Introduction To Data Analysis Using Pandas on Watson Studio

Kunal Malhotra

kunal.malhotra1@ibm.com

@kunal_forever

IBM

CODE

Agenda

- What is Data Analysis ?
- Introduction to Pandas.
- Advantages of Pandas.
- Introduction to Watson Studio.
- How to perform data analysis with pandas.

Data Analysis

- What is it?
  - Apply logical techniques to describe, condense, recap and evaluate Data and illustrate information.

- Goal of Data Analysis:
  - Discover useful information.
  - Provide insights.
  - Suggest conclusions.
  - Support decision Making.

Introduction To Pandas.

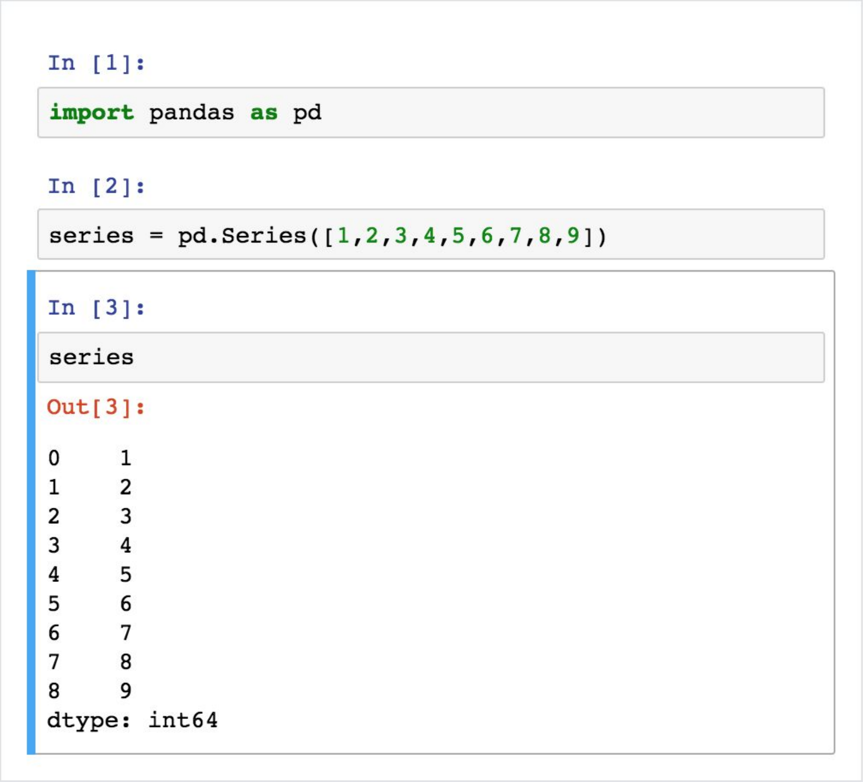
- **Pandas** is a python package for **data analysis.**
- It Provides built-in data structures which simplify the manipulation and analysis of data set.
- Pandas is easy to use and powerful, but “with great power comes great responsibility”.
- http://pandas.pydata.org/pandas-docs/stable/

Pandas: Essential Concepts

- **Series**
- **DataFrame**
- **Reading and Writing Files**
- **Aggregating and Grouping in Pandas**
- **Time Series analysis using pandas**
- **Visualization in pandas**

Introduction To Series In Pandas

Series in pandas is an object which is similar to python built-in list data structure, but differs from it because it has associated label with each **element** or better know as **index**.



In [1]:

```
import pandas as pd
```

In [2]:

```
series = pd.Series([1,2,3,4,5,6,7,8,9])
```

In [3]:

```
series
```

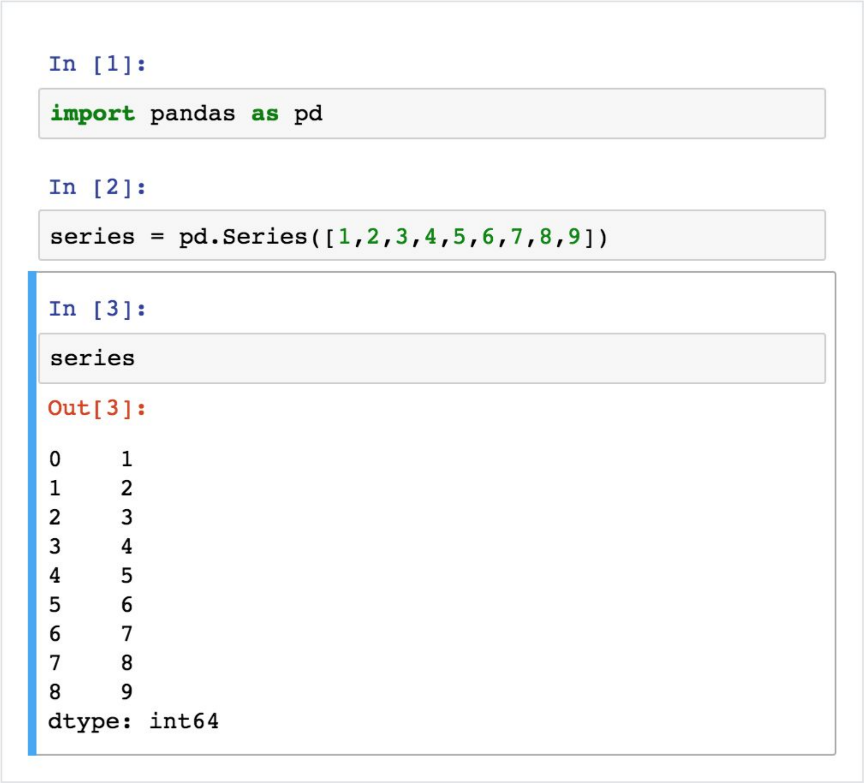
Out[3]:

```
0     1
1     2
2     3
3     4
4     5
5     6
6     7
7     8
8     9
dtype: int64
```

Understanding Series

- **Index** is leftward and **Values** are to the right. (Note:- If **Index** is not provided explicitly, then pandas creates **RangeIndex** starting from **0 to N - 1**, where N is the total number of elements.)

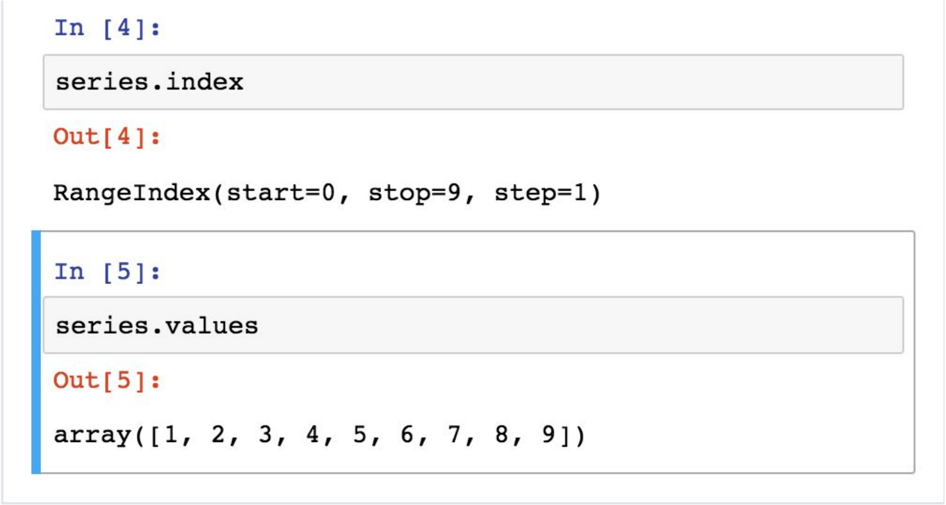
- Each series object has a data type (**dtype**), in the example on the right data type is **int64.**



```
In [1]:
import pandas as pd

In [2]:
series = pd.Series([1,2,3,4,5,6,7,8,9])

In [3]:
series
Out[3]:

0     1
1     2
2     3
3     4
4     5
5     6
6     7
7     8
8     9
dtype: int64
```

