2.Fundamentals of python

Dr. P. Kalpana, M.E., PhD.
Faculty of Mechanical Engineering
IIITDM Kancheepuram

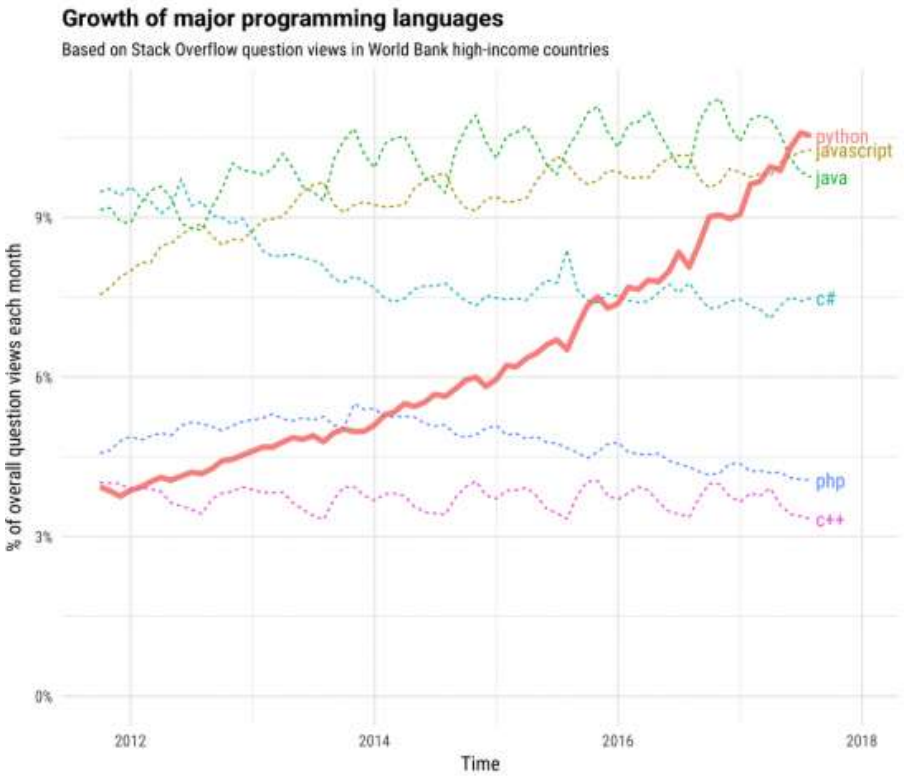
Why Python?

- Simple and easy to use
- free software and open source
- Interpreted
- Dynamically typed
- Extensible
- Embedded
- Extensive Libraries
- Usability
  - Desk top and web applications
  - Data base applications
  - Networking applications
  - Data Analysis (Data Science)
  - Machine Learning
  - IoT and AI Applications
  - Games

