Machine Learning Diploma

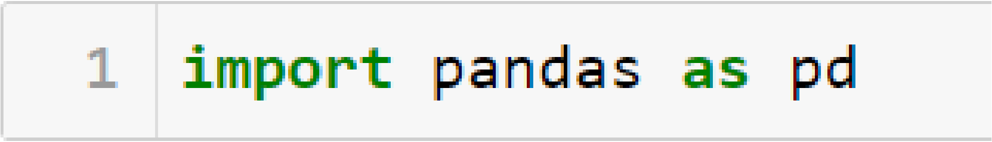
Session 4: Pandas

Agenda:

1	Pandas Basics
2	EDA Using Pandas
3	Data Manipulation
4	Indexing & Slicing
5	Inserting/Dropping DataFrame Columns & Rows
6	Null Values

1. Pandas Basics

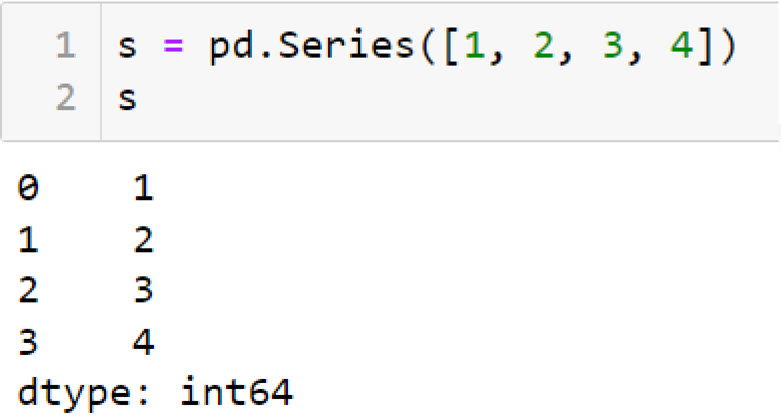
Import:



```python
import pandas as pd
```

Create Series:

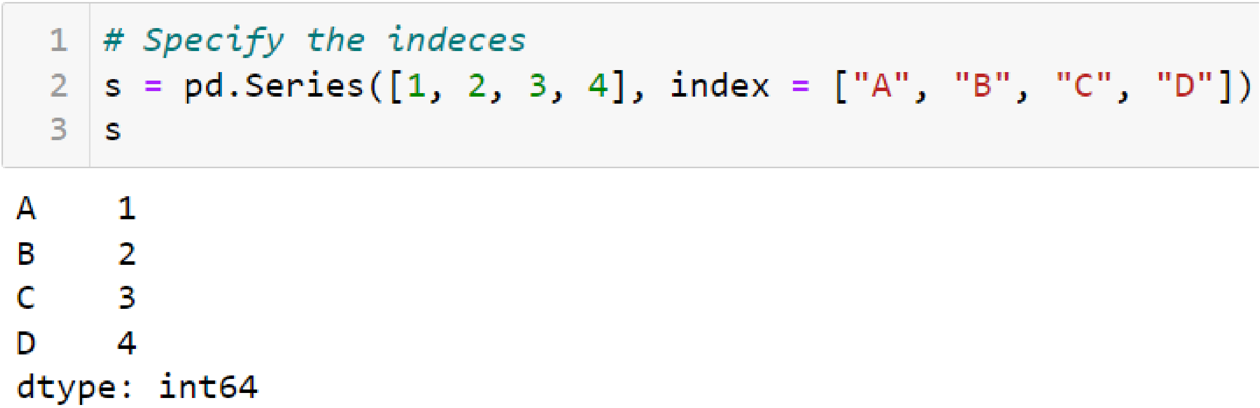
With default index



```python
s = pd.Series([1, 2, 3, 4])
s
```

```
0    1
1    2
2    3
3    4
dtype: int64
```

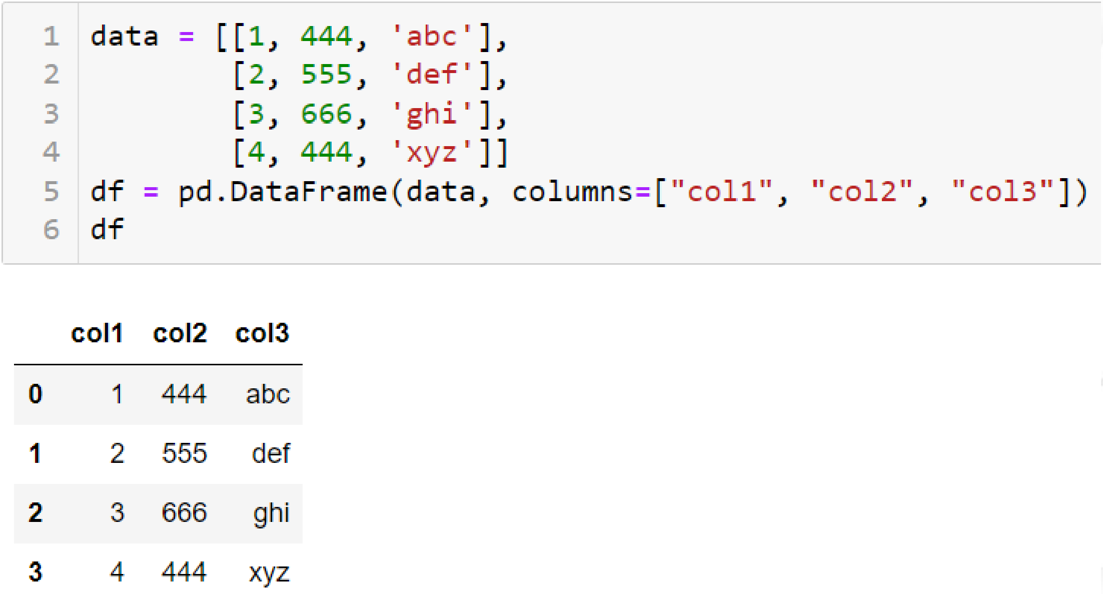
Specify the index



```python
# Specify the indeces
s = pd.Series([1, 2, 3, 4], index = ["A", "B", "C", "D"])
s
```

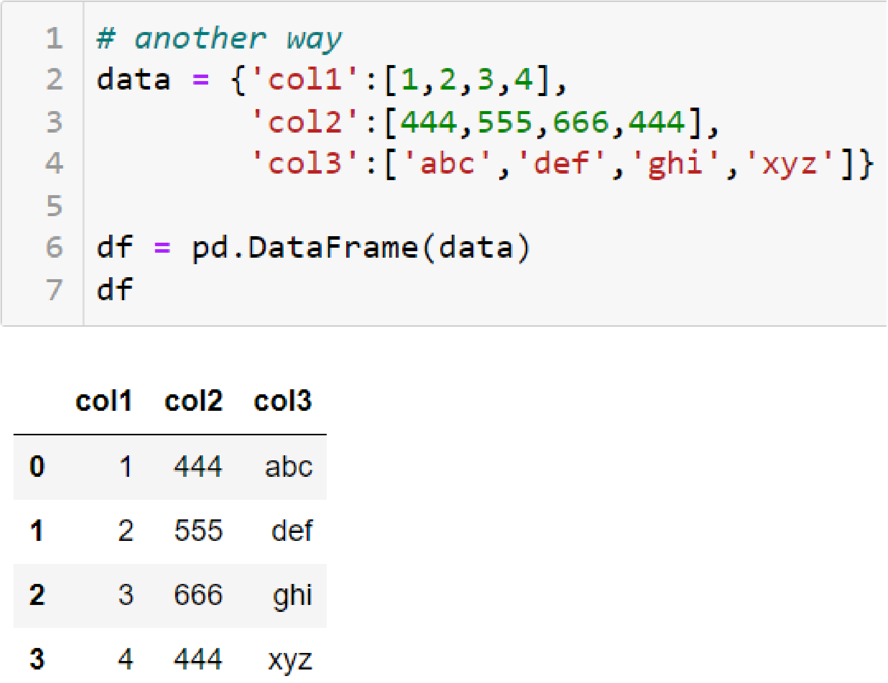
```
A    1
B    2
C    3
D    4
dtype: int64
```

Create DataFrame:



```python
data = [[1, 444, 'abc'],
        [2, 555, 'def'],
        [3, 666, 'ghi'],
        [4, 444, 'xyz']]
df = pd.DataFrame(data, columns=["col1", "col2", "col3"])
df
```

```
   col1  col2  col3
0     1   444   abc
1     2   555   def
2     3   666   ghi
3     4   444   xyz
```

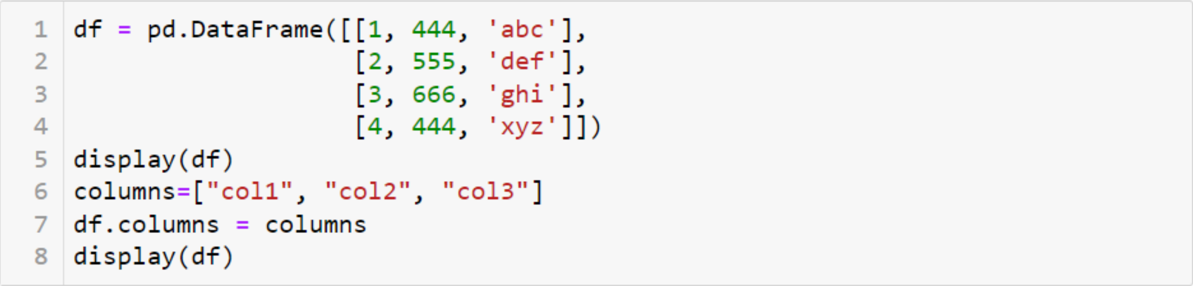


```python
# another way
data = {'col1':[1,2,3,4],
        'col2':[444,555,666,444],
        'col3':['abc','def','ghi','xyz']}

df = pd.DataFrame(data)
df
```

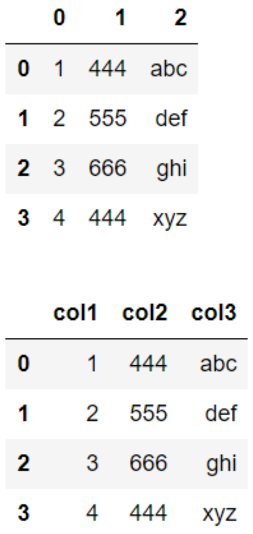
```
   col1  col2  col3
0     1   444   abc
1     2   555   def
2     3   666   ghi
3     4   444   xyz
```

Rename DataFrame Columns & index:

Rename DataFrame Columns

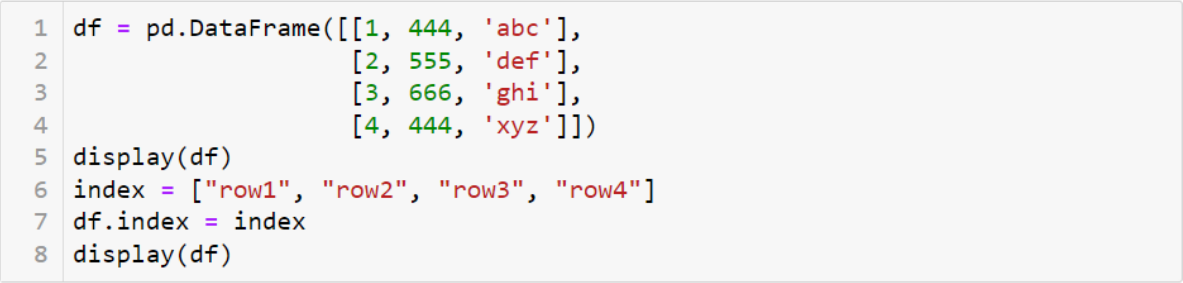


```
df = pd.DataFrame([[1, 444, 'abc'],
                   [2, 555, 'def'],
                   [3, 666, 'ghi'],
                   [4, 444, 'xyz']])
display(df)
columns=["col1", "col2", "col3"]
df.columns = columns
display(df)
```