Operation On Pandas Series

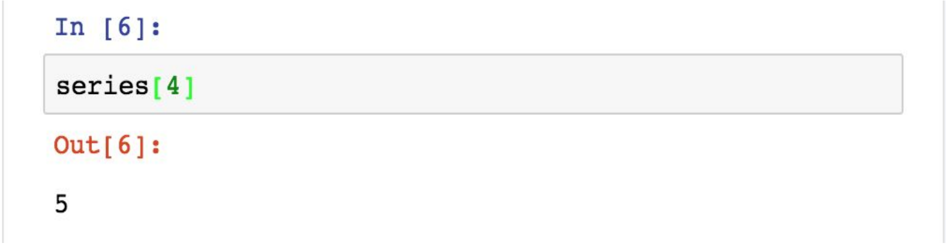
- Pandas series have attributes to extract it’s **values** and **labels.**



```
In [4]:
series.index
Out[4]:
RangeIndex(start=0, stop=9, step=1)
```

```
In [5]:
series.values
Out[5]:
array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

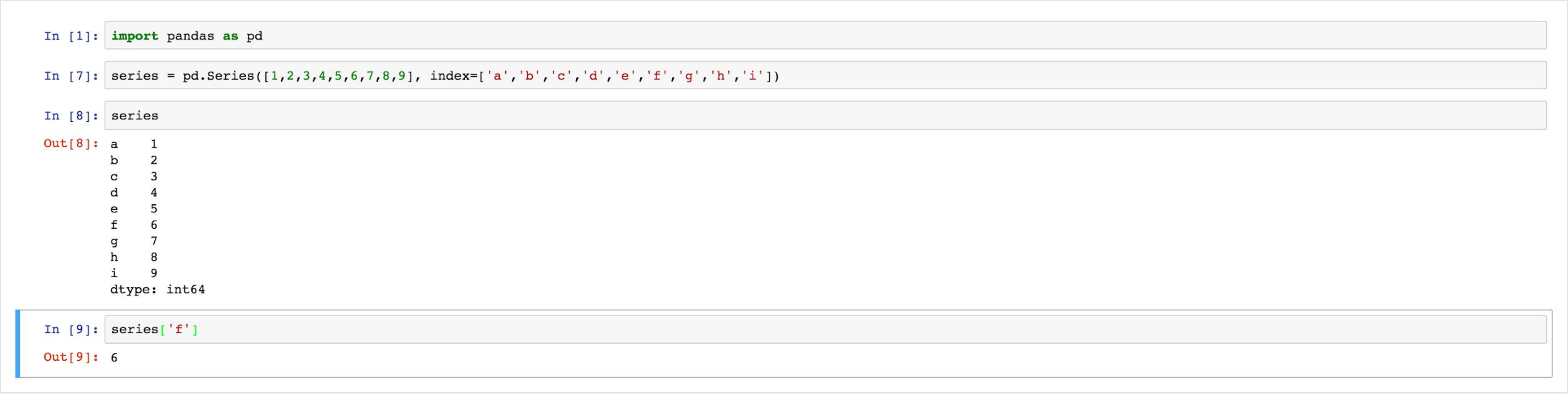
- Elements can be retrieved by their **labels(index)**.



```
In [6]:
series[4]
Out[6]:
5
```

Operation On Pandas Series Continued ..

- **Labels (index)** can be provided explicitly.
- Elements can be retrieved by the provided **labels(index)**.

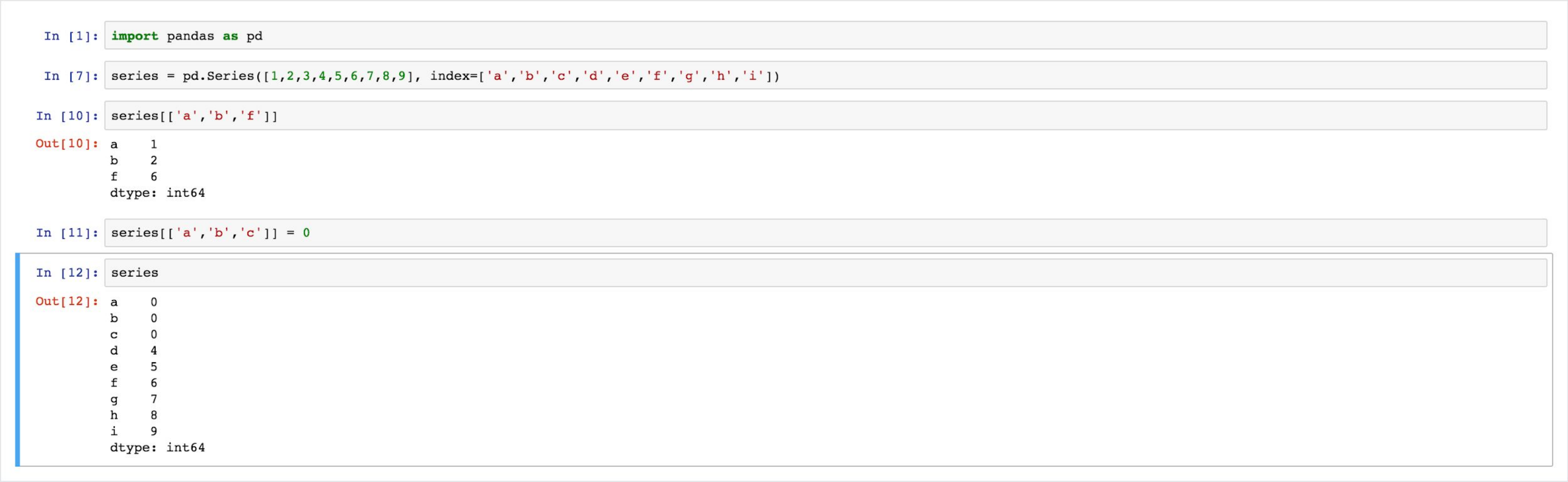


```
In [1]: import pandas as pd

In [7]: series = pd.Series([1,2,3,4,5,6,7,8,9], index=['a','b','c','d','e','f','g','h','i'])

In [8]: series

Out[8]: a    1
        b    2
        c    3
        d    4
        e    5
        f    6
        g    7
        h    8
        i    9
        dtype: int64

In [9]: series['f']

Out[9]: 6
```

IBM
CODE

Operation On Pandas Series Continued ..

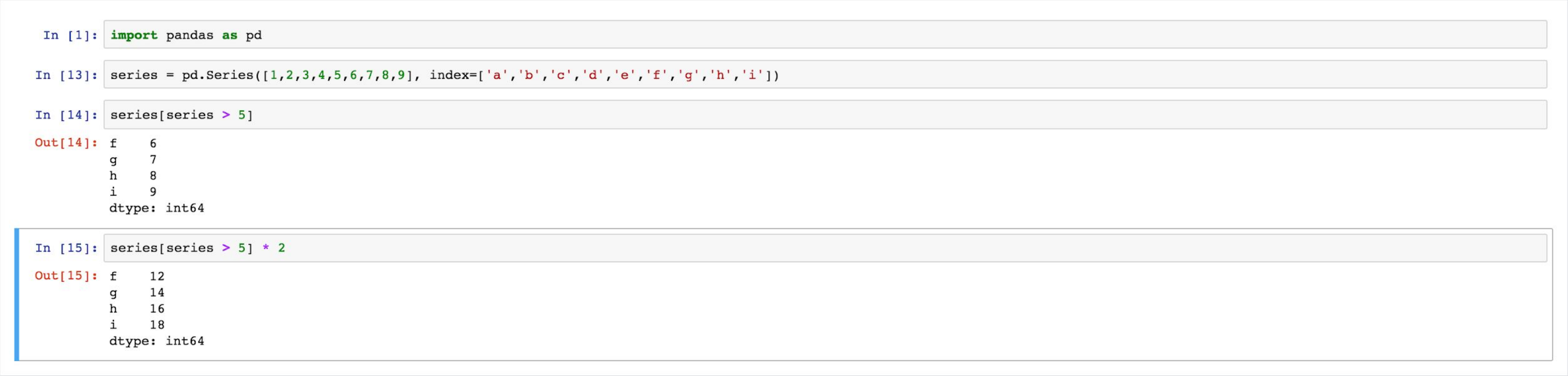
- It is easy to retrieve several elements by their indexes or make group assignment.



```
In [1]: import pandas as pd

In [7]: series = pd.Series([1,2,3,4,5,6,7,8,9], index=['a','b','c','d','e','f','g','h','i'])

In [10]: series[['a','b','f']]

Out[10]: a     1
         b     2
         f     6
         dtype: int64

In [11]: series[['a','b','c']] = 0

In [12]: series

Out[12]: a     0
         b     0
         c     0
         d     4
         e     5
         f     6
         g     7
         h     8
         i     9
         dtype: int64
```

Operation On Pandas Series Continued ..

- Filtering and Maths operations are easy as well.

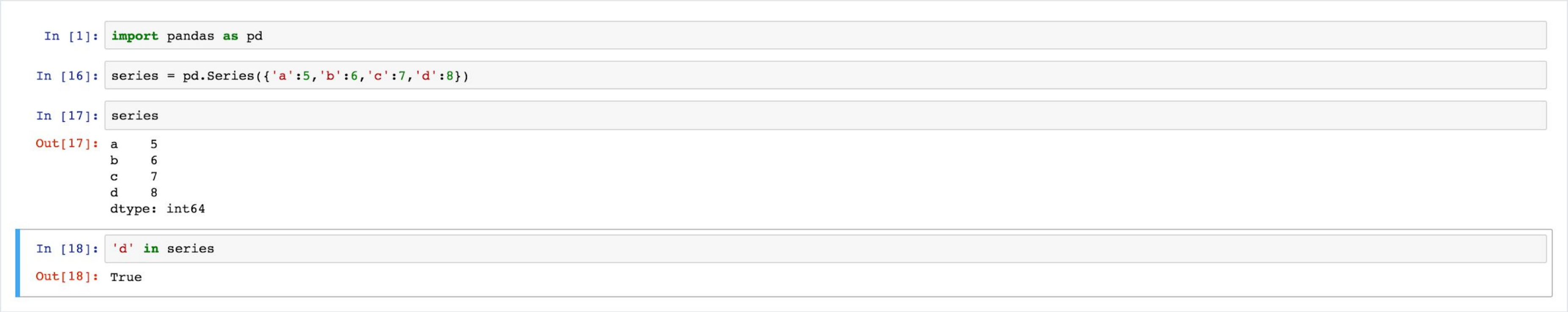