Objectives

- Installing python
- Fundamentals of python
- Data visualization

Top Companies using python

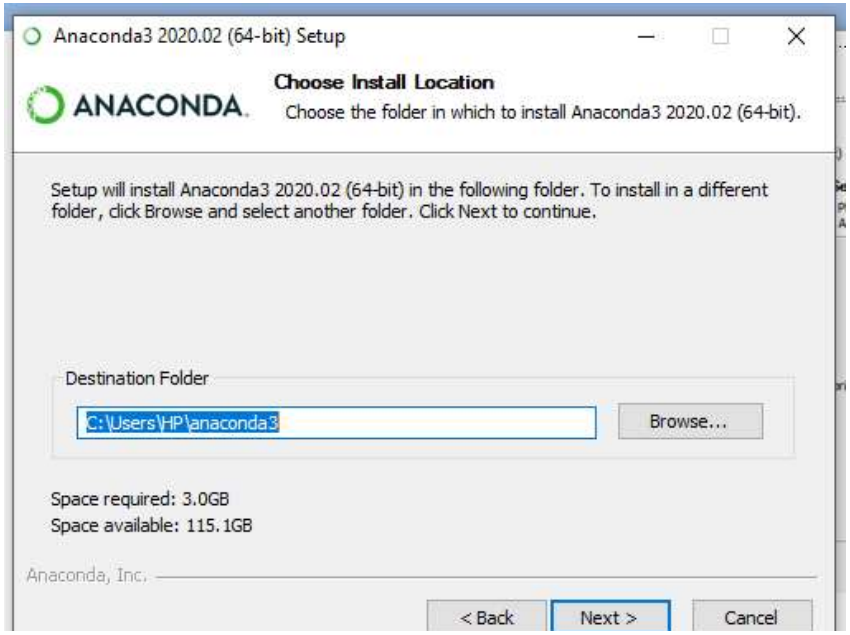


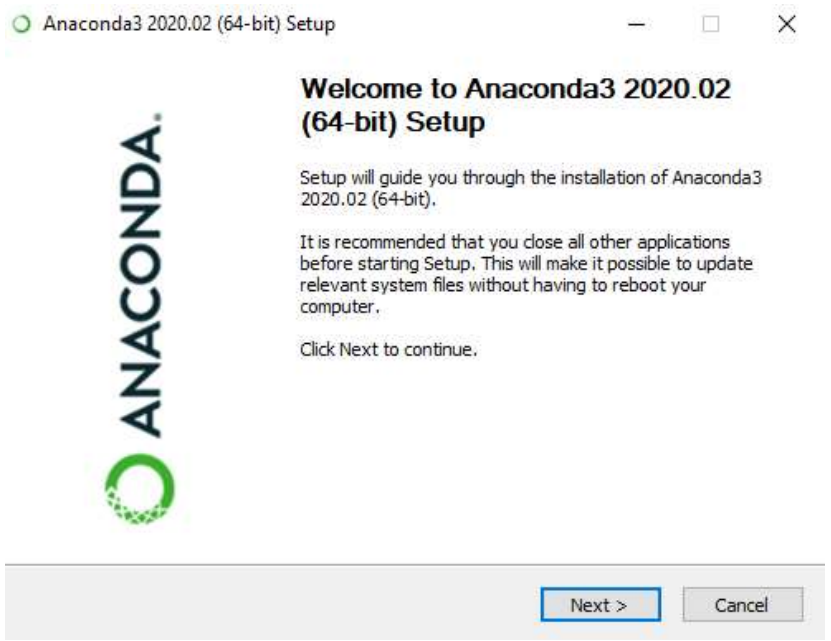
Jupyter Note Book

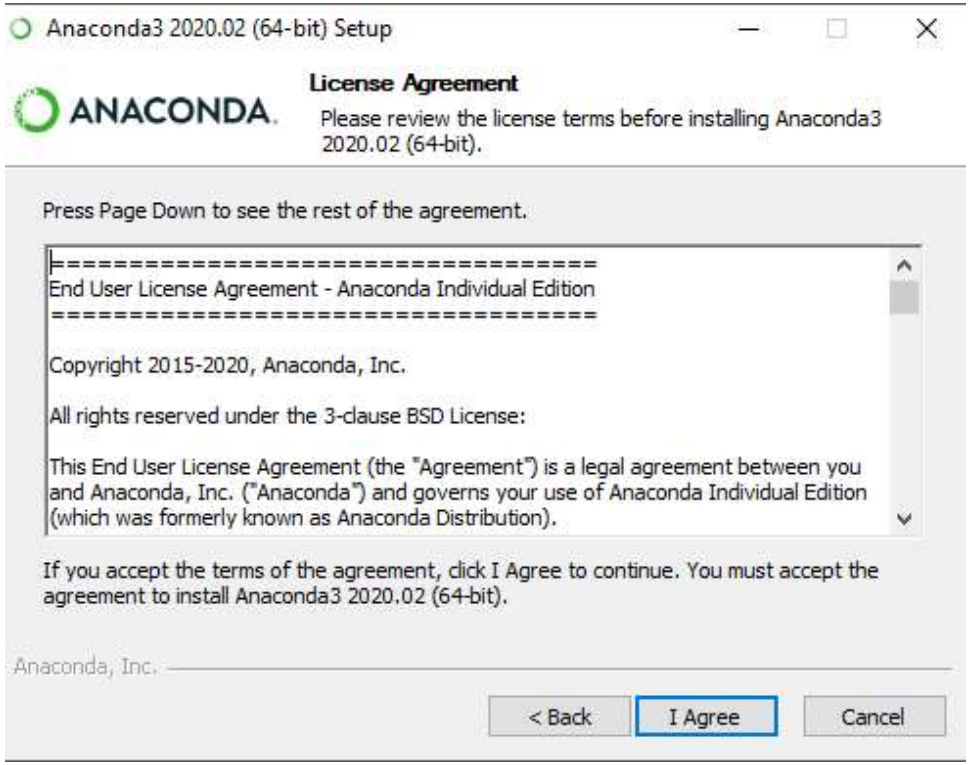
- client-server application
- Edit code on the web browser
- Easy in documentation
- Easy in demonstration
- User friendly

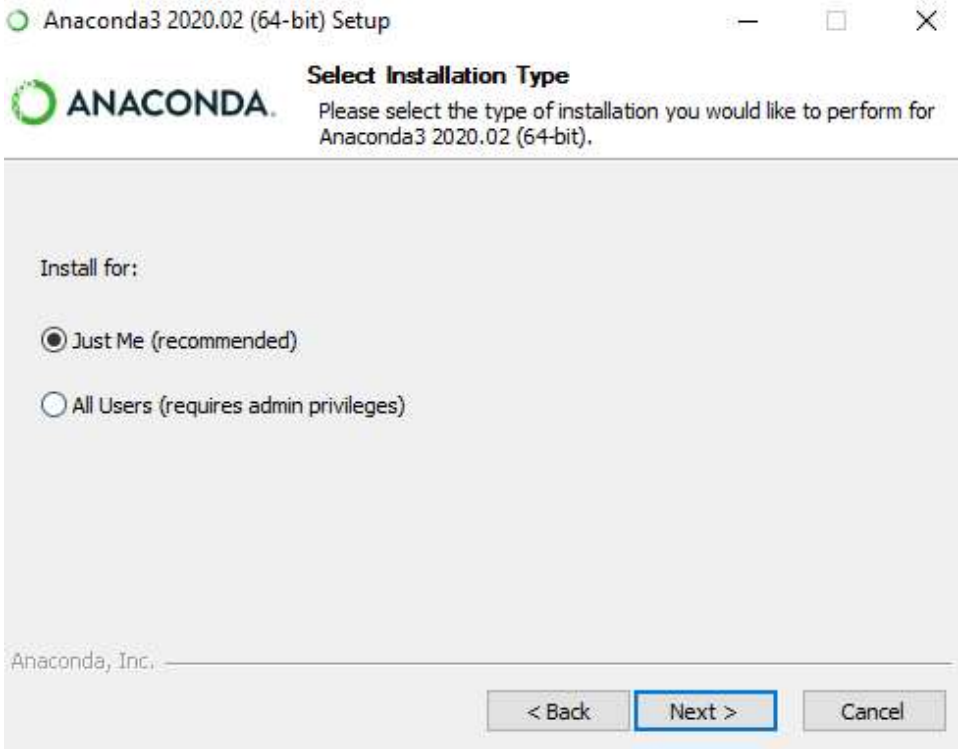
Installation of Python

- Installation Process
- Step 1:Type https//:www.anaconda .com at the address bar of the web browser
- Step 2:click on down load button
- Step 3:download python 3.7 version for windows OS
- Step 4:Double click on file to run the application
- Step 5: Follow the instructions until completion of the process

