Quick Examples of Select Rows From DataFrame by Index Position & Labels

```
# Below are quick example
# Select Rows by Integer Index
df2 = df.iloc[2]      # Select Row by Index
df2 = df.iloc[[2,3,6]] # Select Rows by Index List
df2 = df.iloc[1:5]    # Select Rows by Integer Index Range
df2 = df.iloc[:1]     # Select First Row
df2 = df.iloc[:3]     # Select First 3 Rows
df2 = df.iloc[-1:]    # Select Last Row
df2 = df.iloc[-3:]    # Select Last 3 Row
df2 = df.iloc[::2]    # Selects alternate rows

# Select Rows by Index Labels
df2 = df.loc['r2']     # Select Row by Index Label
df2 = df.loc[['r2','r3','r6']] # Select Rows by Index Label List
df2 = df.loc['r1':'r5'] # Select Rows by Label Index Range
df2 = df.loc['r1':'r5'] # Select Rows by Label Index Range
df2 = df.loc['r1':'r5':2] # Select Alternate Rows with in Index Labels
```

## How to select Columns by Name or Index

- Use `DataFrame.loc[]` and `DataFrame.iloc[]` to select a single column or multiple columns from pandas DataFrame by column names/label or index position respectively.
- `loc[]` is used with column labels/names and `iloc[]` is used with column index/position.

## Quick Examples of Select Columns From DataFrame by Index Position & Labels

# Below are quick example

# By using df[] Notation

```
df2 = df[["Courses","Fee","Duration"]] # select multile columns
```

# Using loc[] to take column slices

```
df2 = df.loc[:, ["Courses","Fee","Duration"]] # Selecte multiple columns
```

```
df2 = df.loc[:, ["Courses","Fee","Discount"]] # Select Random columns
```

```
df2 = df.loc[:, 'Fee':'Discount'] # Select columns between two columns
```

```
df2 = df.loc[:, 'Duration':] # Select columns by range
```

```
df2 = df.loc[:, : 'Duration'] # Select columns by range
```

```
df2 = df.loc[:, ::2] # Select every alternate column
```

# Using iloc[] to select column by Index

```
df2 = df.iloc[:, [1,3]] # Select columns by Index
```

```
df2 = df.iloc[:, 1:4] # Select between indexes 1 and 4 (2,3,4)
```

```
df2 = df.iloc[:, 2:] # Select From 3rd to end
```

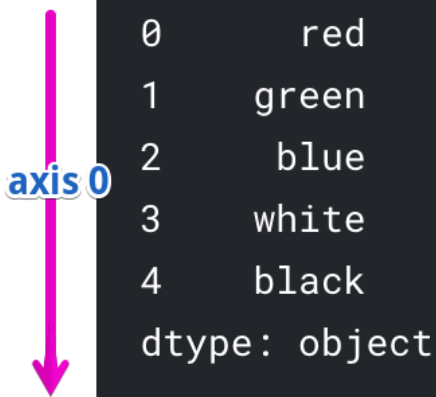
```
df2 = df.iloc[:, :2] # Select First Two Columns
```

## What are Axis in Pandas?

### ▼ A) Axes in Series

- ▼ Series object has only “axis 0” because it has only one dimension.

```
[1]: import pandas as pd
      pd.Series(['red', 'green', 'blue', 'white', 'black'])
```



```
0    red
1  green
2   blue
3  white
4  black
dtype: object
```

The arrow on the image displays “axis 0” and its direction for the Series object.

Usually, in Python, one-dimensional structures are displayed as a row of values. On the contrary, here we see that Series is displayed as a column of values.

Each cell in Series is accessible via index value along the “axis 0”. For our Series object indexes are: 0, 1, 2, 3, 4.

### ▼ B) Axes in DataFrame

DataFrame is a two-dimensional data structure akin to SQL table or Excel spreadsheet. It has columns and rows.



Its columns are made of separate Series objects. Let's see an example:

```
[1]: import pandas as pd
      srs_a = pd.Series([10,30,60,80,90])
      srs_b = pd.Series(['red', 'green', 'blue', 'white', 'black'])
      df = pd.DataFrame({'a': srs_a, 'b': srs_b})
      df
```

A diagram illustrating the axes of a DataFrame. A table with 5 rows and 2 columns is shown. The columns are labeled 'a' and 'b'. The rows are indexed 0 to 4. A vertical pink arrow on the left is labeled 'axis 0' and points downwards. A horizontal pink arrow on the top is labeled 'axis 1' and points to the right.

	a	b
0	10	red
1	30	green
2	60	blue
3	80	white
4	90	black

A DataFrame object has two axes: “axis 0” and “axis 1”. “axis 0” represents rows and “axis 1” represents columns. Now it's clear that Series and DataFrame share the same direction for “axis 0” – it goes along rows direction.

Our DataFrame object has 0, 1, 2, 3, 4 indexes along the “axis 0”, and additionally, it has “axis 1” indexes which are: ‘a’ and ‘b’.

## ▼ Note:

There are a lot of different API calls for Series and DataFrame objects which accept “axis” parameter.

- **Series object has only one axis, so this parameter always equals 0 for it. Thus, you can omit it, because it does not affect the result.**
- **On the contrary, DataFrame has two axes, and “axis” parameter determines along which axis an operation should be performed.**

For example, .sum can be applied along “axis 0”. That means, .sum operation calculates a sum for each column

- If you prefer regular names instead of numbers, each axis has a string alias.
- **“axis 0” has two aliases: ‘index’ and ‘rows’.**
- **“axis 1” has only one: ‘columns’.**

**You can use these aliases instead of numbers**

[Colab paid products](#) - [Cancel contracts here](#)

✓ 0s completed at 15:14