```
In [1]: import pandas as pd

In [13]: series = pd.Series([1,2,3,4,5,6,7,8,9], index=['a','b','c','d','e','f','g','h','i'])

In [14]: series[series > 5]

Out[14]: f    6
         g    7
         h    8
         i    9
         dtype: int64

In [15]: series[series > 5] * 2

Out[15]: f    12
         g    14
         h    16
         i    18
         dtype: int64
```

IBM CODE

Operation On Pandas Series Continued ..

- Series is very similar to dictionary, where key is an index and value is an element. Hence, we have generated a pandas series with python dictionary.



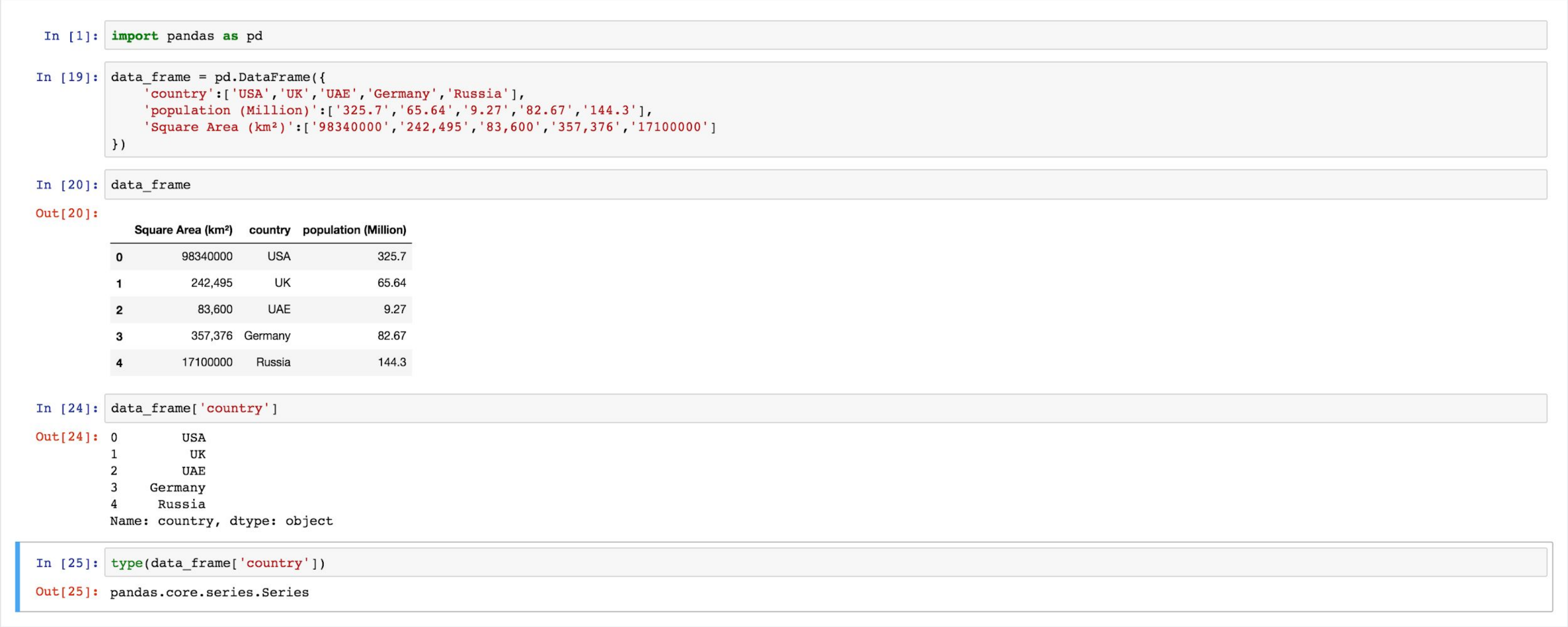
```
In [1]:  import pandas as pd

In [16]: series = pd.Series({'a':5,'b':6,'c':7,'d':8})

In [17]: series

Out[17]: a    5
         b    6
         c    7
         d    8
         dtype: int64

In [18]: 'd' in series

Out[18]: True
```

Introduction To DataFrame in Pandas

- DataFrame in pandas is a two-dimensional data structure, i.e., data is aligned in a tabular fashion. It has rows and columns. Each column in a DataFrame is a series object, rows consist of elements in Series.

- DataFrame can be constructed using built-in Python dictionary.

DataFrame in Pandas



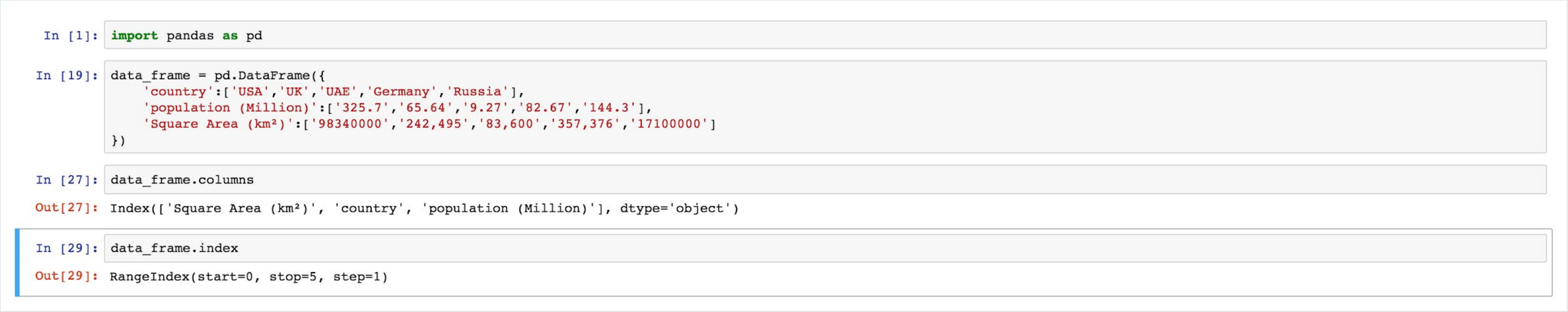
```
In [1]: import pandas as pd

In [19]: data_frame = pd.DataFrame({
             'country':['USA','UK','UAE','Germany','Russia'],
             'population (Million)':['325.7','65.64','9.27','82.67','144.3'],
             'Square Area (km²)':['98340000','242,495','83,600','357,376','17100000']
         })

In [20]: data_frame
Out[20]:
```

	Square Area (km²)	country	population (Million)
0	98340000	USA	325.7
1	242,495	UK	65.64
2	83,600	UAE	9.27
3	357,376	Germany	82.67
4	17100000	Russia	144.3

```
In [24]: data_frame['country']
Out[24]: 0        USA
         1         UK
         2        UAE
         3    Germany
         4     Russia
         Name: country, dtype: object

In [25]: type(data_frame['country'])
Out[25]: pandas.core.series.Series
```

Understanding DataFrame

- **DataFrame** object has **2 indexes**: column index and row index.

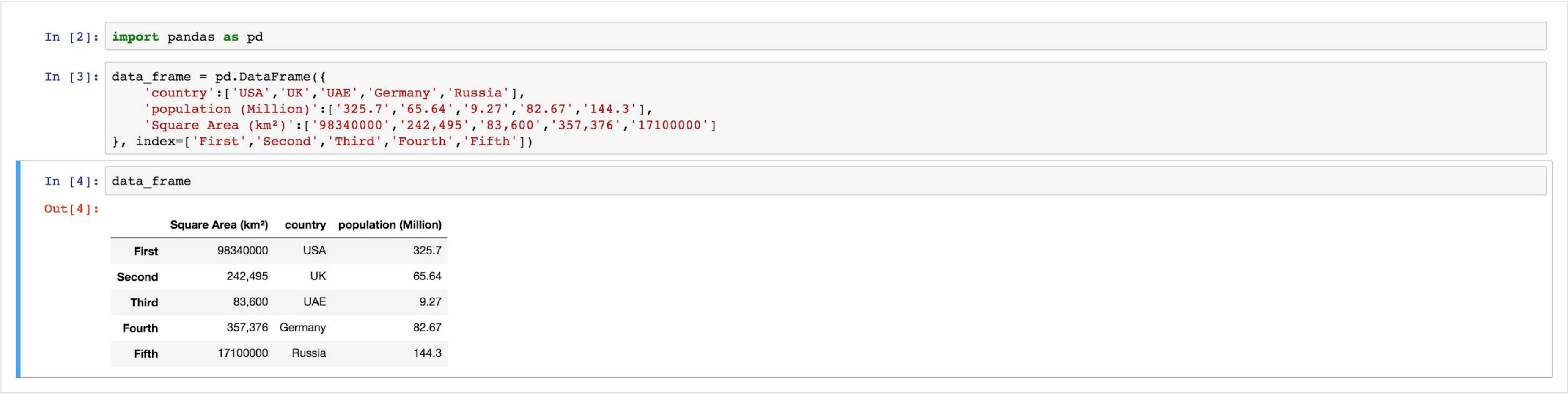


```
In [1]: import pandas as pd

In [19]: data_frame = pd.DataFrame({
             'country':['USA','UK','UAE','Germany','Russia'],
             'population (Million)':['325.7','65.64','9.27','82.67','144.3'],
             'Square Area (km²)':['98340000','242,495','83,600','357,376','17100000']
         })

In [27]: data_frame.columns

Out[27]: Index(['Square Area (km²)', 'country', 'population (Million)'], dtype='object')

In [29]: data_frame.index

Out[29]: RangeIndex(start=0, stop=5, step=1)
```

In the above image the DataFrame has 5 elements from 0 to 4.

Operation On Pandas DataFrame

- There are numerous ways to provide row index explicitly, for example you can provide index when creating a DataFrame or do it “on the fly” during runtime.