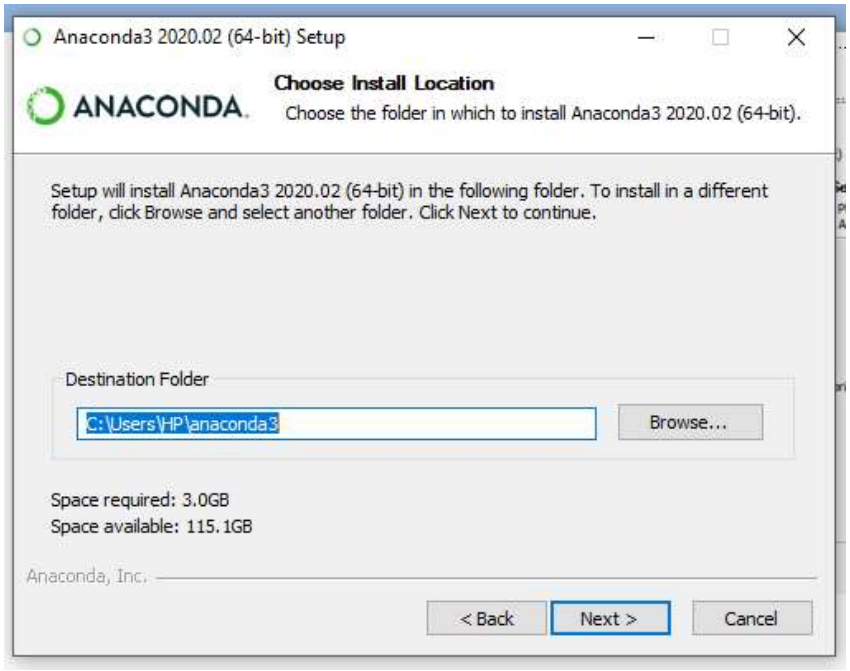


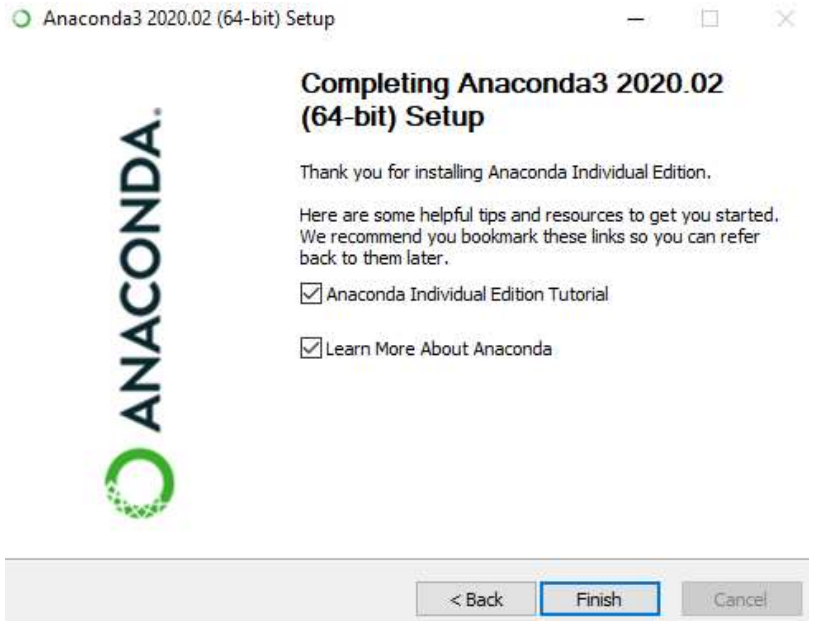



Anaconda3 2020.02 (64-bit) Setup
ANACONDA.
License Agreement
Please review the license terms before installing Anaconda3 2020.02 (64-bit).
Press Page Down to see the rest of the agreement.
End User License Agreement - Anaconda Individual Edition
Copyright 2015-2020, Anaconda, Inc.
All rights reserved under the 3-clause BSD License:
This End User License Agreement (the "Agreement") is a legal agreement between you and Anaconda, Inc. ("Anaconda") and governs your use of Anaconda Individual Edition (which was formerly known as Anaconda Distribution).
If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Anaconda3 2020.02 (64-bit).
Anaconda, Inc.
< Back
I Agree
Cancel