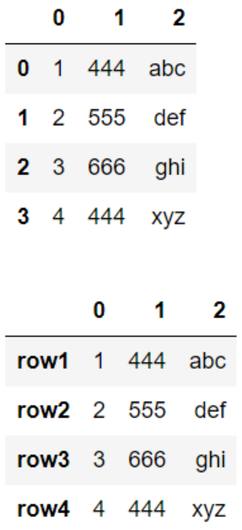


```
   0    1    2
0  1  444  abc
1  2  555  def
2  3  666  ghi
3  4  444  xyz

   col1  col2  col3
0     1   444   abc
1     2   555   def
2     3   666   ghi
3     4   444   xyz
```

Rename DataFrame index



```
df = pd.DataFrame([[1, 444, 'abc'],
                   [2, 555, 'def'],
                   [3, 666, 'ghi'],
                   [4, 444, 'xyz']])
display(df)
index = ["row1", "row2", "row3", "row4"]
df.index = index
display(df)
```



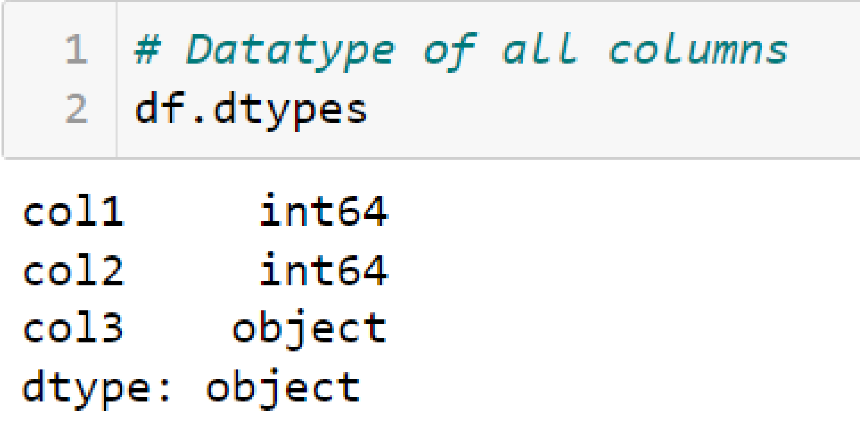
```
   0    1    2
0  1  444  abc
1  2  555  def
2  3  666  ghi
3  4  444  xyz

      0    1    2
row1  1  444  abc
row2  2  555  def
row3  3  666  ghi
row4  4  444  xyz
```

Pandas Dtypes:

Bool ➢ Represents Numerical datatypes with True & False values.	**Int** ➢ Represents Numerical datatypes with integer values.	**Float** ➢ Represents Numerical datatypes with continuous values.
Category ➢ Represents Categorical datatypes.	**Object** ➢ Is a mix of categorical datatypes & Numerical datatypes. ➢ Can carry any python object; such as, lists, tuples, strings, etc.	

Pandas Dtypes:

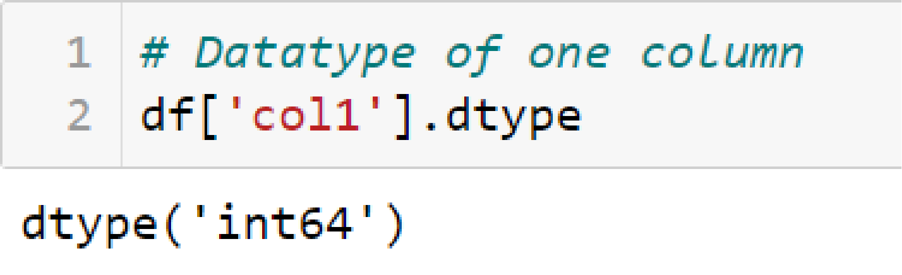
Get Datatypes of all columns



```python
# Datatype of all columns
df.dtypes
```

```
col1      int64
col2      int64
col3     object
dtype: object
```

Get Datatype of one column

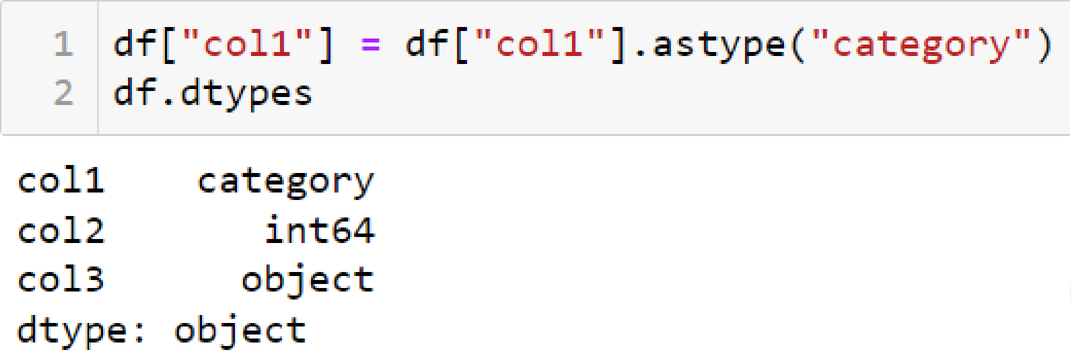


```python
# Datatype of one column
df['col1'].dtype
```

```
dtype('int64')
```

Change Datatype:

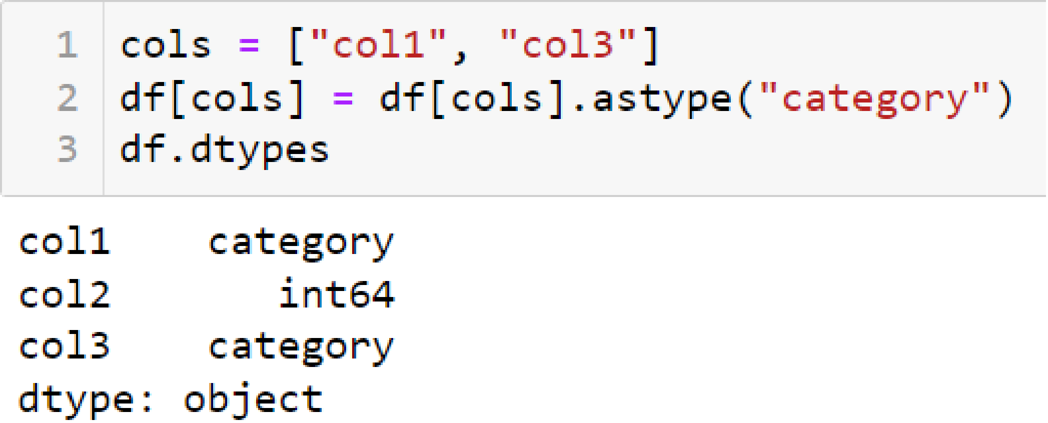
Change Datatype of one column



```python
df["col1"] = df["col1"].astype("category")
df.dtypes
```

```
col1    category
col2       int64
col3      object
dtype: object
```

Change Datatypes of group of columns



```python
cols = ["col1", "col3"]
df[cols] = df[cols].astype("category")
df.dtypes
```

```
col1    category
col2       int64
col3    category
dtype: object
```

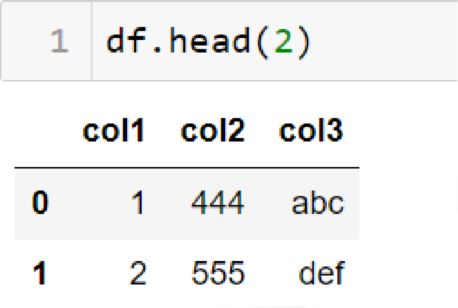
2. EDA using Pandas

EDA using Pandas:

- EDA is about exploring and understanding the data and getting insights about it.
- EDA is short for Exploratory Data Analysis.
- Pandas provides built-in methods and features that helps us to answer different questions about the data.

EDA using Pandas:

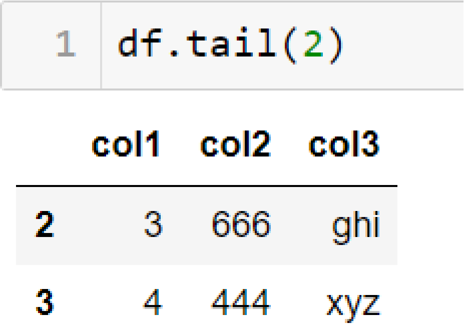
Get the first n rows of DataFrame



```python
df.head(2)
```

	col1	col2	col3
0	1	444	abc
1	2	555	def

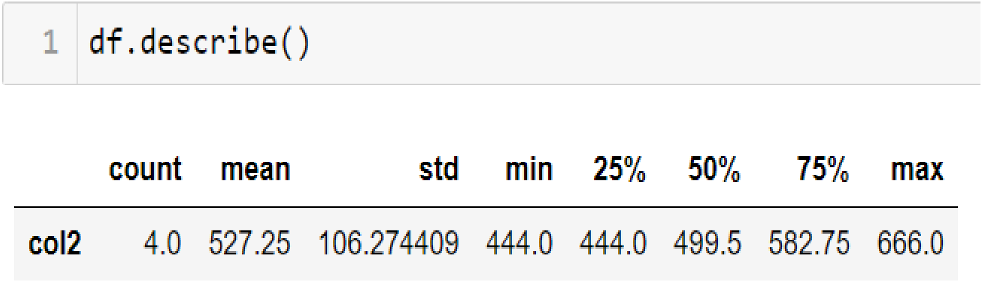
Get the last n rows of DataFrame



```python
df.tail(2)
```

	col1	col2	col3
2	3	666	ghi
3	4	444	xyz

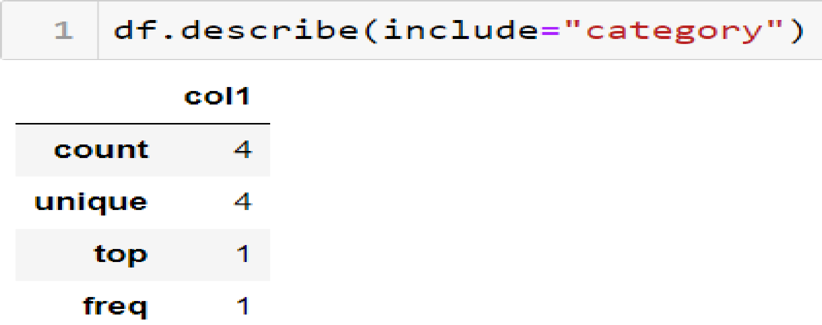
Get Statistical Measures about Numerical Columns



```python
df.describe()
```

	count	mean	std	min	25%	50%	75%	max
col2	4.0	527.25	106.274409	444.0	444.0	499.5	582.75	666.0