```
In [2]: import pandas as pd
```

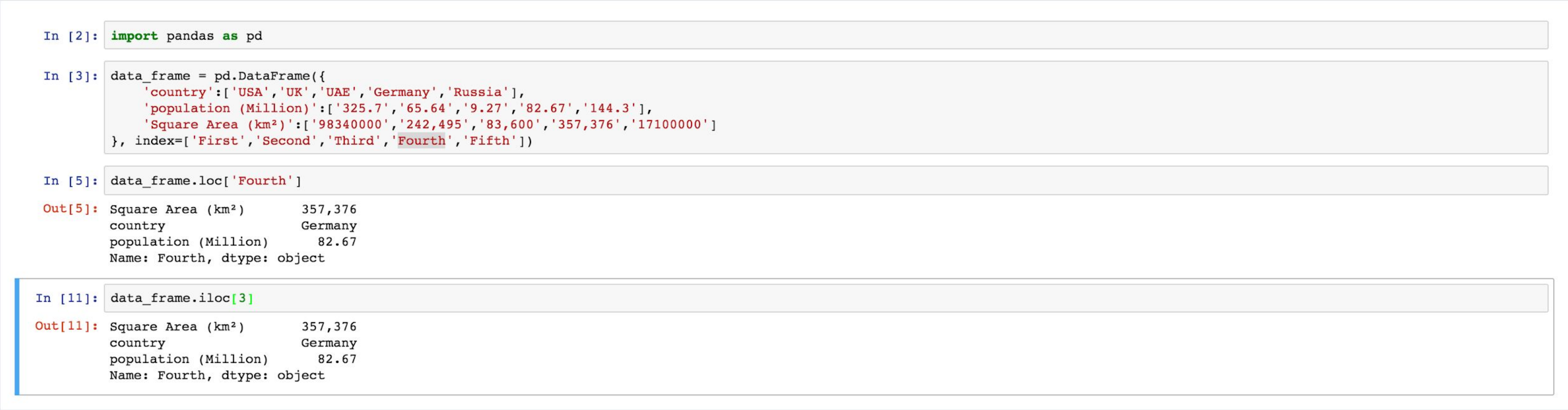
```
In [3]: data_frame = pd.DataFrame({
            'country':['USA','UK','UAE','Germany','Russia'],
            'population (Million)':['325.7','65.64','9.27','82.67','144.3'],
            'Square Area (km²)':['98340000','242,495','83,600','357,376','17100000']
        }, index=['First','Second','Third','Fourth','Fifth'])
```

```
In [4]: data_frame
Out[4]:
        Square Area (km²)  country  population (Million)
First            98340000      USA                 325.7
Second            242,495       UK                 65.64
Third              83,600      UAE                  9.27
Fourth            357,376  Germany                 82.67
Fifth            17100000   Russia                 144.3
```

IBM CODE

Operation On Pandas DataFrame Continued ..

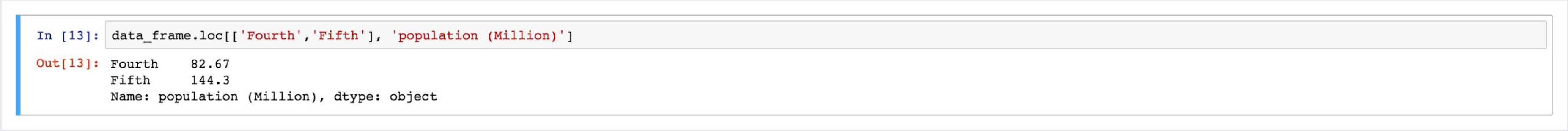
- Row access using index can be performed in several ways. The two most important ones are:-
    - using **.loc** and providing index label
    - using **.iloc** and providing index number



```
In [2]: import pandas as pd

In [3]: data_frame = pd.DataFrame({
            'country':['USA','UK','UAE','Germany','Russia'],
            'population (Million)':['325.7','65.64','9.27','82.67','144.3'],
            'Square Area (km²)':['98340000','242,495','83,600','357,376','17100000']
        }, index=['First','Second','Third','Fourth','Fifth'])

In [5]: data_frame.loc['Fourth']

Out[5]: Square Area (km²)       357,376
        country                 Germany
        population (Million)      82.67
        Name: Fourth, dtype: object

In [11]: data_frame.iloc[3]

Out[11]: Square Area (km²)       357,376
         country                 Germany
         population (Million)      82.67
         Name: Fourth, dtype: object
```

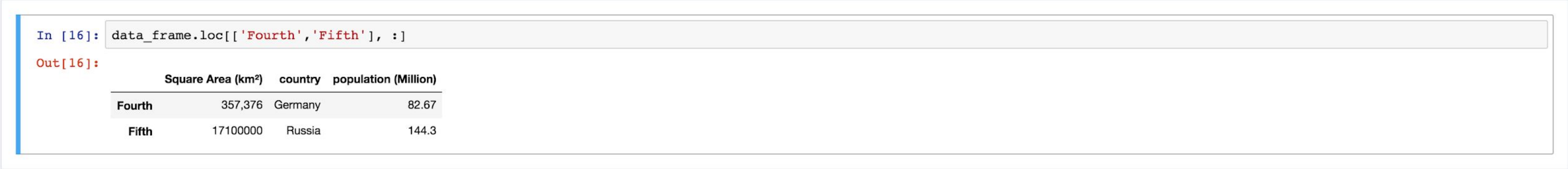
Operation On Pandas DataFrame Continued ..

- Selection of particular rows and columns



```
In [13]: data_frame.loc[['Fourth','Fifth'], 'population (Million)']

Out[13]: Fourth     82.67
         Fifth      144.3
         Name: population (Million), dtype: object
```

- **.loc** takes **2 arguments**: index list and column list, **slicing** operation is supported

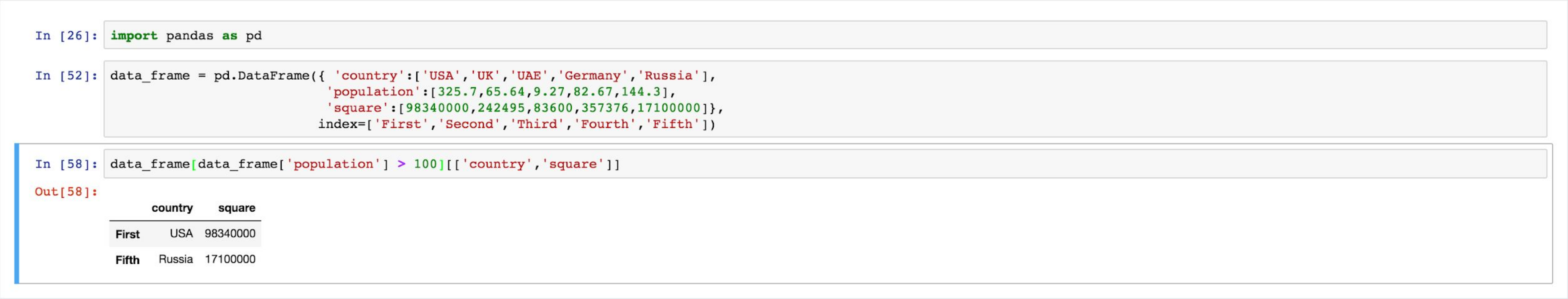


```
In [16]: data_frame.loc[['Fourth','Fifth'], :]