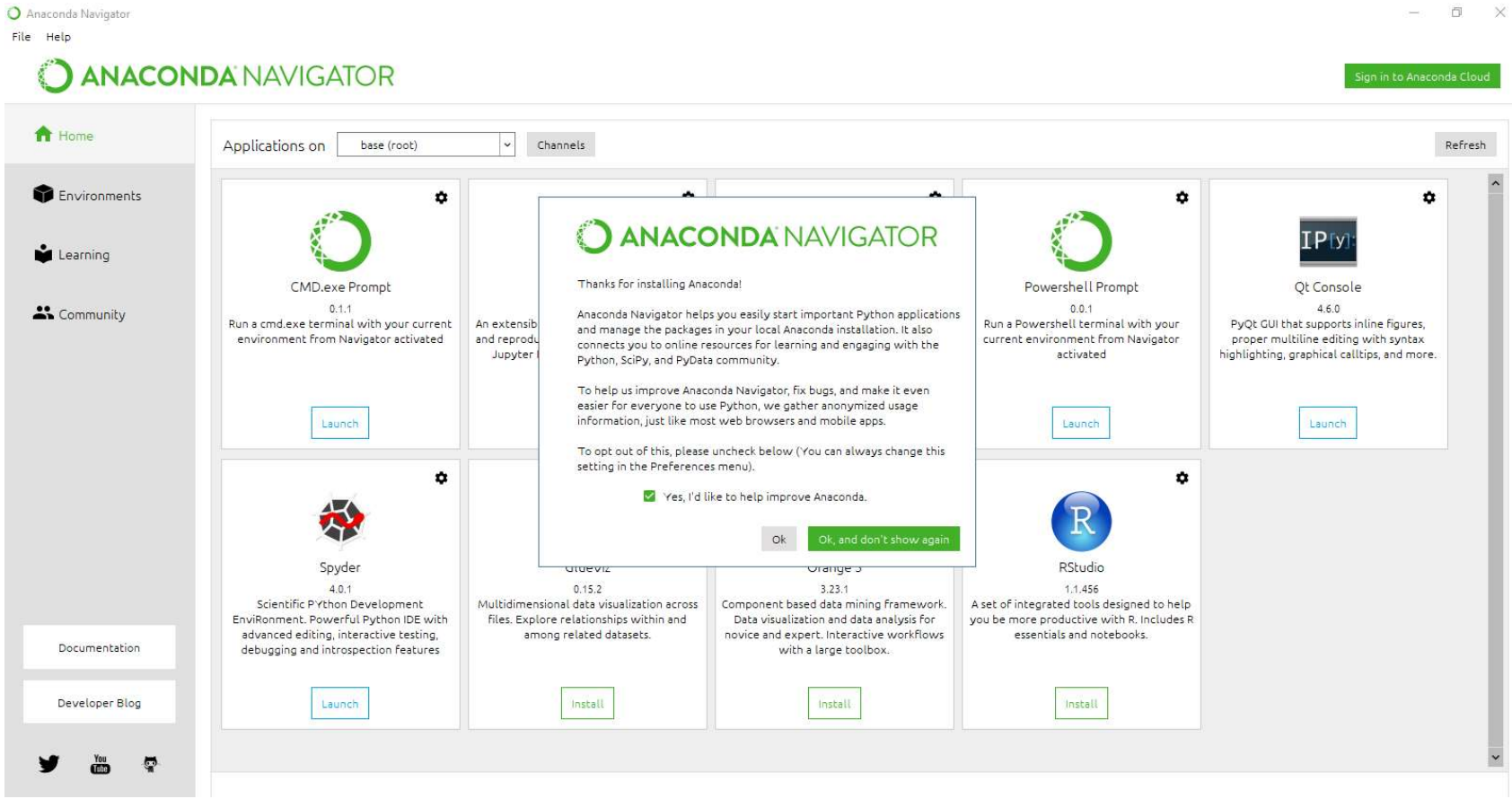

Anaconda3 2020.02 (64-bit) Setup
ANACONDA.
Select Installation Type
Please select the type of installation you would like to perform for Anaconda3 2020.02 (64-bit).
Install for:
Just Me (recommended)
All Users (requires admin privileges)
Anaconda, Inc.
< Back
Next >
Cancel