Get Statistical Measures about Categorical Columns

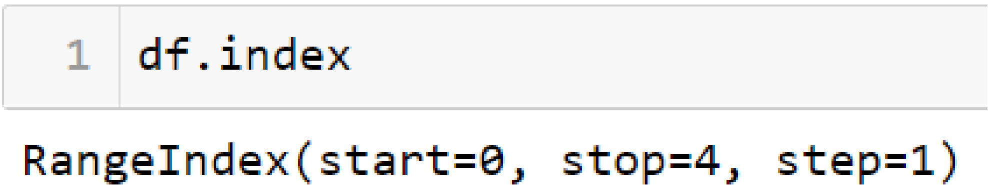


```python
df.describe(include="category")
```

	col1
count	4
unique	4
top	1
freq	1

EDA using Pandas:

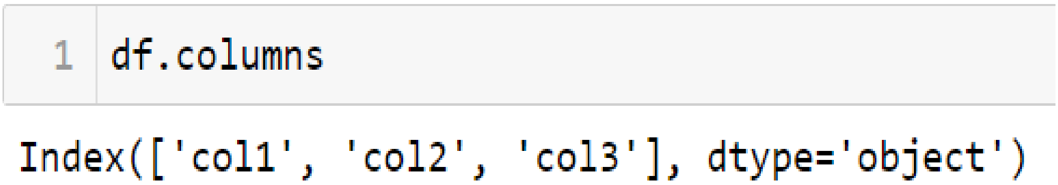
Get DataFrame Rows Names



```python
df.index
```

```
RangeIndex(start=0, stop=4, step=1)
```

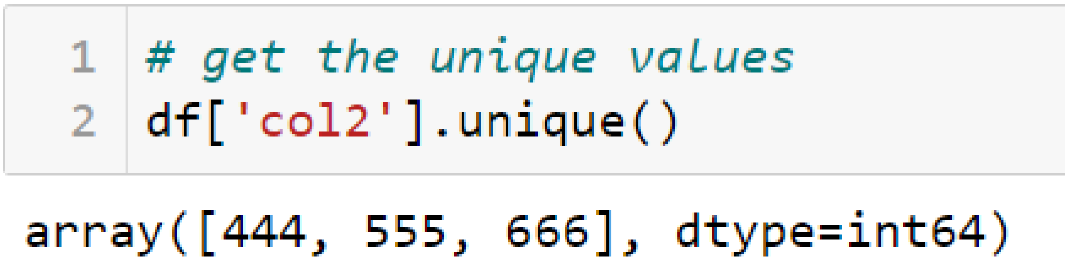
Get DataFrame Columns Names



```python
df.columns
```

```
Index(['col1', 'col2', 'col3'], dtype='object')
```

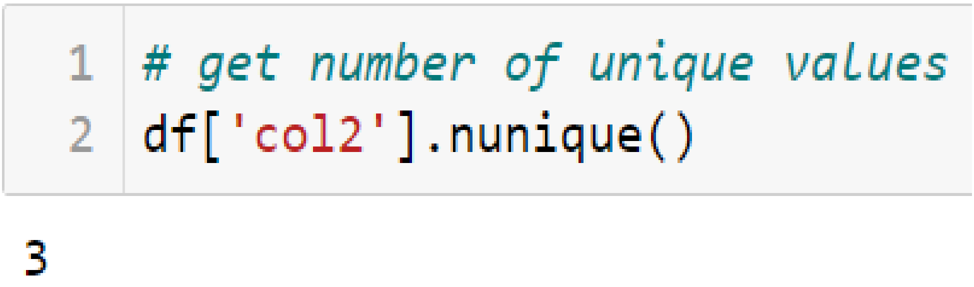
Get Unique Values



```python
# get the unique values
df['col2'].unique()
```

```
array([444, 555, 666], dtype=int64)
```

Get Unique Values Number

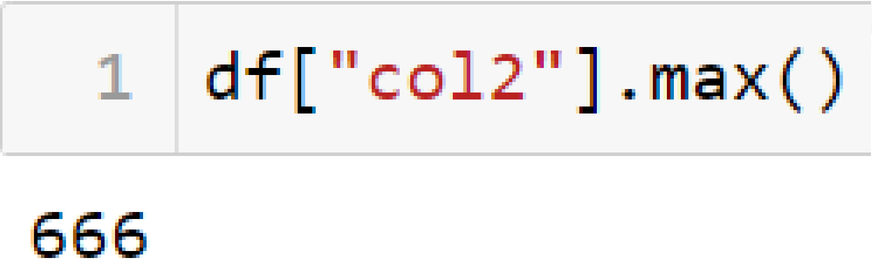


```python
# get number of unique values
df['col2'].nunique()
```

```
3
```

EDA using Pandas:

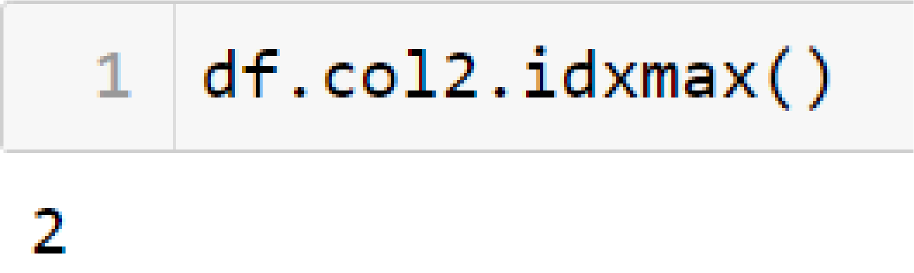
Get Max Value



```
df["col2"].max()
```

```
666
```

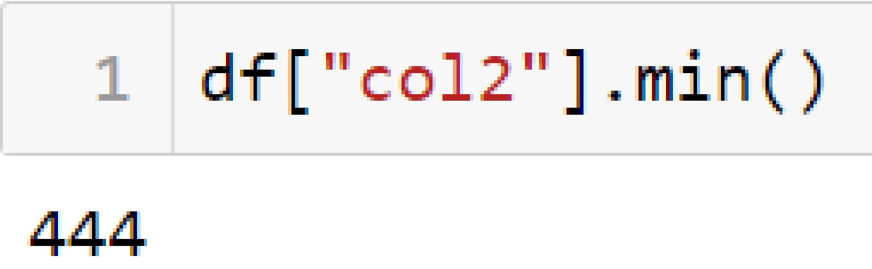
Get Element whose value is Max



```
df.col2.idxmax()
```

```
2
```

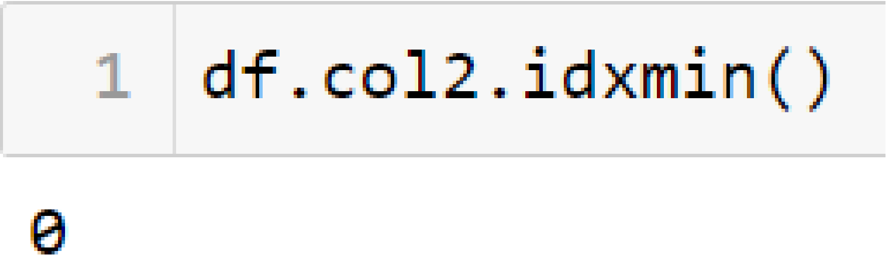
Get Min Value



```
df["col2"].min()
```

```
444
```

Get Element whose value is Min

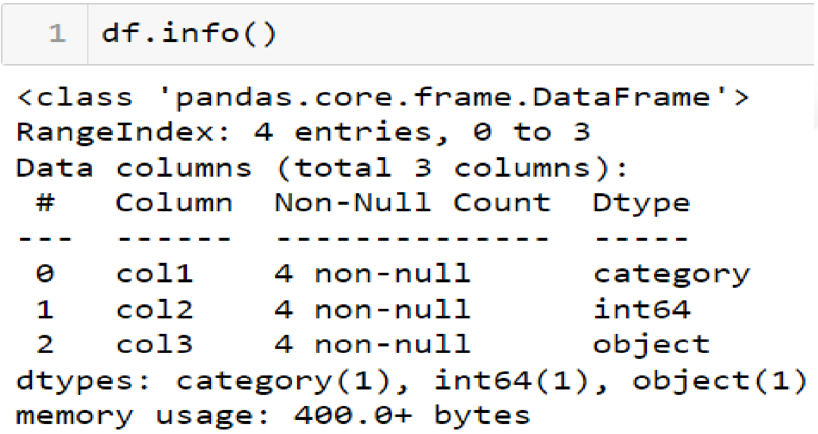


```
df.col2.idxmin()
```

```
0
```

EDA using Pandas:

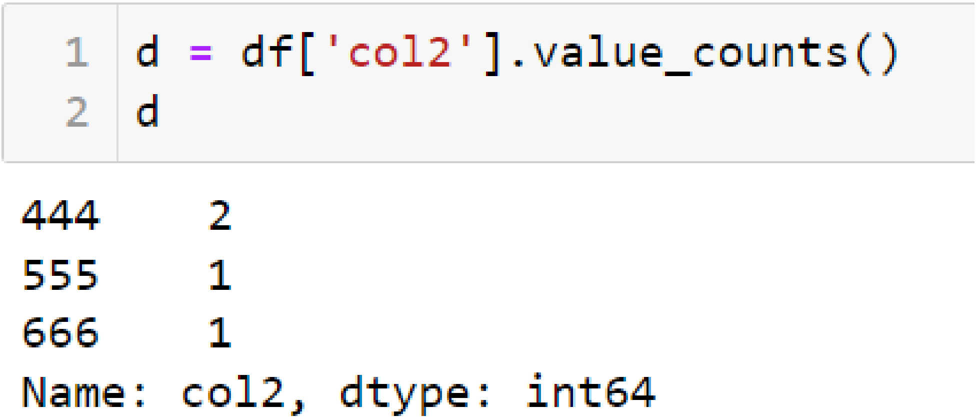
DataFrame Basic Information



```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype   
---  ------  --------------  -----   
 0   col1    4 non-null      category
 1   col2    4 non-null      int64   
 2   col3    4 non-null      object  
dtypes: category(1), int64(1), object(1)
memory usage: 400.0+ bytes
```

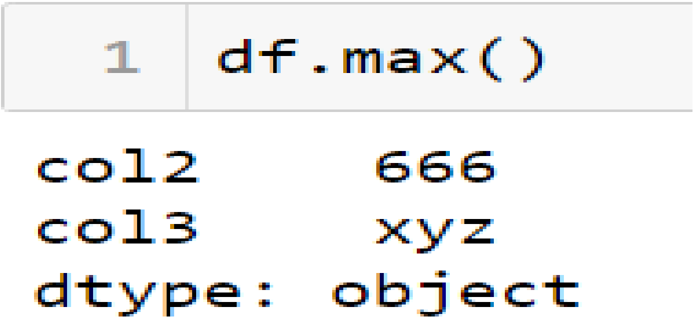
Unique Values Frequency



```python
d = df['col2'].value_counts()
d
```

```
444    2
555    1
666    1
Name: col2, dtype: int64
```