Out[16]:
```

	Square Area (km^2)	country	population (Million)
Fourth	357,376	Germany	82.67
Fifth	17100000	Russia	144.3

Operation On Pandas DataFrame Continued ..

- Filtering can be performed using Boolean arrays.

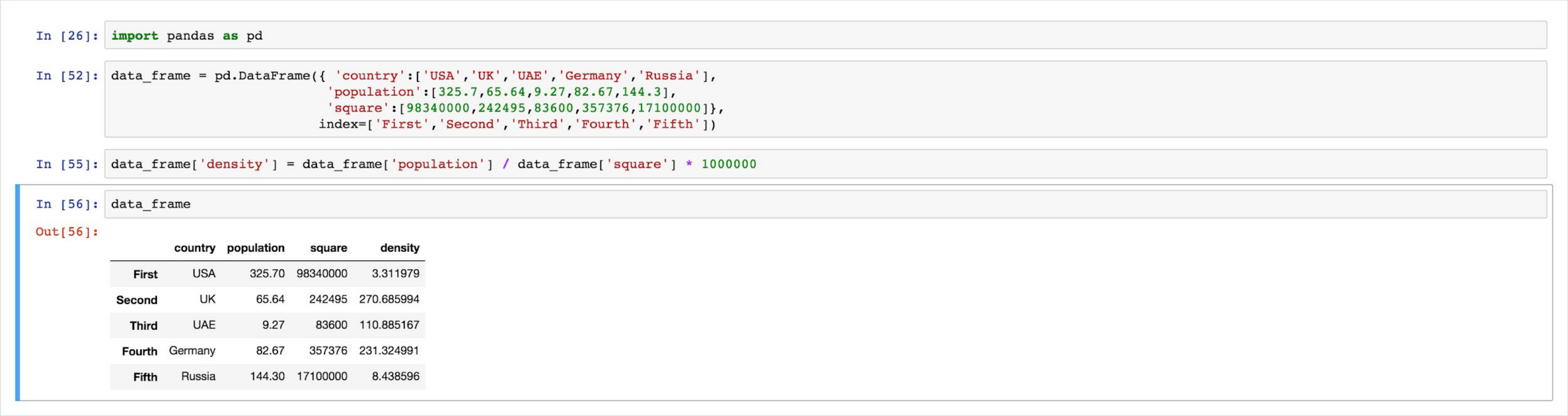


```
In [26]: import pandas as pd

In [52]: data_frame = pd.DataFrame({ 'country':['USA','UK','UAE','Germany','Russia'],
                                    'population':[325.7,65.64,9.27,82.67,144.3],
                                    'square':[98340000,242495,83600,357376,17100000]},
                                   index=['First','Second','Third','Fourth','Fifth'])

In [58]: data_frame[data_frame['population'] > 100][['country','square']]
Out[58]:
```

	country	square
First	USA	98340000
Fifth	Russia	17100000

Operation On Pandas DataFrame Continued ..

- Adding a new column, for example adding population density column.



```
In [26]: import pandas as pd
```

```
In [52]: data_frame = pd.DataFrame({ 'country':['USA','UK','UAE','Germany','Russia'],
                                    'population':[325.7,65.64,9.27,82.67,144.3],
                                    'square':[98340000,242495,83600,357376,17100000]},
                                   index=['First','Second','Third','Fourth','Fifth'])
```

```
In [55]: data_frame['density'] = data_frame['population'] / data_frame['square'] * 1000000
```

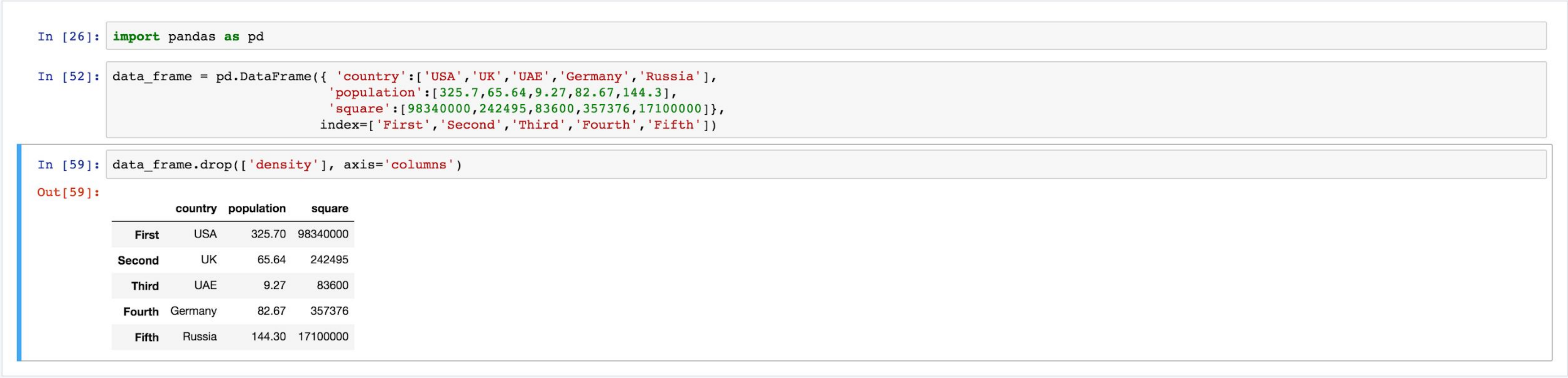
```
In [56]: data_frame
```

Out[56]:

	country	population	square	density
First	USA	325.70	98340000	3.311979
Second	UK	65.64	242495	270.685994
Third	UAE	9.27	83600	110.885167
Fourth	Germany	82.67	357376	231.324991
Fifth	Russia	144.30	17100000	8.438596

Operation On Pandas DataFrame Continued ..

- Deleting a column.



```
In [26]: import pandas as pd
```

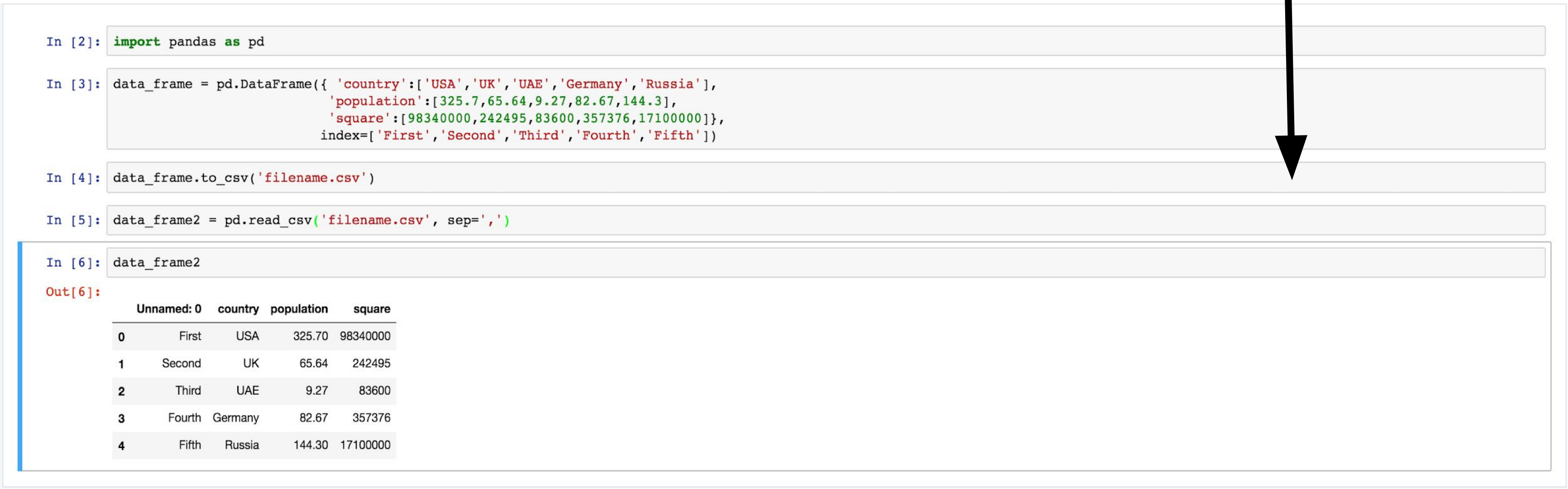
```
In [52]: data_frame = pd.DataFrame({ 'country':['USA','UK','UAE','Germany','Russia'],
                                    'population':[325.7,65.64,9.27,82.67,144.3],
                                    'square':[98340000,242495,83600,357376,17100000]},
                                   index=['First','Second','Third','Fourth','Fifth'])
```

```
In [59]: data_frame.drop(['density'], axis='columns')
```

```
Out[59]:
         country  population    square
  First      USA      325.70  98340000
 Second       UK       65.64    242495
  Third      UAE        9.27     83600
 Fourth  Germany       82.67    357376
  Fifth   Russia      144.30  17100000
```

Reading and Writing Files in Pandas

- Pandas support many popular file formats including **CSV, XML, HTML, Excel, SQL, JSON** many more.
- For example, writing a dataframe to a **CSV file** and then reading it.