Python Installation Process

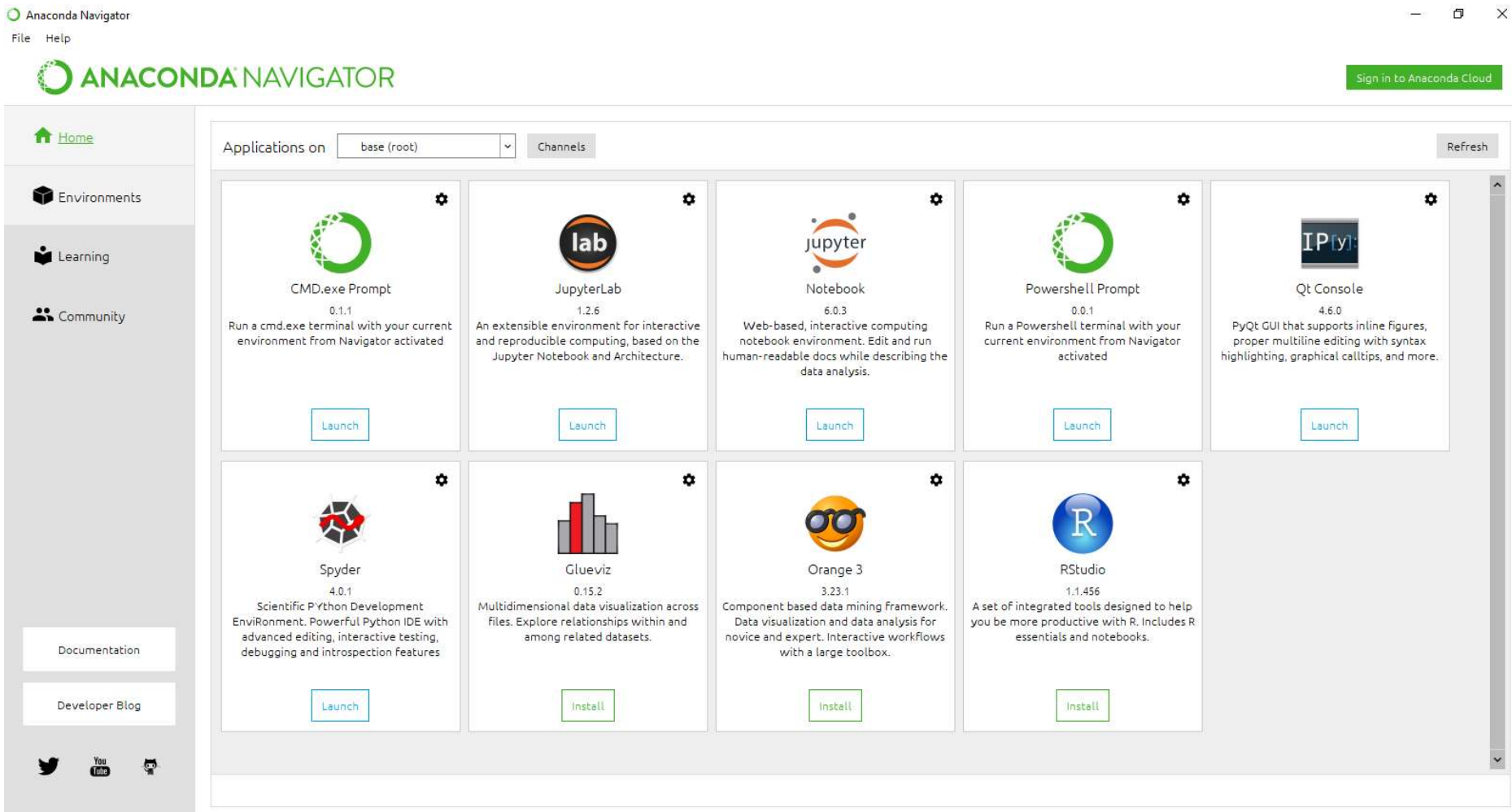




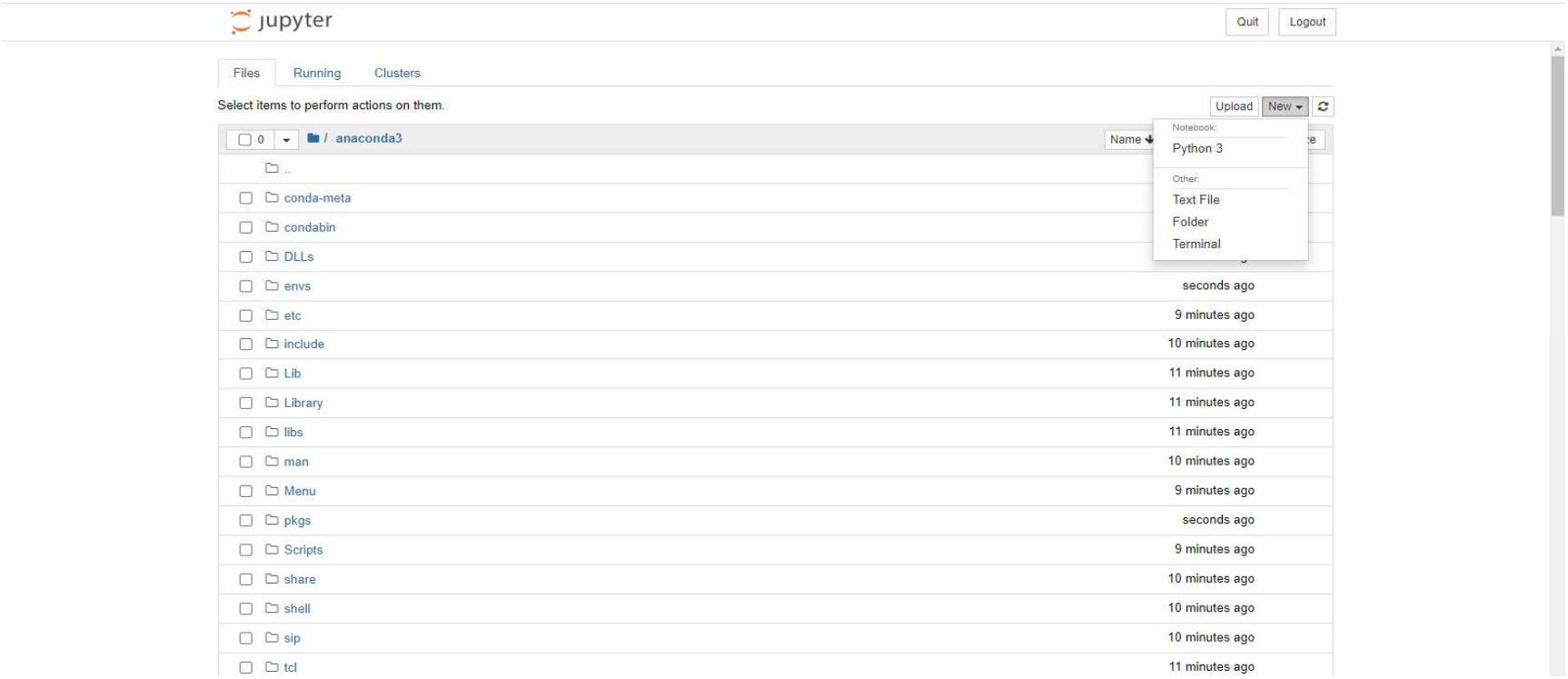
Python Installation Process



Python Installation Process

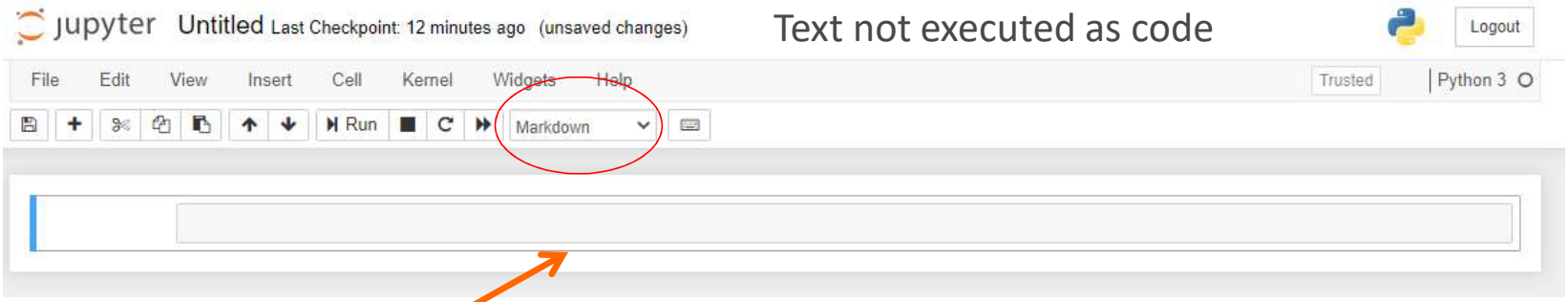


About Jupyter Notebook

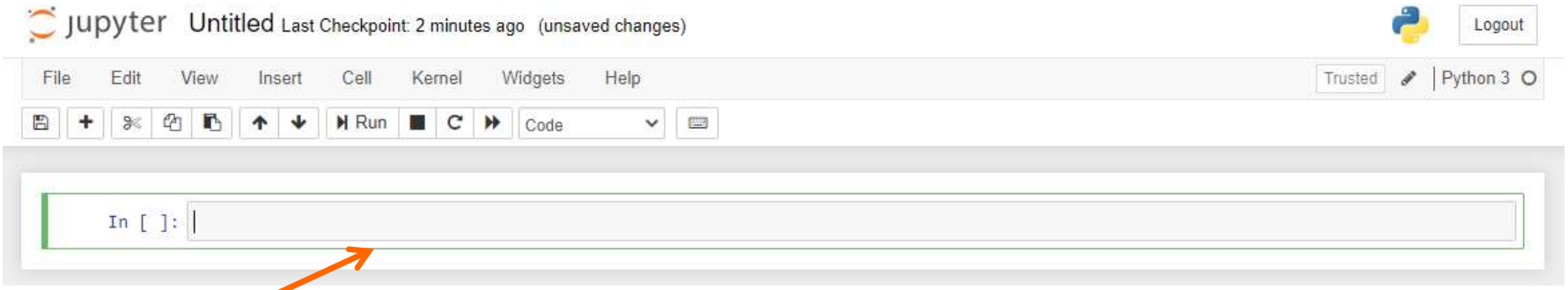


About Jupyter Notebook

It contains documentation
Text not executed as code



Cell<-Access using enter key



Input Field<-Green color indicated Edit Mode
<-Blue color indicates command mode

About Jupyter Notebook

- Command allows to edit notebook as whole
- To close edit mode- press escape Key
- Execution three ways
  - Ctrl + Enter (output field can not be modified)
    - run selected cells
  - Shift + enter (output field is modified)
    - run the current cell, select below
  - Run Button on Jupiter Interface
  - Alt + Enter run the current cell, insert below
- Comment line is written preceding with #symbol
- A insert cell above
- B insert cell below
- D, D (press the key twice) delete selected cells
- Y change the cell type to *Code*
- M change the cell type to *Markdown*

Fundamentals of python

- Loading a simple delimited data file
- Counting how many rows and columns are loaded
- Determining which type of data is loaded
- Looking at different parts of the data by subsetting rows and columns

- Data Source: https://github.com/jennybc/gapminder/blob/master/data/gapminder.rdata
- www.github.com/jennybc/gapminder
- pip install pyreadr
- import pandas as pd
- import numpy as np
- import matplotlib.pyplot as plt
- import pyreadr
- df=pyreadr.read_r('E:/IIITDM/Courses/Data_Analytics/Data/gapminder.rdata')
- df1=pd.read_csv(" E:/IIITDM/Courses/Data_Analytics/Data/gapminder.csv ")