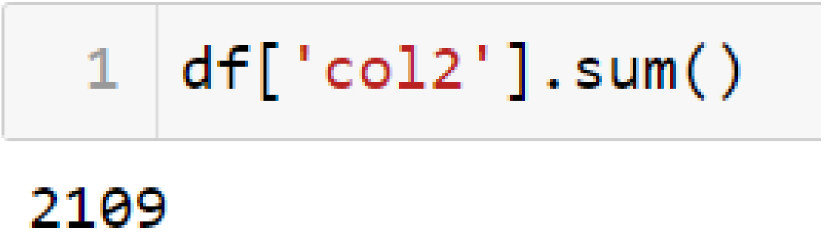
Get Max Value of All Columns



```python
df.max()
```

```
col2     666
col3     xyz
dtype: object
```

Summation



```python
df['col2'].sum()
```

```
2109
```

3. Data Manipulation

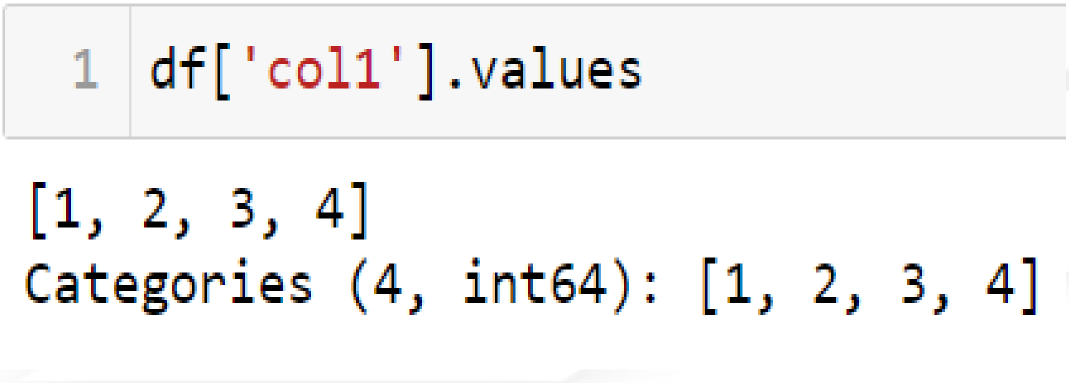
Data Manipulation:

- Pandas provides built-in methods and attributes that allows you to apply different operations over Pandas DataFrames or Series.

- Below are the most popular & Important attributes that allows you to apply data manipulation.

Data Manipulation (Pandas to Numpy):

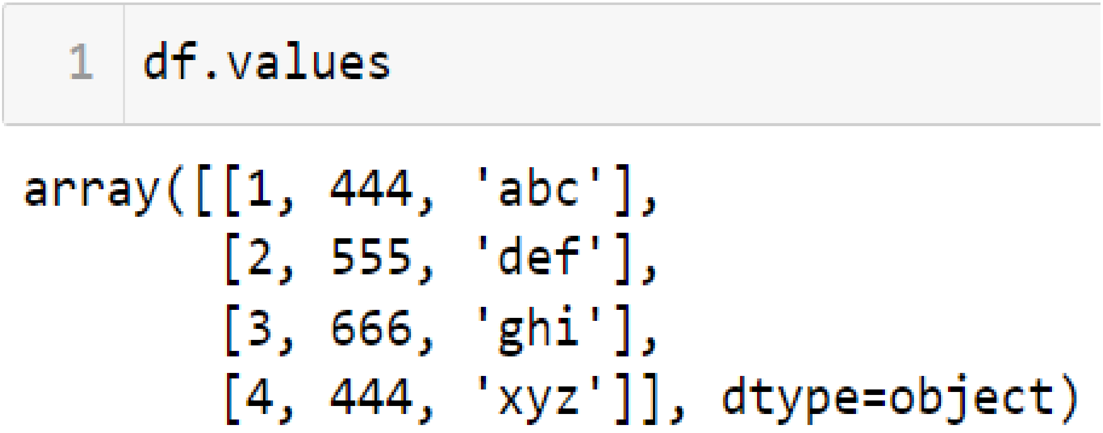
Convert Pandas Series into Numpy 1D-Array



```
df['col1'].values
```

```
[1, 2, 3, 4]
Categories (4, int64): [1, 2, 3, 4]
```

Convert Pandas DataFrame into Numpy 2D-Array

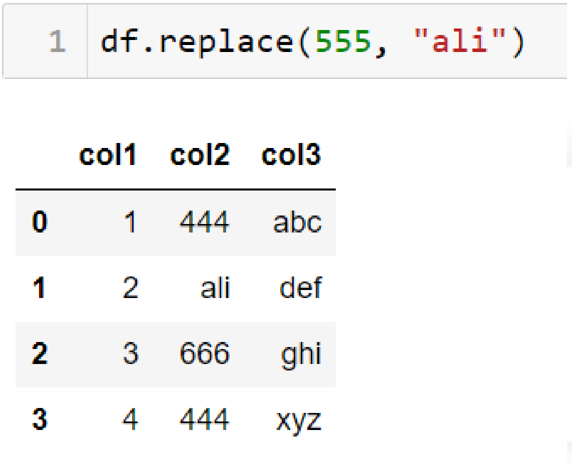


```
df.values
```

```
array([[1, 444, 'abc'],
       [2, 555, 'def'],
       [3, 666, 'ghi'],
       [4, 444, 'xyz']], dtype=object)
```

Data Manipulation (Replace Values):

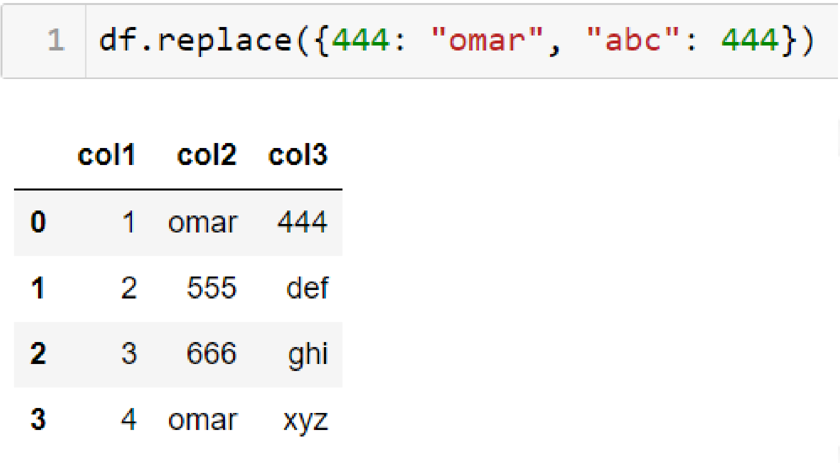
Replace a Single Value



```
df.replace(555, "ali")
```

	col1	col2	col3
0	1	444	abc
1	2	ali	def
2	3	666	ghi
3	4	444	xyz

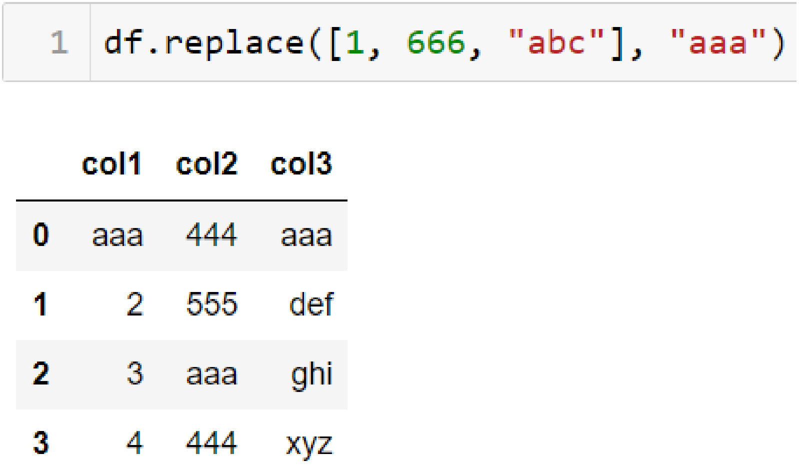
Replace Multiple Values using Dictionary



```
df.replace({444: "omar", "abc": 444})
```

	col1	col2	col3
0	1	omar	444
1	2	555	def
2	3	666	ghi
3	4	omar	xyz

Replace Multiple Values by One Value

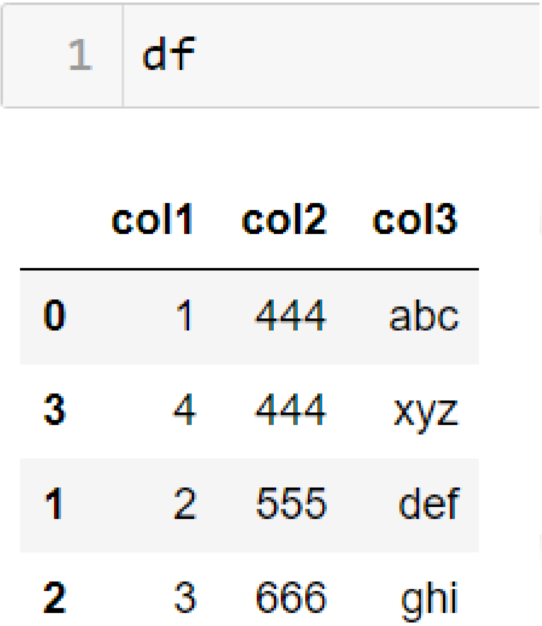


```
df.replace([1, 666, "abc"], "aaa")
```

	col1	col2	col3
0	aaa	444	aaa
1	2	555	def
2	3	aaa	ghi
3	4	444	xyz

Data Manipulation (Mapping):