```
In [2]: import pandas as pd

In [3]: data_frame = pd.DataFrame({ 'country':['USA','UK','UAE','Germany','Russia'],
                                   'population':[325.7,65.64,9.27,82.67,144.3],
                                   'square':[98340000,242495,83600,357376,17100000]},
                                  index=['First','Second','Third','Fourth','Fifth'])

In [4]: data_frame.to_csv('filename.csv')

In [5]: data_frame2 = pd.read_csv('filename.csv', sep=',')

In [6]: data_frame2

Out[6]:
     Unnamed: 0  country  population    square
  0       First      USA      325.70  98340000
  1      Second       UK       65.64    242495
  2       Third      UAE        9.27     83600
  3      Fourth  Germany       82.67    357376
  4       Fifth   Russia      144.30  17100000
```

Reading And Writing Files in Pandas Continued ..

- ***to_csv*** method takes many arguments for example, **separator character.**

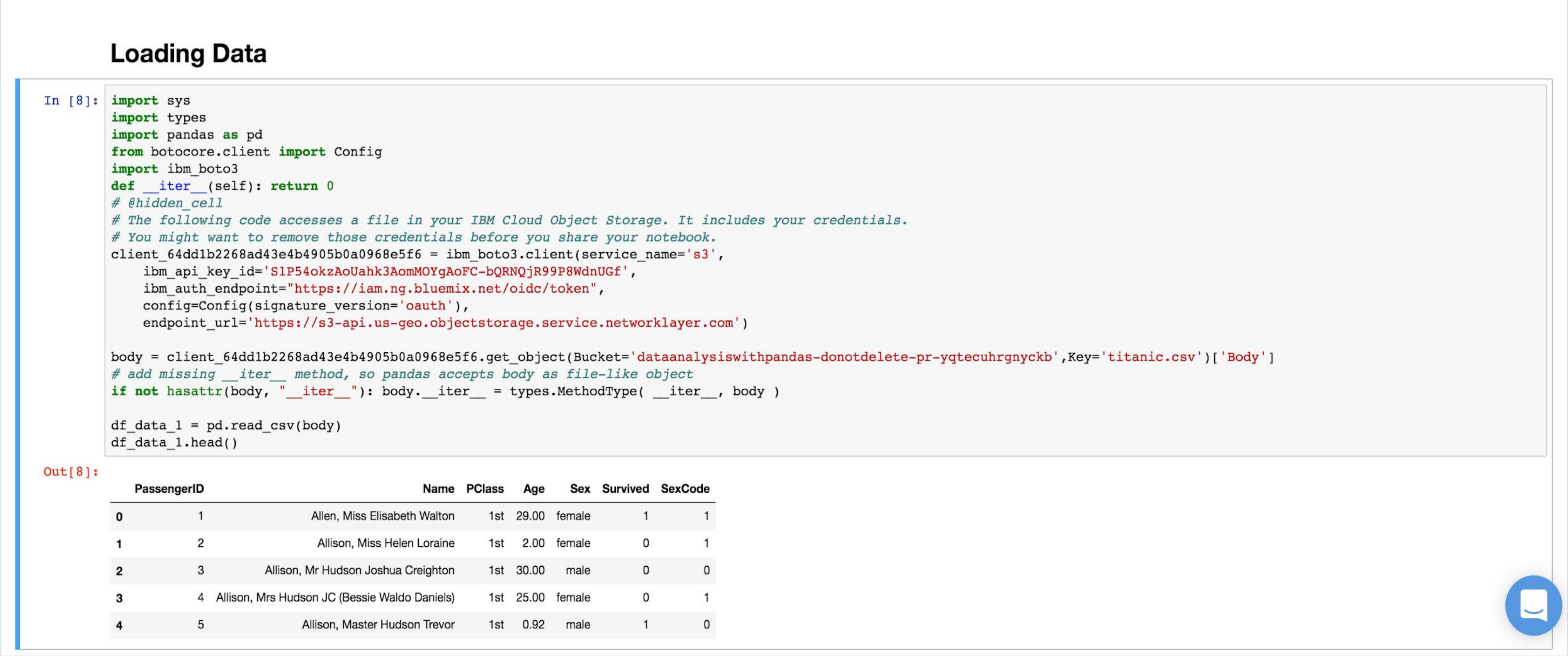
- As shown on the previous slides, named argument *sep* points to a separator character in CSV file called *filename.csv*.

- There are many different ways to construct DataFrame from external sources, for example using *read_sql* method pandas can perform SQL query and store results inside a new DataFrame instance

Aggregating and Grouping in Pandas

- Grouping is probably one of the most popular methods in data analysis. If you want to group data in pandas you have to use ***.groupby*** method.

- In order to demonstrate aggregates and grouping in pandas I decided to choose popular Titanic dataset (https://yadi.sk/d/TfhJdE2k3EyALt)



Loading Data

In [8]:

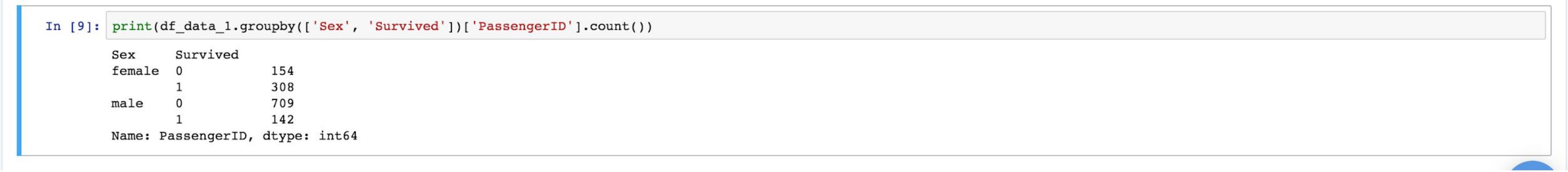
```
import sys
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3
def __iter__(self): return 0
# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share your notebook.
client_64dd1b2268ad43e4b4905b0a0968e5f6 = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='S1P54okzAoUahk3AomMOYgAoFC-bQRNQjR99P8WdnUGf',
    ibm_auth_endpoint="https://iam.ng.bluemix.net/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3-api.us-geo.objectstorage.service.networklayer.com')

body = client_64dd1b2268ad43e4b4905b0a0968e5f6.get_object(Bucket='dataanalysiswithpandas-donotdelete-pr-yqtecuhrgnyckb',Key='titanic.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df_data_1 = pd.read_csv(body)
df_data_1.head()
```

Out[8]:

	PassengerID	Name	PClass	Age	Sex	Survived	SexCode
0	1	Allen, Miss Elisabeth Walton	1st	29.00	female	1	1
1	2	Allison, Miss Helen Loraine	1st	2.00	female	0	1
2	3	Allison, Mr Hudson Joshua Creighton	1st	30.00	male	0	0
3	4	Allison, Mrs Hudson JC (Bessie Waldo Daniels)	1st	25.00	female	0	1
4	5	Allison, Master Hudson Trevor	1st	0.92	male	1	0

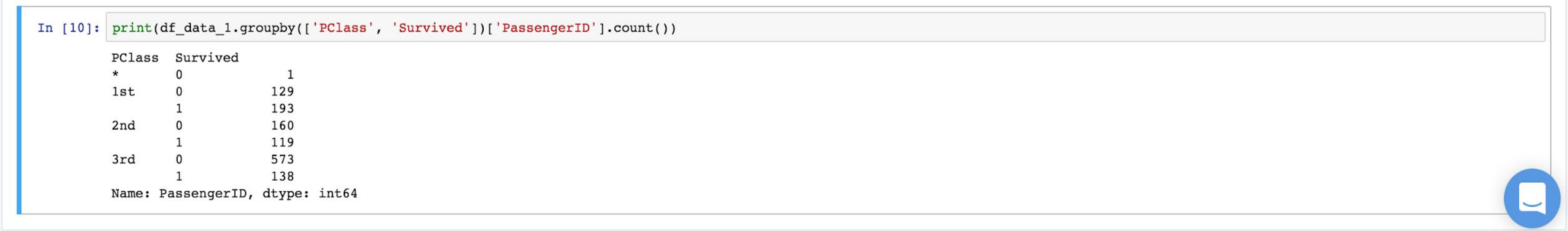
Aggregating and Grouping in Pandas Continued ..

- Let’s calculate how many passengers (women and men) survived and how many did not, we will use ***.groupby*** as stated above.



```
In [9]: print(df_data_1.groupby(['Sex', 'Survived'])['PassengerID'].count())

        Sex     Survived
        female  0           154
                1           308
        male    0           709
                1           142
        Name: PassengerID, dtype: int64
```

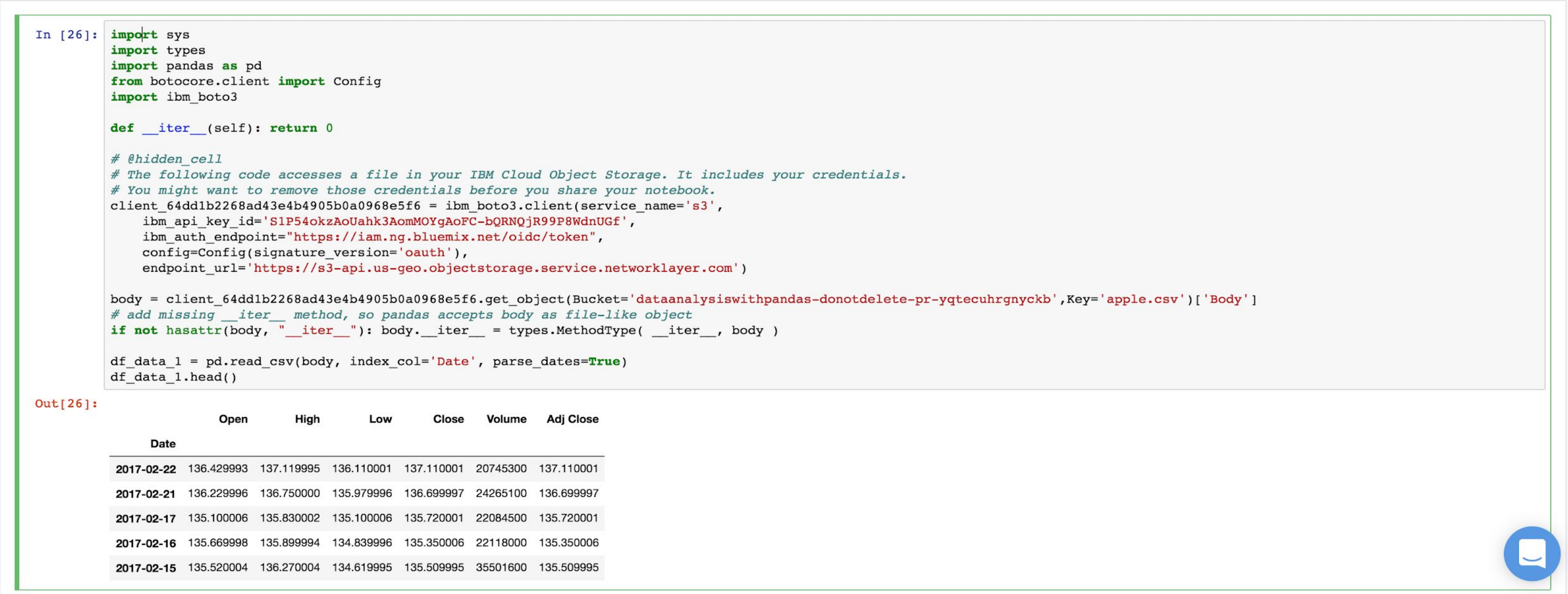
- Now, let’s analyze the same data by cabin class