Fundamentals of python

write.csv (gapminder,"E:/IIITDM/Courses/Data_Analytics/Data/gapminder.csv", row.names = FALSE)

df1=pd.read_csv("E:\IIITDM\Courses\Data_Analytics\Data\gapminder.csv")

```
In [10]: df1=pd.read_csv("E:\IIITDM\Courses\Data_Analytics\Data\gapminder.csv")

In [11]: df1
Out[11]:
          country continent  year  lifeExp       pop   gdpPercap
0     Afghanistan      Asia  1952   28.801   8425333  779.445314
1     Afghanistan      Asia  1957   30.332   9240934  820.853030
2     Afghanistan      Asia  1962   31.997  10267083  853.100710
3     Afghanistan      Asia  1967   34.020  11537966  836.197138
4     Afghanistan      Asia  1972   36.088  13079460  739.981106
...           ...       ...   ...      ...       ...         ...
1699     Zimbabwe    Africa  1987   62.351   9216418  706.157306
1700     Zimbabwe    Africa  1992   60.377  10704340  693.420786
1701     Zimbabwe    Africa  1997   46.809  11404948  792.449960
1702     Zimbabwe    Africa  2002   39.989  11926563  672.038623
1703     Zimbabwe    Africa  2007   43.487  12311143  469.709298

[1704 rows x 6 columns]
```

Fundamentals of python

- To know the header of the table
- print(df1.head())
- Head function gives first five rows of the table

```
In [21]: print(df1.head())
```

```
       country continent  year  lifeExp       pop   gdpPercap
0  Afghanistan      Asia  1952   28.801   8425333  779.445314
1  Afghanistan      Asia  1957   30.332   9240934  820.853030
2  Afghanistan      Asia  1962   31.997  10267083  853.100710
3  Afghanistan      Asia  1967   34.020  11537966  836.197138
4  Afghanistan      Asia  1972   36.088  13079460  739.981106
```