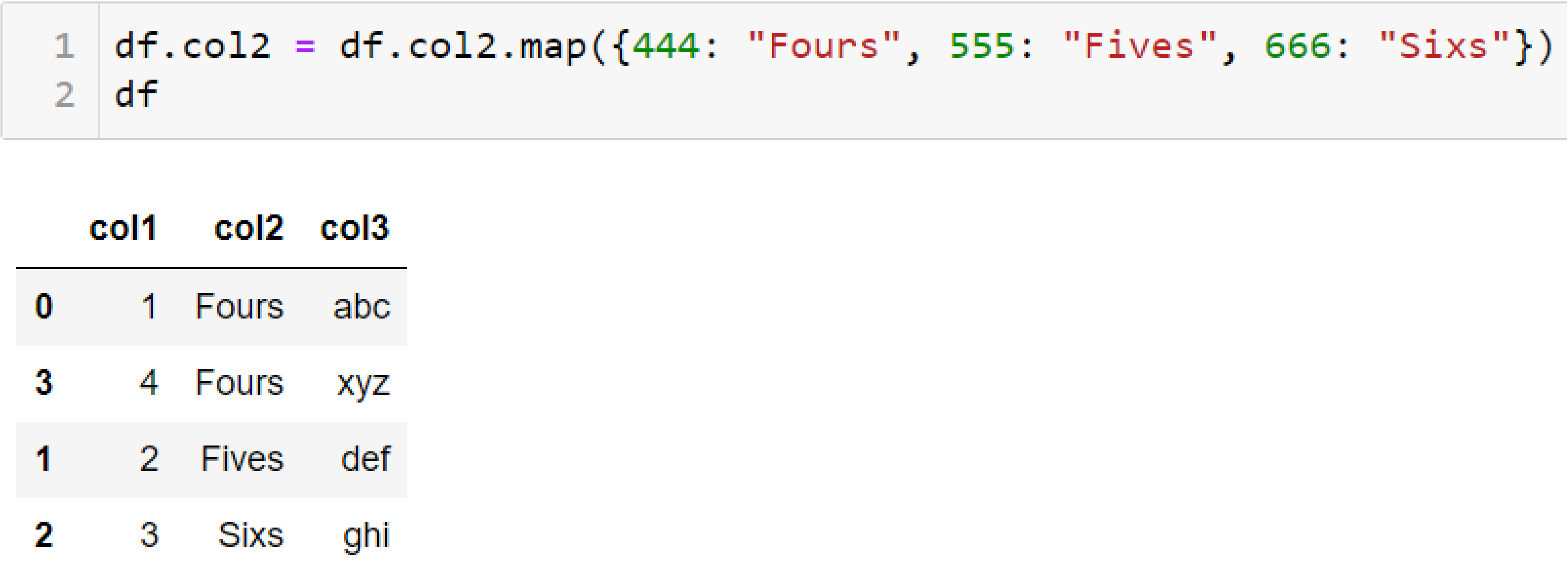
Before Mapping



```python
df
```

	col1	col2	col3
0	1	444	abc
3	4	444	xyz
1	2	555	def
2	3	666	ghi

After Mapping

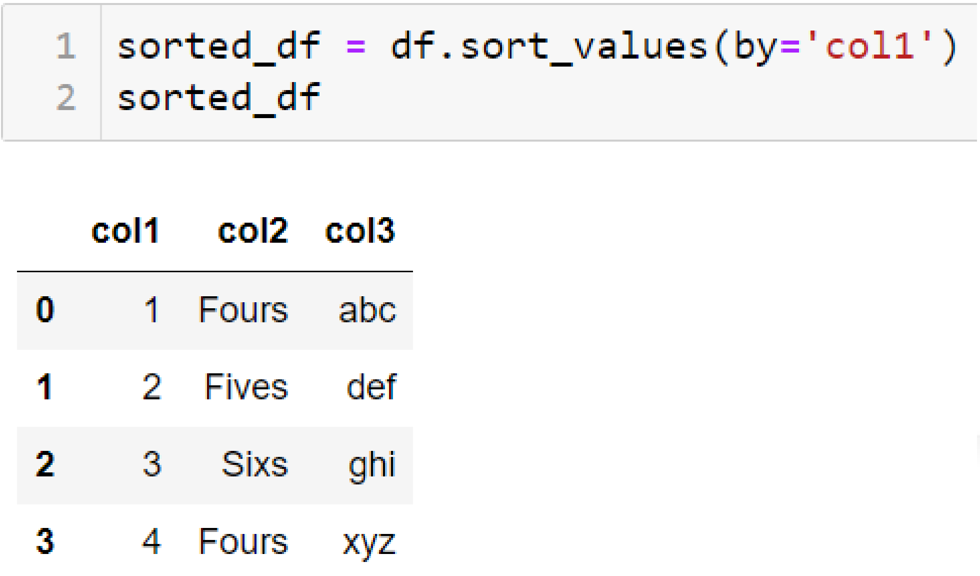


```python
df.col2 = df.col2.map({444: "Fours", 555: "Fives", 666: "Sixs"})
df
```

	col1	col2	col3
0	1	Fours	abc
3	4	Fours	xyz
1	2	Fives	def
2	3	Sixs	ghi

Data Manipulation (Sorting):

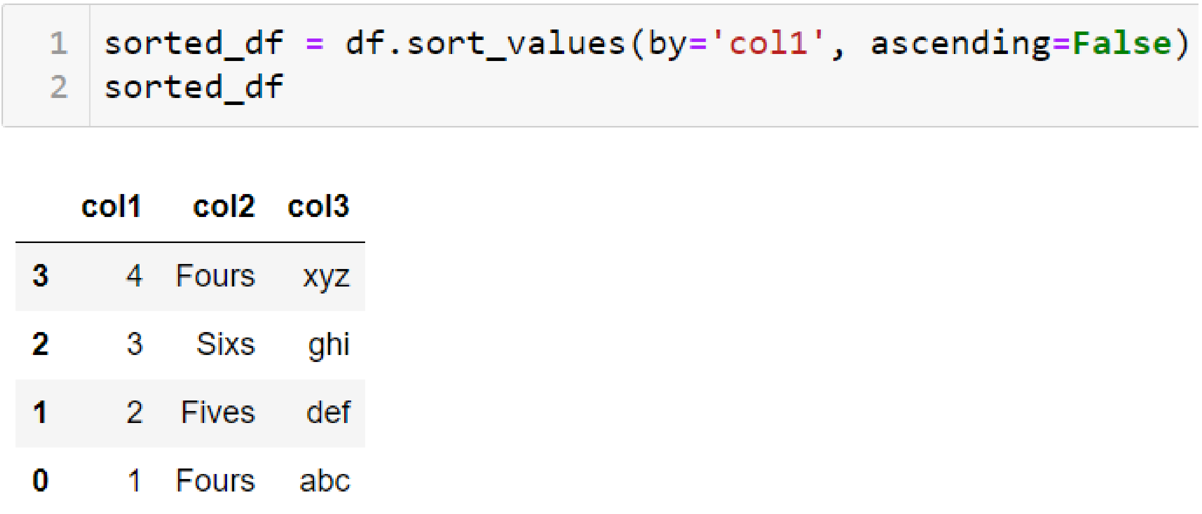
Ascending sorting



```python
sorted_df = df.sort_values(by='col1')
sorted_df
```

```
   col1   col2  col3
0     1  Fours   abc
1     2  Fives   def
2     3   Sixs   ghi
3     4  Fours   xyz
```

Descending sorting



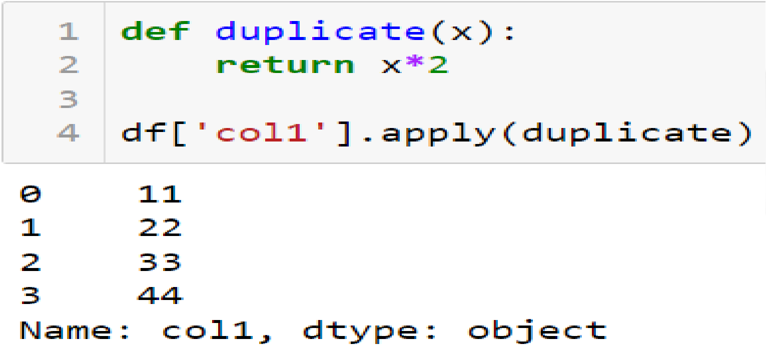
```python
sorted_df = df.sort_values(by='col1', ascending=False)
sorted_df
```

```
   col1   col2  col3
3     4  Fours   xyz
2     3   Sixs   ghi
1     2  Fives   def
0     1  Fours   abc
```

Data Manipulation (Apply method):

- Is a methods used to apply a certain function to each sample in the DataFrame.

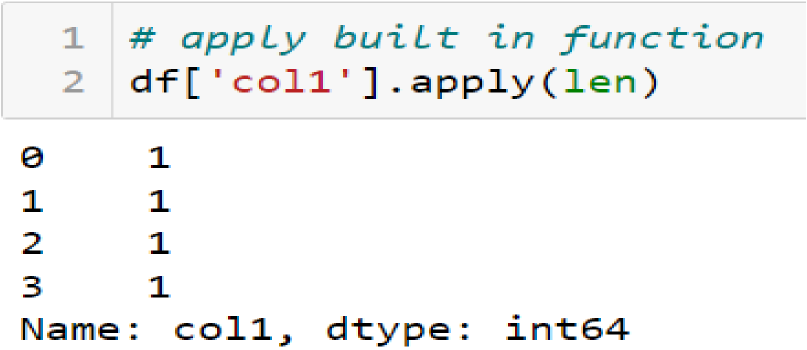
Example1



```python
def duplicate(x):
    return x*2

df['col1'].apply(duplicate)
```

```
0     11
1     22
2     33
3     44
Name: col1, dtype: object
```

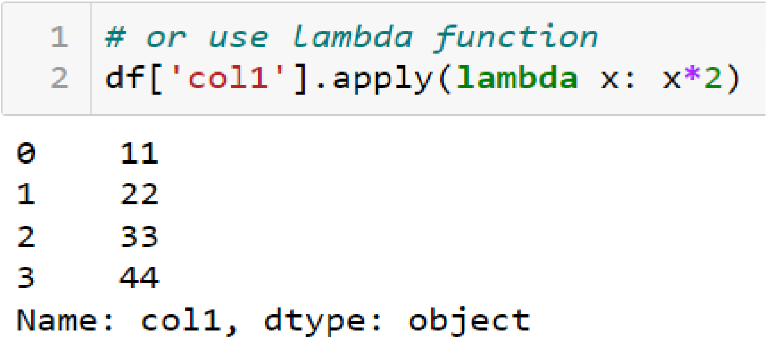
Example2



```python
# apply built in function
df['col1'].apply(len)
```

```
0     1
1     1
2     1
3     1
Name: col1, dtype: int64
```

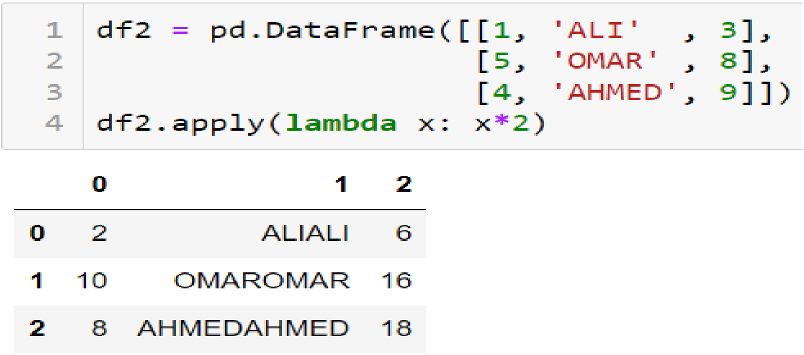
Example3



```python
# or use lambda function
df['col1'].apply(lambda x: x*2)
```

```
0     11
1     22
2     33
3     44
Name: col1, dtype: object
```

Example4



```python
df2 = pd.DataFrame([[1, 'ALI'  , 3],
                    [5, 'OMAR' , 8],
                    [4, 'AHMED', 9]])
df2.apply(lambda x: x*2)
```

```
    0             1   2
0   2        ALIALI   6
1  10      OMAROMAR  16
2   8    AHMEDAHMED  18
```

4. Indexing & Slicing

Indexing:

- Means accessing one element in Series or DataFrame, using its index or name.

- There are two ways to apply indexing:
  - Either using the element’s Index.
  - Or using the element’s Name.

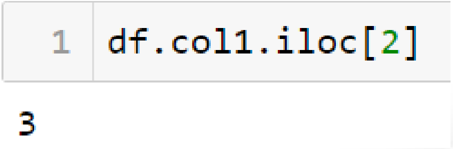
Slicing:

- Means accessing many elements in Series or DataFrame, by specifying a range of Indices or name.