```
In [10]: print(df_data_1.groupby(['PClass', 'Survived'])['PassengerID'].count())

         PClass  Survived
         *       0             1
         1st     0           129
                 1           193
         2nd     0           160
                 1           119
         3rd     0           573
                 1           138
         Name: PassengerID, dtype: int64
```

Time series analysis using Pandas

Pandas was created to analyze time series data. In order to illustrate how easy it is, We will use Apple’s last 5 year stock prices (https://yadi.sk/d/po_usmXT3ExwzV).



In [26]:

```
import sys
import types
import pandas as pd
from botocore.client import Config
import ibm_boto3

def __iter__(self): return 0

# @hidden_cell
# The following code accesses a file in your IBM Cloud Object Storage. It includes your credentials.
# You might want to remove those credentials before you share your notebook.
client_64dd1b2268ad43e4b4905b0a0968e5f6 = ibm_boto3.client(service_name='s3',
    ibm_api_key_id='S1P54okzAoUahk3AomMOYgAoFC-bQRNQjR99P8WdnUGf',
    ibm_auth_endpoint="https://iam.ng.bluemix.net/oidc/token",
    config=Config(signature_version='oauth'),
    endpoint_url='https://s3-api.us-geo.objectstorage.service.networklayer.com')

body = client_64dd1b2268ad43e4b4905b0a0968e5f6.get_object(Bucket='dataanalysiswithpandas-donotdelete-pr-yqtecuhrgnyckb',Key='apple.csv')['Body']
# add missing __iter__ method, so pandas accepts body as file-like object
if not hasattr(body, "__iter__"): body.__iter__ = types.MethodType( __iter__, body )

df_data_1 = pd.read_csv(body, index_col='Date', parse_dates=True)
df_data_1.head()
```

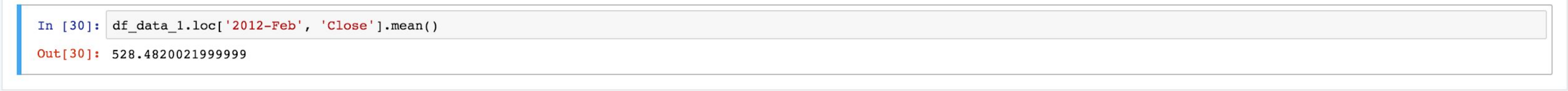
Out[26]:

	Open	High	Low	Close	Volume	Adj Close
Date						
2017-02-22	136.429993	137.119995	136.110001	137.110001	20745300	137.110001
2017-02-21	136.229996	136.750000	135.979996	136.699997	24265100	136.699997
2017-02-17	135.100006	135.830002	135.100006	135.720001	22084500	135.720001
2017-02-16	135.669998	135.899994	134.839996	135.350006	22118000	135.350006
2017-02-15	135.520004	136.270004	134.619995	135.509995	35501600	135.509995

Time series analysis using Pandas Continued ..

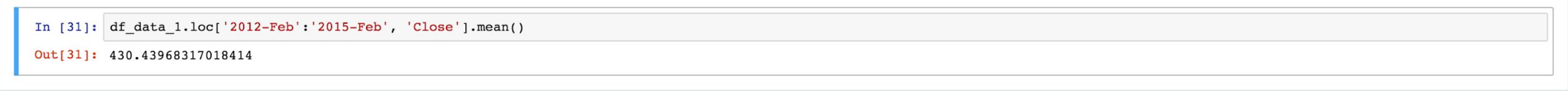
The image on the previous slide shows a sorted DataFrame with *DatetimeIndex* by *Date* column. If datetime column is different from ISO8601 format, then you have to use built-in pandas function ***pandas.to_datetime*** to format it.

- Let’s now calculate mean closing price.



```
In [30]: df_data_1.loc['2012-Feb', 'Close'].mean()
Out[30]: 528.4820021999999
```

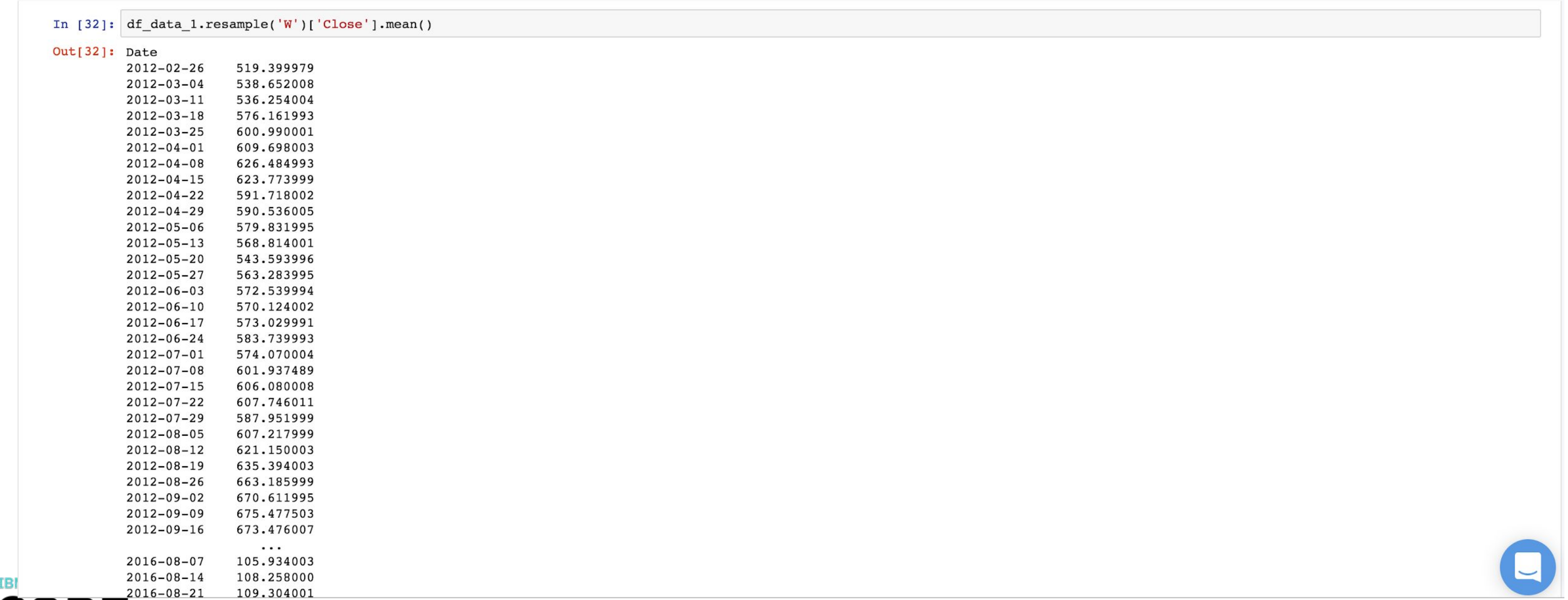
- But what about specific time period?



```
In [31]: df_data_1.loc['2012-Feb':'2015-Feb', 'Close'].mean()
Out[31]: 430.43968317018414
```

Time series analysis using Pandas Continued ..

- Let’s calculate mean of closing price by weeks.