- To show the size of the Table (Number of rows and columns)
- print(df1.shape)

```
In [22]: print(df1.shape)
```

```
(1704, 6)
```

Fundamentals of python

• TO get the column Names

print(df1.columns)

```
In [24]: print(df1.columns)

Index(['country', 'continent', 'year', 'lifeExp', 'pop', 'gdpPercap'], dtype='object')
```

To get the data types of each column

```
In [25]: print (df1.dtypes)

country       object
continent     object
year           int64
lifeExp      float64
pop            int64
gdpPercap    float64
dtype: object
```

Panda types Vs Python Types

Pandas Type	Python Type	Description
`object`	`string`	Most common data type
`int64`	`int`	Whole numbers
`float64`	`float`	Numbers with decimals
`datetime64`	`datetime`	`datetime` is found in the Python standard library (i.e., it is not loaded by default and needs to be imported)

Source:**Pandas for Everyone , Daniel Y. Chen,**

Fundamentals of python

To get more information about the data

```
In [27]: print(df1.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1704 entries, 0 to 1703
Data columns (total 6 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   country    1704 non-null   object 
 1   continent  1704 non-null   object 
 2   year       1704 non-null   int64  
 3   lifeExp    1704 non-null   float64
 4   pop        1704 non-null   int64  
 5   gdpPercap  1704 non-null   float64
dtypes: float64(2), int64(2), object(2)
memory usage: 80.0+ KB
None
```

Looking at Columns, Rows, and Cells

- **Subsetting Columns**
- If we want to examine multiple columns, we can specify them by names, positions, or ranges.

Subsetting Columns by Name

just get the country column and save it to its own variable

country_df = df['country']

show the first 5 observations

print(country_df.head())

```
In [28]: country_df = df1['country']

In [29]: country_df

Out[29]: 0       Afghanistan
         1       Afghanistan
         2       Afghanistan
         3       Afghanistan
         4       Afghanistan
                    ...
         1699        Zimbabwe
         1700        Zimbabwe
         1701        Zimbabwe
         1702        Zimbabwe
         1703        Zimbabwe
         Name: country, Length: 1704, dtype: object

In [30]: country_df.head(5)

Out[30]: 0    Afghanistan
         1    Afghanistan
         2    Afghanistan
         3    Afghanistan
         4    Afghanistan
         Name: country, dtype: object
```

Looking at Columns, Rows, and Cells

- *show the last 5 observations*

```
In [31]: print(country_df.tail())

1699    Zimbabwe
1700    Zimbabwe
1701    Zimbabwe
1702    Zimbabwe
1703    Zimbabwe
Name: country, dtype: object
```

print(subset.tail())

```
In [34]: print(subset.tail(5))

      country continent  year
1699  Zimbabwe    Africa  1987
1700  Zimbabwe    Africa  1992
1701  Zimbabwe    Africa  1997
1702  Zimbabwe    Africa  2002
1703  Zimbabwe    Africa  2007
```

- *Looking at country, continent, and year*
- subset = df[['country', 'continent', 'year']]
  print(subset.head())

```
In [32]: subset = df1[['country', 'continent', 'year']]

In [33]: print(subset.head(5))

       country continent  year
0  Afghanistan      Asia  1952
1  Afghanistan      Asia  1957
2  Afghanistan      Asia  1962
3  Afghanistan      Asia  1967
4  Afghanistan      Asia  1972
```

Subsetting Rows

- Rows can be subset in multiple ways, by row name or row index

Subset method	Description
`loc`	Subset based on index label (row name)
`iloc`	Subset based on row index (row number)

Subset Rows by Index Label: loc

Python counts from 0th Row

```
In [36]: print(df1.head())
```

```
       country continent  year  lifeExp       pop   gdpPercap
0  Afghanistan      Asia  1952   28.801   8425333  779.445314
1  Afghanistan      Asia  1957   30.332   9240934  820.853030
2  Afghanistan      Asia  1962   31.997  10267083  853.100710
3  Afghanistan      Asia  1967   34.020  11537966  836.197138
4  Afghanistan      Asia  1972   36.088  13079460  739.981106
```

```
In [38]: print(df1.loc[0])
```

```
country      Afghanistan
continent           Asia
year                1952
lifeExp           28.801
pop              8425333
gdpPercap        779.445
Name: 0, dtype: object
```

Subsetting Rows

- To get the 100th row
- print(df1.loc[99])

```
In [39]: print(df1.loc[99])
```

```
country      Bangladesh
continent          Asia
year               1967
lifeExp          43.453
pop            62821884
gdpPercap       721.186
Name: 99, dtype: object
```

Subsetting Rows

- *# get the last row*
  # this will cause an error
- **print**(df.loc[-1])

```
In [40]: print(df1.loc[-1])
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\anaconda3\lib\site-packages\pandas\core\indexes\range.py in get_loc(self, key, method, tolerance)
    349             try:
--> 350                 return self._range.index(new_key)
    351             except ValueError:

ValueError: -1 is not in range

During handling of the above exception, another exception occurred:

KeyError                                  Traceback (most recent call last)
<ipython-input-40-450ed06f07d6> in <module>
----> 1 print(df1.loc[-1])

~\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
   1765
   1766             maybe_callable = com.apply_if_callable(key, self.obj)
-> 1767             return self._getitem_axis(maybe_callable, axis=axis)
   1768
   1769     def _is_scalar_access(self, key: Tuple):
```

Subsetting Rows

- *# get the last row (correctly)*
  # use the first value given from shape to get the number of rows

```
In [41]: number_of_rows = df1.shape[0]

In [42]: number_of_rows
Out[42]: 1704

In [43]: last_row_index = number_of_rows - 1
```