- There are two ways to apply Slicing:
  - Either, using a range of elements’ Indices.
  - Or using a range of elements’ Names.

Indexing & Slicing using Index:

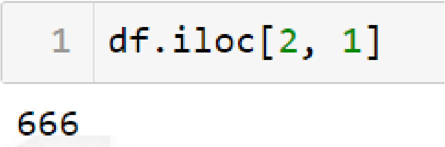
- **Series Indexing**



```
df.col1.iloc[2]
```

```
3
```

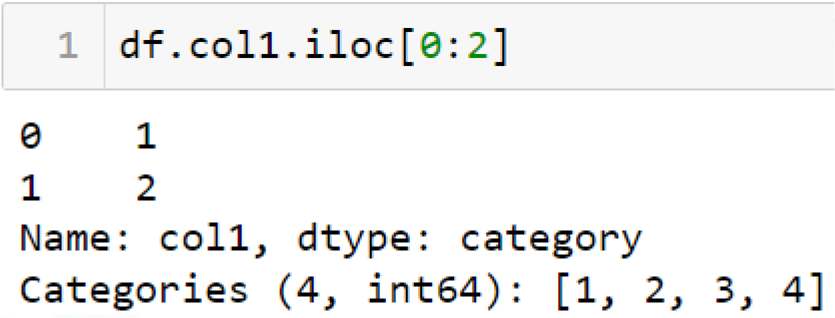
- **Matrix Indexing**



```
df.iloc[2, 1]
```

```
666
```

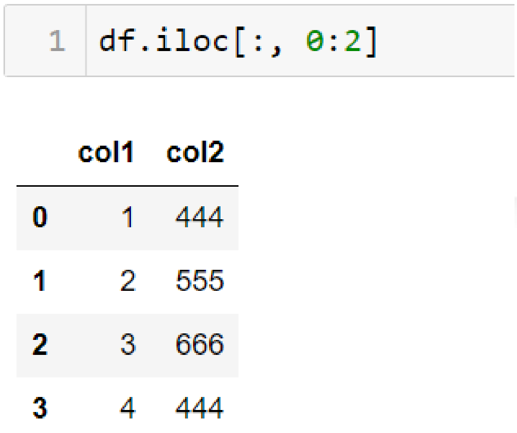
- **Series Slicing**



```
df.col1.iloc[0:2]
```

```
0    1
1    2
Name: col1, dtype: category
Categories (4, int64): [1, 2, 3, 4]
```

- **Matrix Slicing**

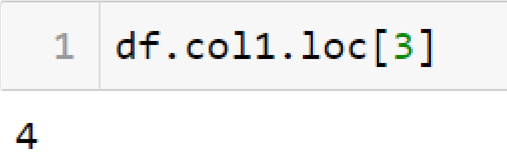


```
df.iloc[:, 0:2]
```

	col1	col2
0	1	444
1	2	555
2	3	666
3	4	444

Indexing & Slicing using Names:

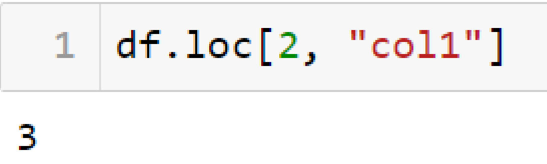
- **Series Indexing**



```
df.col1.loc[3]
```

```
4
```

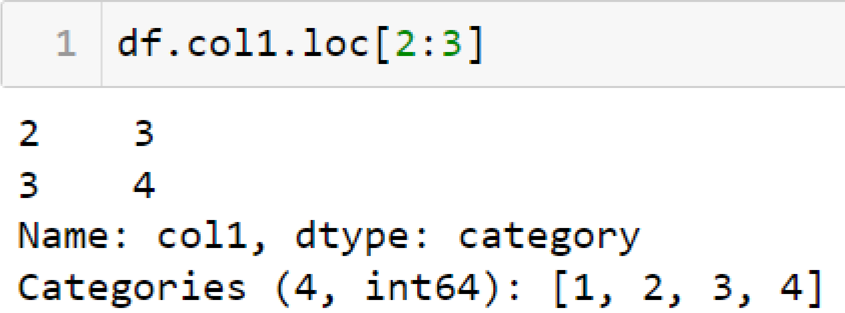
- **Matrix Indexing**



```
df.loc[2, "col1"]
```

```
3
```

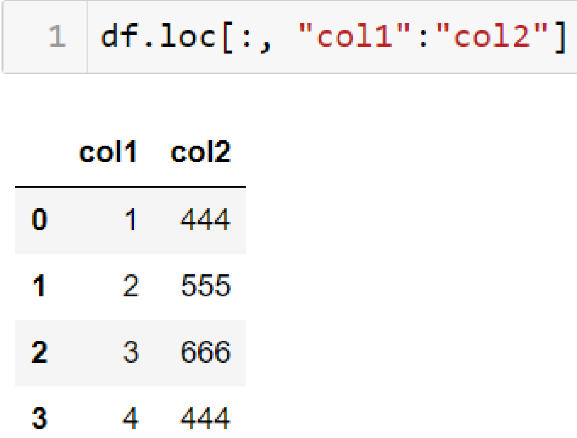
- **Series Slicing**



```
df.col1.loc[2:3]
```

```
2    3
3    4
Name: col1, dtype: category
Categories (4, int64): [1, 2, 3, 4]
```

- **Matrix Slicing**



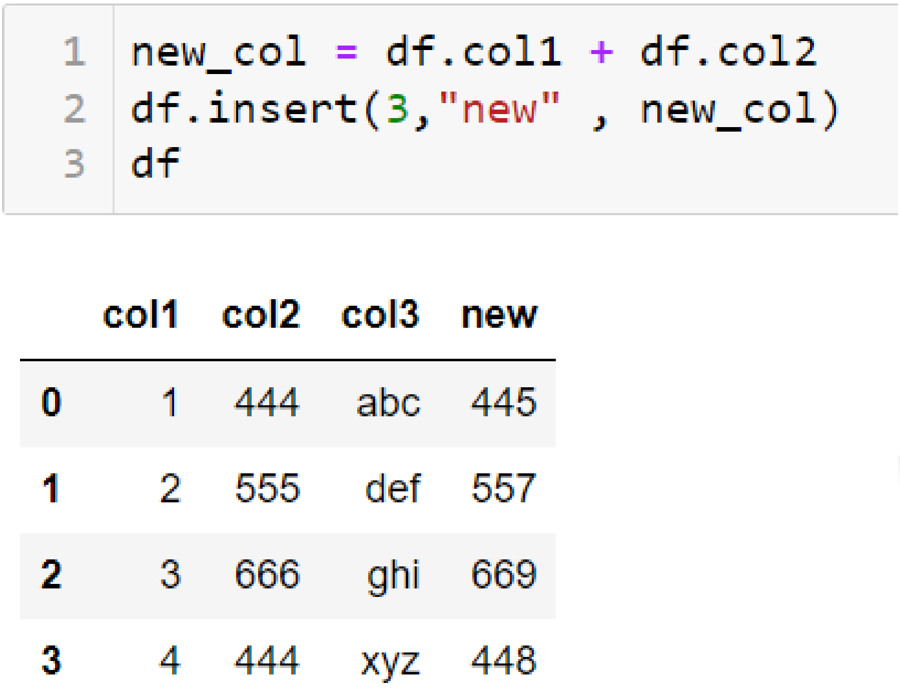
```
df.loc[:, "col1":"col2"]
```

	col1	col2
0	1	444
1	2	555
2	3	666
3	4	444

5. Inserting/Dropping DataFrame Columns & Rows

Insert New Columns:

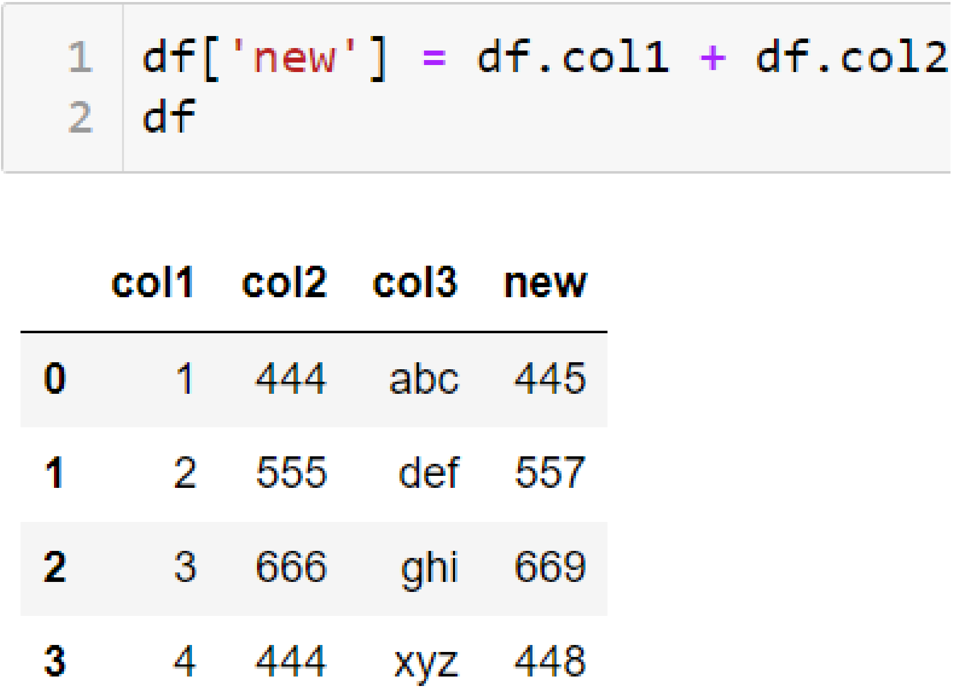
The first way



```
new_col = df.col1 + df.col2
df.insert(3,"new" , new_col)
df
```

	col1	col2	col3	new
0	1	444	abc	445
1	2	555	def	557
2	3	666	ghi	669
3	4	444	xyz	448

Another way

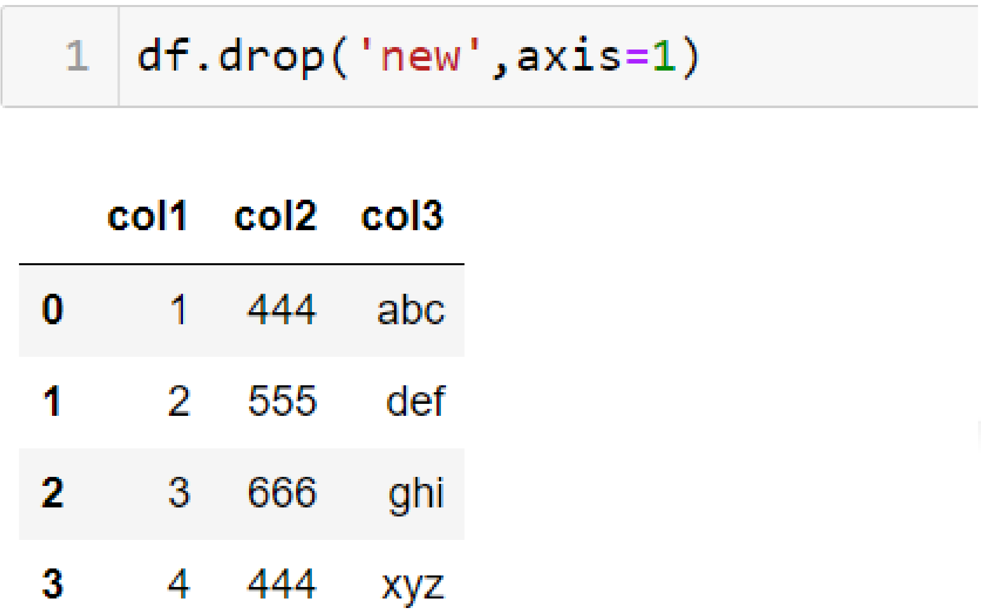


```
df['new'] = df.col1 + df.col2
df
```

	col1	col2	col3	new
0	1	444	abc	445
1	2	555	def	557
2	3	666	ghi	669
3	4	444	xyz	448