```
In [32]: df_data_1.resample('W')['Close'].mean()

Out[32]: Date
         2012-02-26    519.399979
         2012-03-04    538.652008
         2012-03-11    536.254004
         2012-03-18    576.161993
         2012-03-25    600.990001
         2012-04-01    609.698003
         2012-04-08    626.484993
         2012-04-15    623.773999
         2012-04-22    591.718002
         2012-04-29    590.536005
         2012-05-06    579.831995
         2012-05-13    568.814001
         2012-05-20    543.593996
         2012-05-27    563.283995
         2012-06-03    572.539994
         2012-06-10    570.124002
         2012-06-17    573.029991
         2012-06-24    583.739993
         2012-07-01    574.070004
         2012-07-08    601.937489
         2012-07-15    606.080008
         2012-07-22    607.746011
         2012-07-29    587.951999
         2012-08-05    607.217999
         2012-08-12    621.150003
         2012-08-19    635.394003
         2012-08-26    663.185999
         2012-09-02    670.611995
         2012-09-09    675.477503
         2012-09-16    673.476007
                          ...
         2016-08-07    105.934003
         2016-08-14    108.258000
         2016-08-21    109.304001
```

Time series analysis using Pandas Continued ..

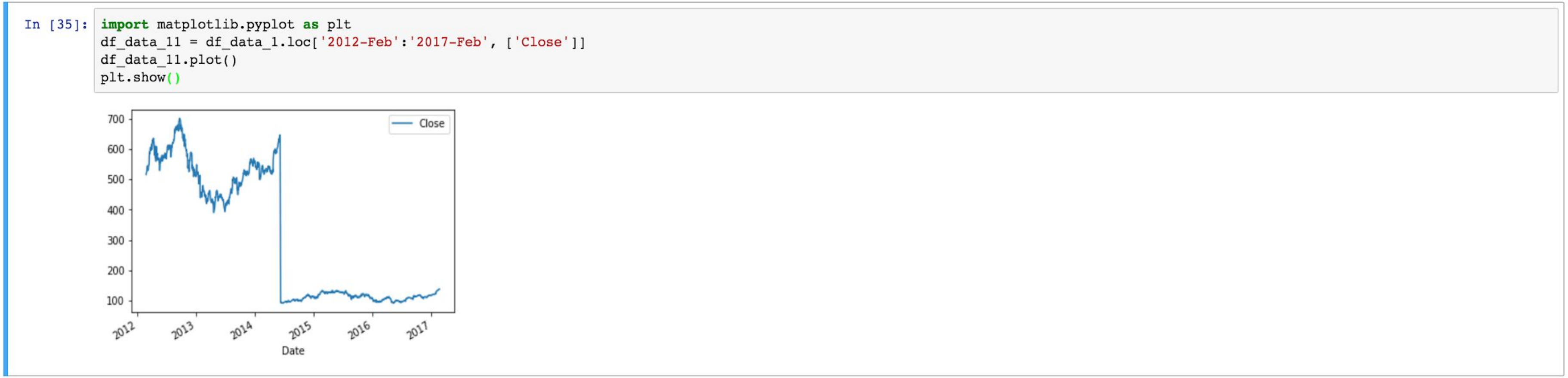
- Resampling is a very powerful tool when it comes to time series analysis.
- Resampling can be defined as a number of string aliases, given to useful common time series frequencies. In the above image, I am using “W”.
- For more information on resampling, refer pandas official documentation (http://pandas.pydata.org/pandas-docs/stable/timeseries.html#offset-aliases)

Visualization in Pandas

For visualization pandas use library called **matplotlib.**

Let’s see how Apple stock prices change over time on a graph:

- Taking Closing price between Feb, 2012 and Feb, 2017



```
In [35]: import matplotlib.pyplot as plt
         df_data_11 = df_data_1.loc['2012-Feb':'2017-Feb', ['Close']]
         df_data_11.plot()
         plt.show()
```

Visualization in Pandas Continued ..

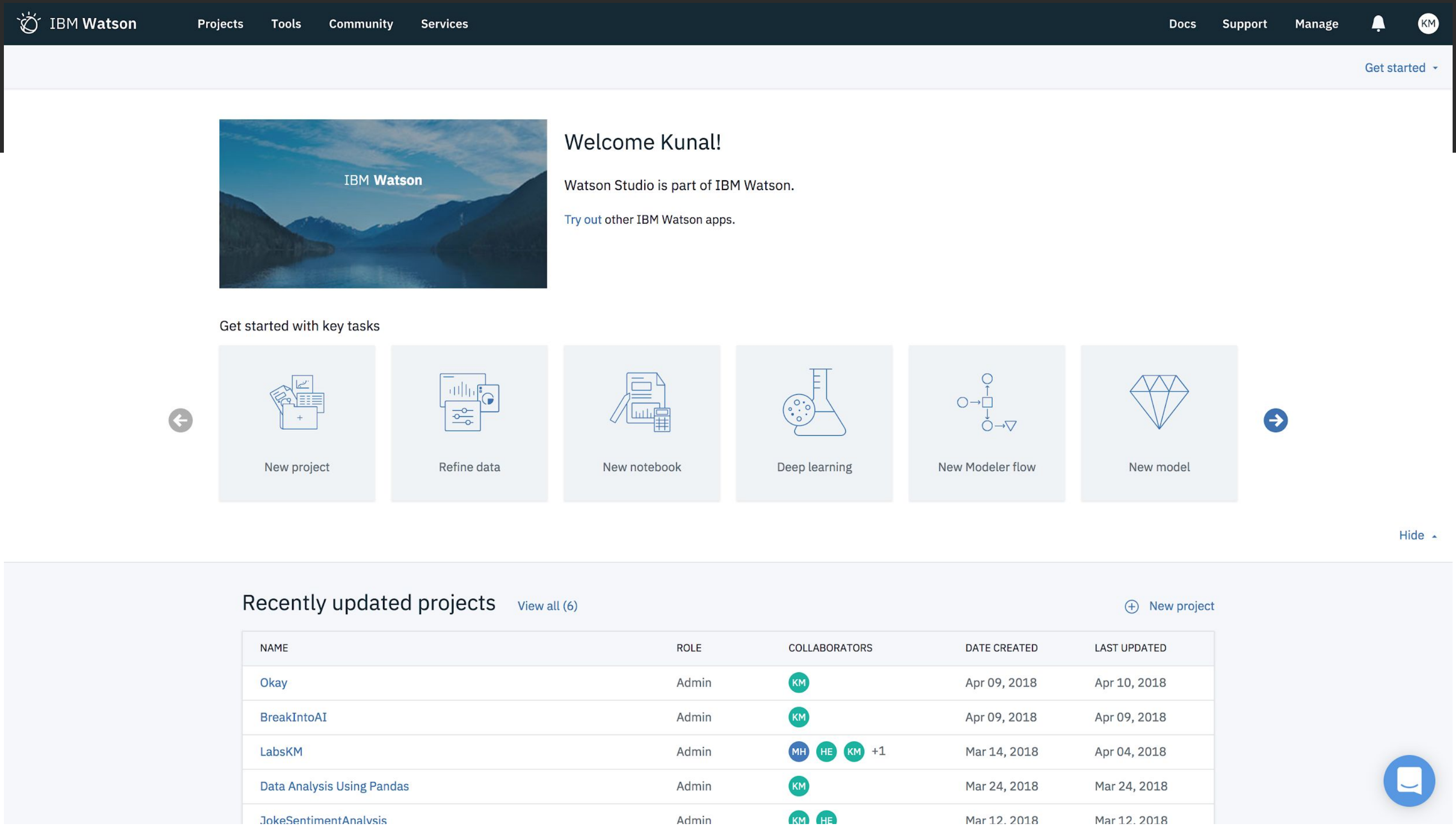
Values of X-axis are represented by index values of DataFrame (by default if not provide explicitly), Y-axis represents the closing price.

Advantage Of Pandas

Pandas have several advantages over other solutions such as NumPy, Statsmodels, SciPy etc.

1. A Pandas DataFrame can have non-homogeneous data i.e you can have different data types(int, float, string, datetime etc.) all in one place.
2. Good IO capabilities.
3. Pandas have built in functionality for a lot of common data-processing applications, for example:- easy group by syntax, easy joins (which is extremely efficient in pandas), rolling windows.

Introduction To IBM Watson Studio



Welcome Kunal!

Watson Studio is part of IBM Watson.

Try out other IBM Watson apps.

Get started with key tasks

Recently updated projects View all (6)

NAME	ROLE	COLLABORATORS	DATE CREATED	LAST UPDATED
Okay	Admin	KM	Apr 09, 2018	Apr 10, 2018
BreakIntoAI	Admin	KM	Apr 09, 2018	Apr 09, 2018
LabsKM	Admin	MH HE KM +1	Mar 14, 2018	Apr 04, 2018
Data Analysis Using Pandas	Admin	KM	Mar 24, 2018	Mar 24, 2018
JokeSentimentAnalysis	Admin	KM HE	Mar 12, 2018	Mar 12, 2018

IBM Watson Studio

- Watson Studio is a cloud based development and deployment environment for Machine Learning, Deep Learning, Data Governance and Data Exploration.
- A platform build for business analyst, data engineer, data scientist and developer to simplify their tasks with an intuitive UI and provide massive computing power.
- A platform where insights can be traced back to models, projects, notebooks and data sources and where model can evolve and automatically update themselves.

Advantages Of IBM Watson Studio For Data Analysis.

IBM Watson Studio is an IDE (Integrated Development Environment), available on IBM Cloud, with many advantages such as:-

1. Offering easier access to large amounts of data while decreasing the total time of analysis.
2. Rapid development experience with access to tools and utilities that break down language barriers.
3. Ability to integrate and connect to multiple data sources, allows refining, and accessing big data engines.

May the force of Pandas Be With You :)

IBM

CODE

IBM