Drop Columns:

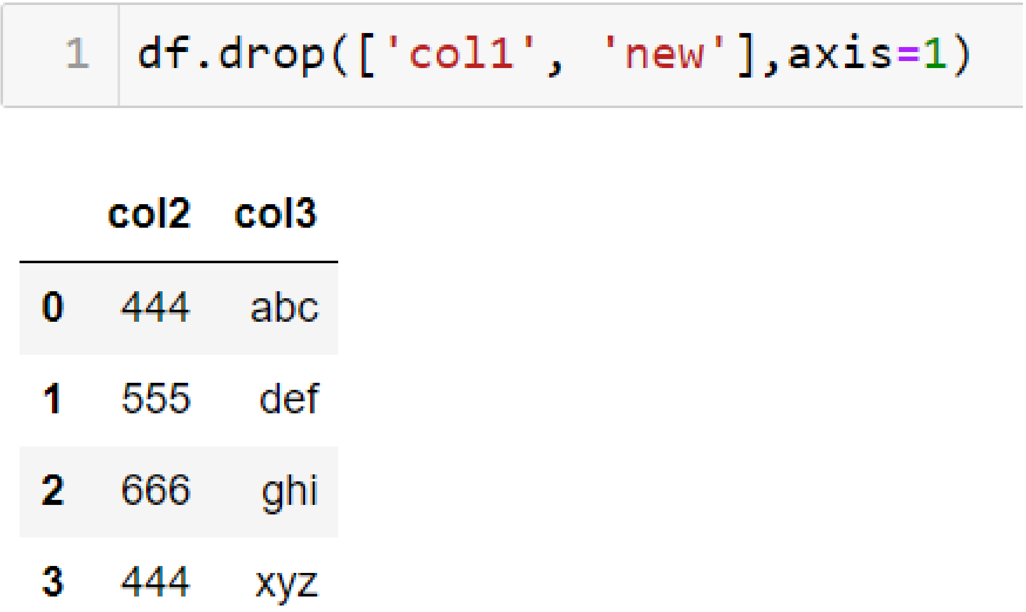
Drop one columns



```
df.drop('new',axis=1)
```

	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz

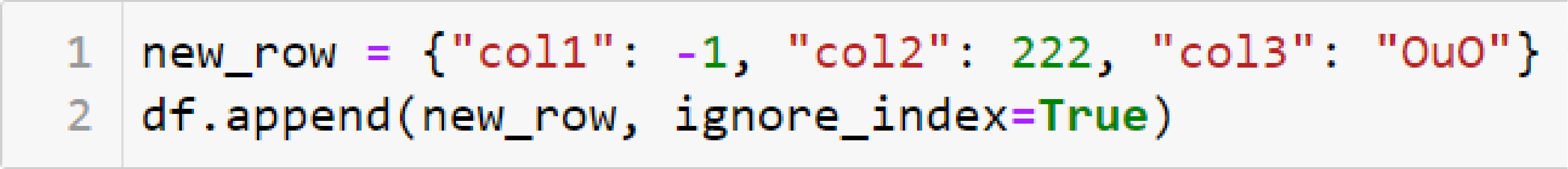
Drop many columns



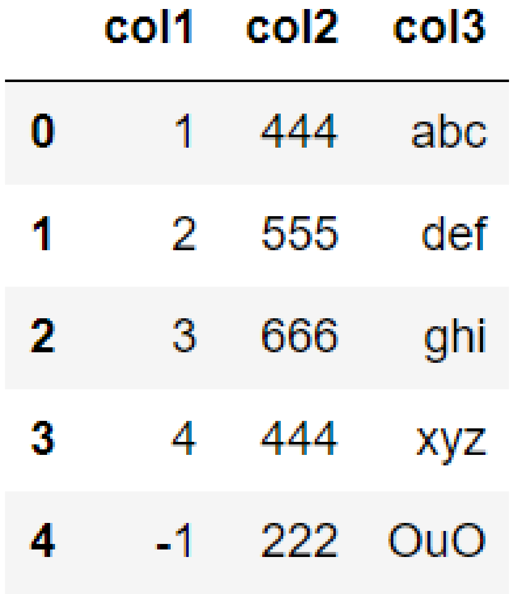
```
df.drop(['col1', 'new'],axis=1)
```

	col2	col3
0	444	abc
1	555	def
2	666	ghi
3	444	xyz

Insert New Rows:



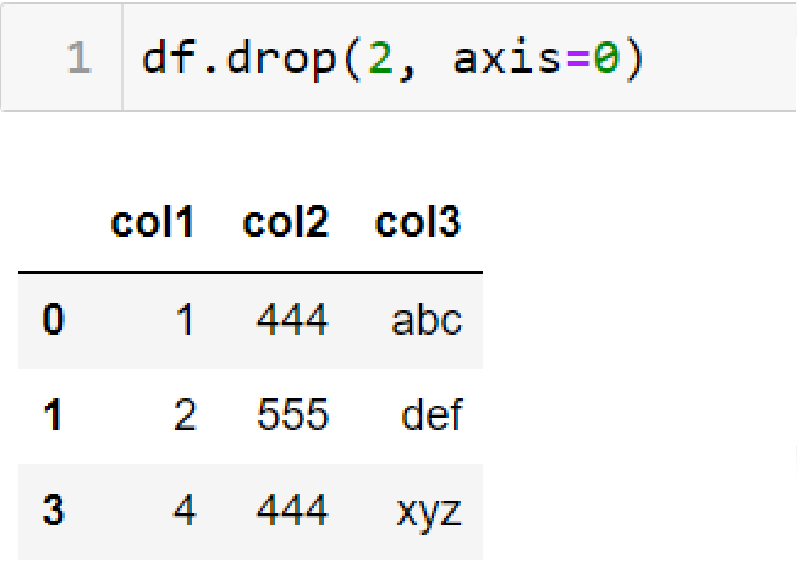
```
new_row = {"col1": -1, "col2": 222, "col3": "OuO"}
df.append(new_row, ignore_index=True)
```



	col1	col2	col3
0	1	444	abc
1	2	555	def
2	3	666	ghi
3	4	444	xyz
4	-1	222	OuO

Drop Rows:

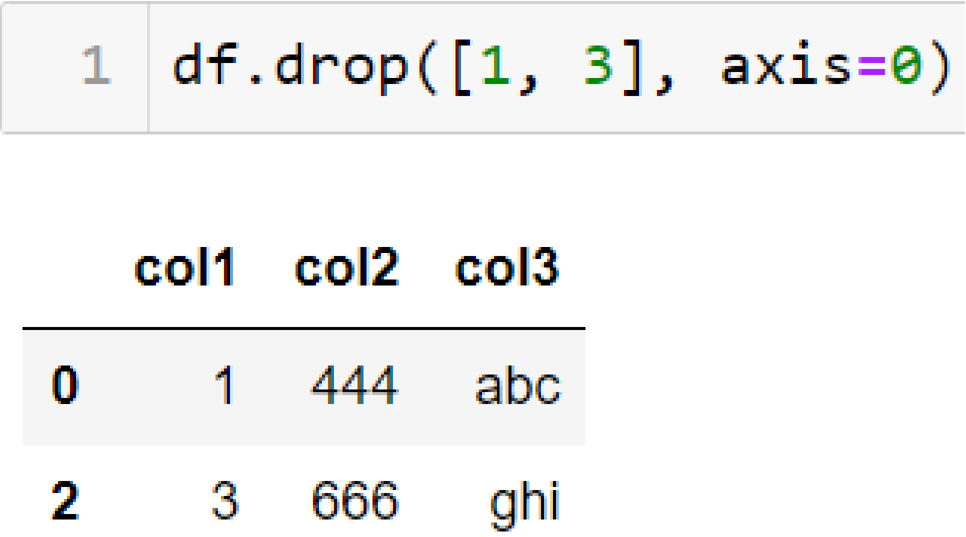
Drop one row



```python
df.drop(2, axis=0)
```

	col1	col2	col3
0	1	444	abc
1	2	555	def
3	4	444	xyz

Drop many rows



```python
df.drop([1, 3], axis=0)
```

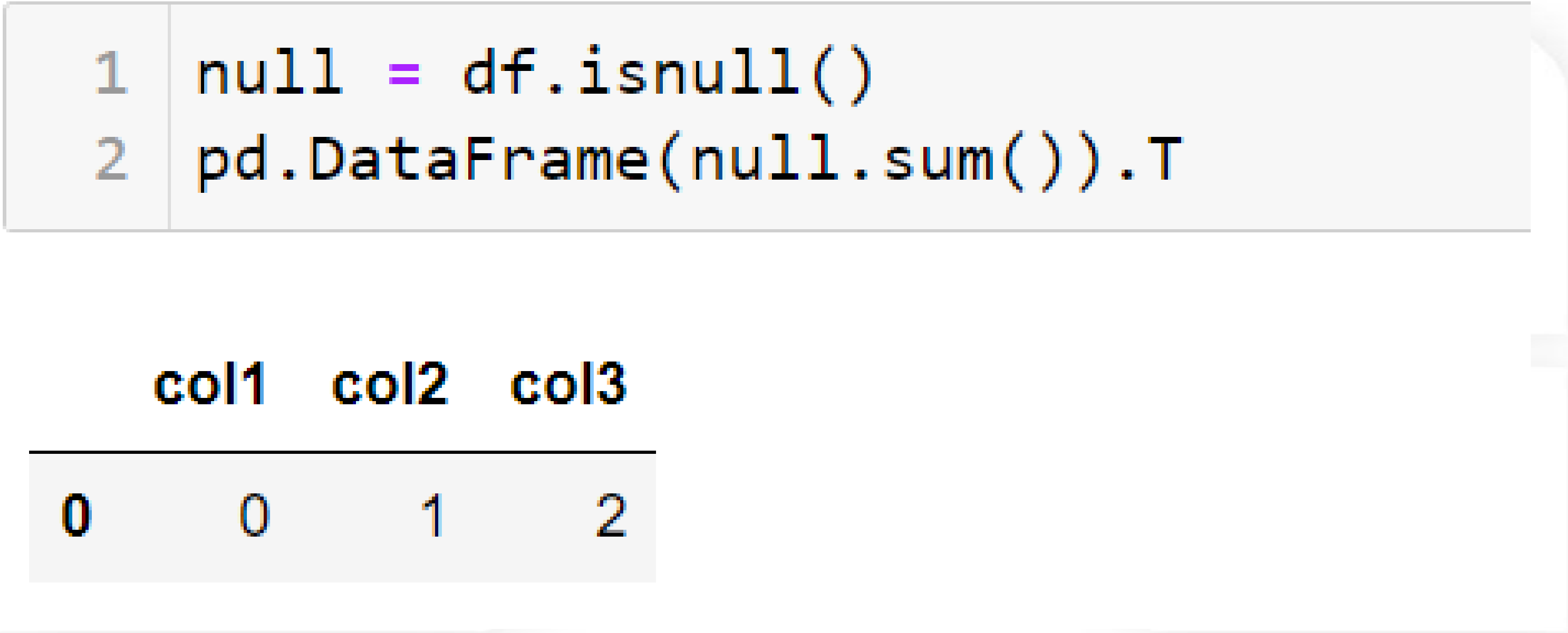
	col1	col2	col3
0	1	444	abc
2	3	666	ghi

6. Null Values

What are Null Values?

- Null Values means missing values, which means that an element doesn’t have a value, or have a value of None or Nan.
- Null Values occur due to problems during gathering data, for example a client forgot or to enter his age.

Check for Null Values:



```python
null = df.isnull()
pd.DataFrame(null.sum()).T
```

	col1	col2	col3
0	0	1	2

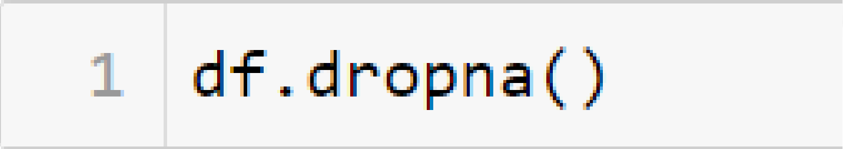
Handle Null Values:

- There are three options to do to handle missing values:

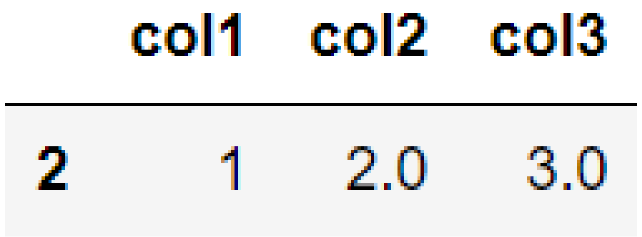
1. Drop rows that contain Null Values.
2. Drop columns that contain Null Values.
3. Replace Null Values with Mean, Median, or Mode.

Handle Null Values (Drop Rows):

Drop all rows

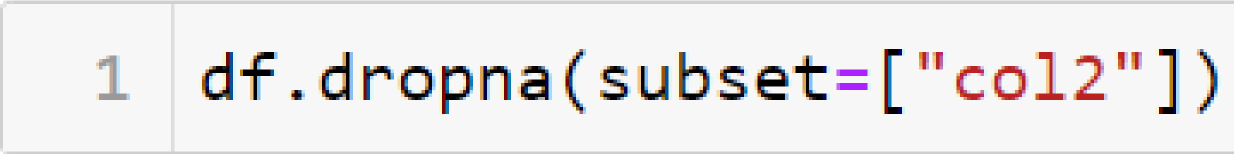


```python
df.dropna()
```

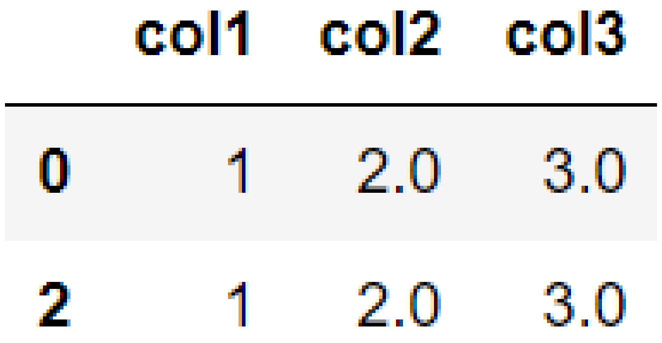


	col1	col2	col3
2	1	2.0	3.0

Drop rows in specific columns



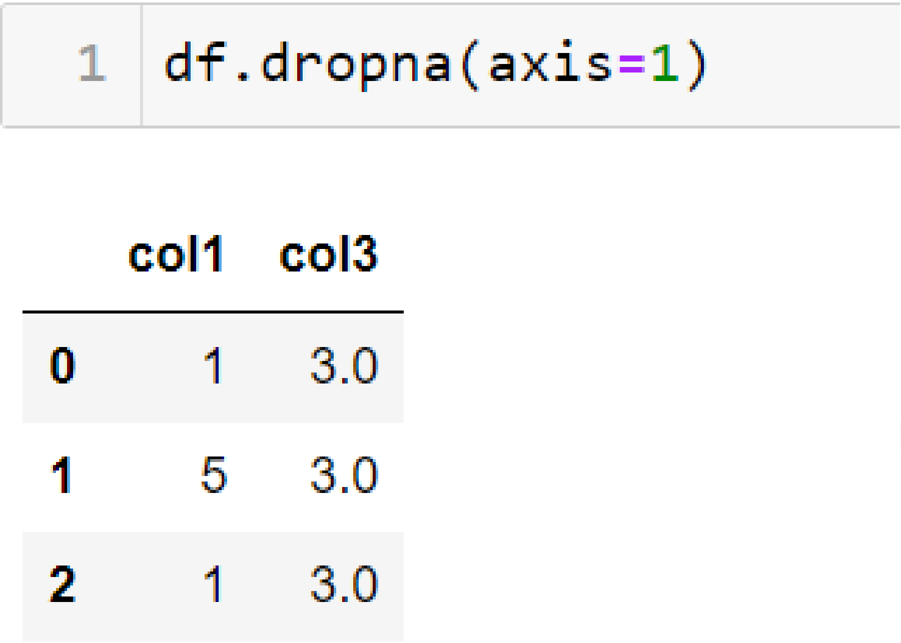
```python
df.dropna(subset=["col2"])
```



	col1	col2	col3
0	1	2.0	3.0
2	1	2.0	3.0

Handle Null Values (Drop Columns):

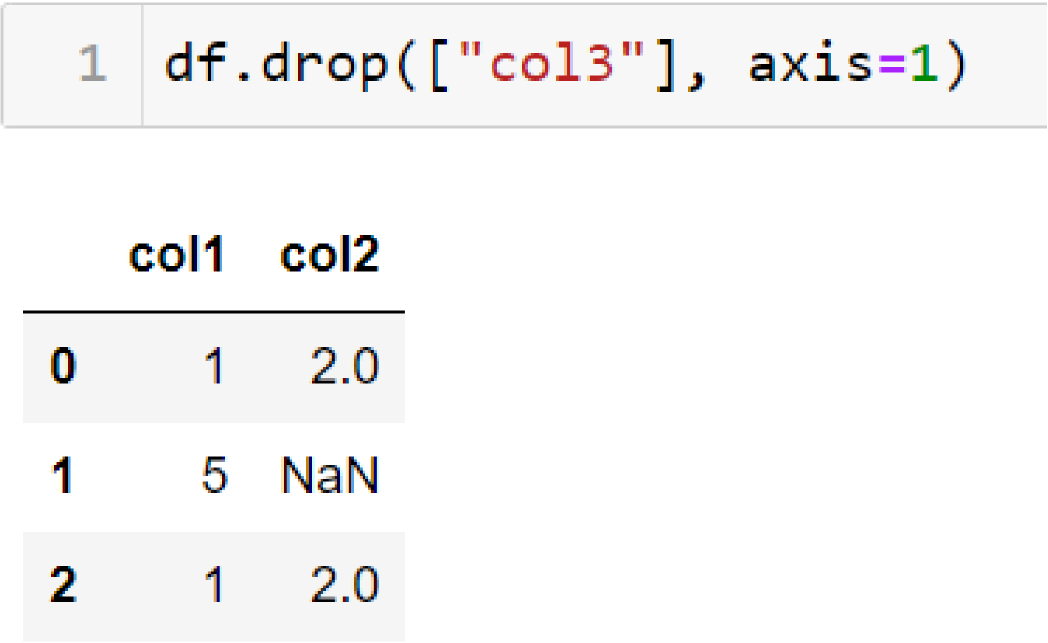
Drop all columns



```python
df.dropna(axis=1)
```

	col1	col3
0	1	3.0
1	5	3.0
2	1	3.0

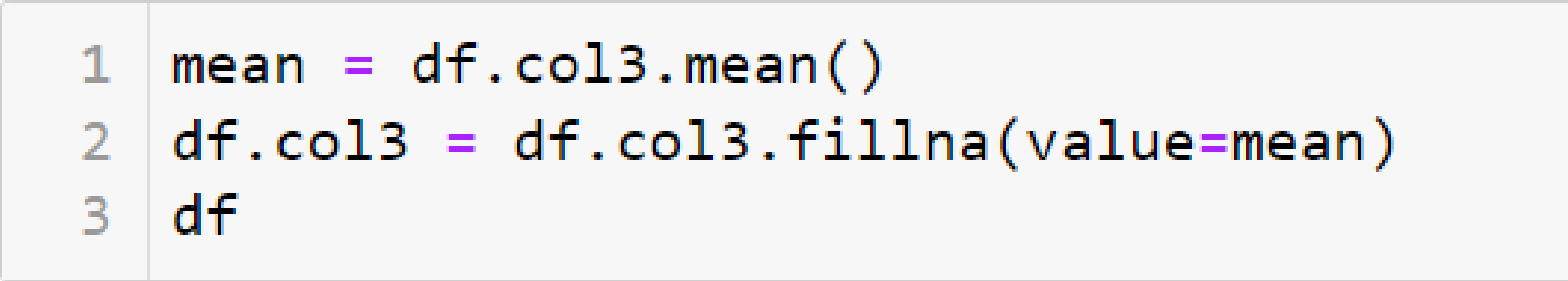
Drop Specific Columns



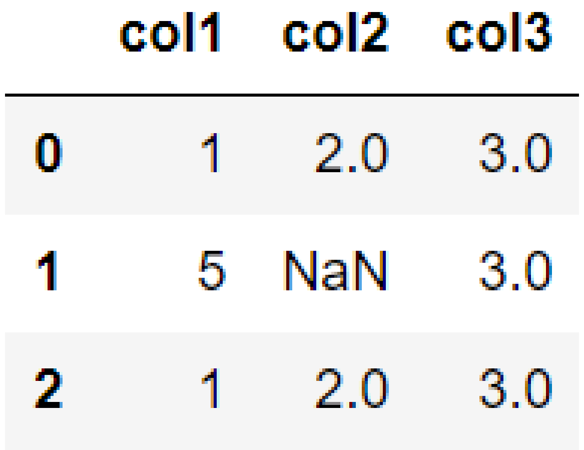
```python
df.drop(["col3"], axis=1)
```

	col1	col2
0	1	2.0
1	5	NaN
2	1	2.0

Handle Null Values (Replace Rows Null Values):



```
mean = df.col3.mean()
df.col3 = df.col3.fillna(value=mean)
df
```



	col1	col2	col3
0	1	2.0	3.0
1	5	NaN	3.0
2	1	2.0	3.0

Thank You