- *# now do the subset using the index of the last row*
  print(df1.loc[last_row_index])

```
In [45]: print(df1.loc[last_row_index])
country      Zimbabwe
continent      Africa
year             2007
lifeExp        43.487
pop          12311143
gdpPercap     469.709
Name: 1703, dtype: object
```

print(df1.tail(n=1))

```
In [47]: print(df1.tail(n=1))
      country continent  year  lifeExp      pop   gdpPercap
1703 Zimbabwe    Africa  2007   43.487 12311143  469.709298
```

Subsetting Rows

Notice that when we used tail() and loc, the results were printed out differently.

```
In [48]: subset_loc=df1.loc[0]
         subset_head=df1.head(n=1)

In [49]: subset_loc

Out[49]: country      Afghanistan
         continent           Asia
         year                1952
         lifeExp           28.801
         pop              8425333
         gdpPercap        779.445
         Name: 0, dtype: object

In [50]: subset_head
```

Out[50]:

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314

Subsetting Multiple Rows

 # select the first, 100th, and 1000th rows
 # note the double square brackets similar to the syntax used to subset multiple columns

print(df1.loc[[0,99,999]])

```
In [51]: print(df1.loc[[0,99,999]])
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
99	Bangladesh	Asia	1967	43.453	62821884	721.186086
999	Mongolia	Asia	1967	51.253	1149500	1226.041130

Subset Rows by Row Number: iloc

- *# get the 2nd row* **print**(df.iloc[0])

```
In [52]: print(df1.iloc[0])
```

```
country      Afghanistan
continent           Asia
year                1952
lifeExp           28.801
pop              8425333
gdpPercap        779.445
Name: 0, dtype: object
```

- *## get the 100th row* **print**(df.iloc[99])

```
In [53]: print(df1.iloc[99])
```

```
country      Bangladesh
continent           Asia
year                1967
lifeExp           43.453
pop             62821884
gdpPercap        721.186
Name: 99, dtype: object
```

Subset Rows by Row Number: iloc

- With iloc, we can pass in the -1 to get the last row—something we couldn't do with loc.
- *# using -1 to get the last row*
  print(df.iloc[-1])

```
In [54]: print(df1.iloc[-1])

country      Zimbabwe
continent      Africa
year             2007
lifeExp        43.487
pop          12311143
gdpPercap     469.709
Name: 1703, dtype: object
```

Subset Rows by Row Number: iloc

- *## get the first, 100th, and 1000th rows*
  print(df.iloc[[0, 99, 999]])

```
In [55]: print(df1.iloc[[0, 99, 999]])
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
99	Bangladesh	Asia	1967	43.453	62821884	721.186086
999	Mongolia	Asia	1967	51.253	1149500	1226.041130

Mixing It Up

- The loc and iloc attributes can be used to obtain subsets of columns, rows, or both.
- The general syntax for loc and iloc uses square brackets with a comma. The part to the left of the comma is the row values to subset;
- the part to the right of the comma is the column values to subset.
- df1.loc[[rows], [columns]] or df1.iloc[[rows], [columns]]
- df1.loc[[0], ['year']]

```
In [58]: df1.loc[[0], ['year']]
```

Out[58]:

	year
0	1952

Subsetting Columns

- must use Python's slicing syntax
- The Python slicing syntax uses a colon, :
- colon, the attribute refers to everything
- to subset the column(s). df.loc[:, [columns]]

- *# subset columns with loc*
  # note the position of the colon
  # it is used to select all rows
- subset = df1.loc[:, ['year', 'pop']]
- print(subset.head())

```
In [60]: subset = df1.loc[:, ['year', 'pop']]
         print(subset.head())
```

```
   year       pop
0  1952   8425333
1  1957   9240934
2  1962  10267083
3  1967  11537966
4  1972  13079460
```

Subsetting Columns

- *# subset columns with iloc*
  # iloc will alow us to use integers
  # -1 will select the last column
  subset = df.iloc[:, [2, 4, -1]]
  print(subset.head())

```
In [61]: subset = df1.iloc[:, [2, 4, -1]]
         print(subset.head())
```

	year	pop	gdpPercap
0	1952	8425333	779.445314
1	1957	9240934	820.853030
2	1962	10267083	853.100710
3	1967	11537966	836.197138
4	1972	13079460	739.981106

Subsetting Columns

- *# subset columns with loc*
  # but pass in integer values
  # this will cause an error
  subset = df1.loc[:, [2, 4, -1]]
  print(subset.head())

```
Traceback (most recent call last):
  File "<ipython-input-1-719bcb04e3c1>", line
2, in <module>
    subset = df.loc[:, [2, 4, -1]]
KeyError: 'None of [[2, 4, -1]] are in the
[columns]'
```

- *# subset columns with iloc*
  # but pass in index names
  # this will cause an error
- subset = df1.iloc[:, ['year', 'pop']]
  print(subset.head()

```
Traceback (most recent call last):
  File "<ipython-input-1-43f52fceab49>", line
2, in <module>
    subset = df.iloc[:, ['year', 'pop']]
TypeError: cannot perform reduce with flexible
type
```

Subsetting Columns by Range

- built-in range function to create a range of values in Python
- range(5) is called, five integers are returned: 0 – 4.
- *# create a range of integers from 0 to 4 inclusive*
- small_range = list(range(5))
  print(small_range)

```
In [64]: small_range = list(range(5))
         print(small_range)

[0, 1, 2, 3, 4]
```

- *# subset the dataframe with the range*

subset = df.iloc[:, small_range]

print(subset.head())

```
In [65]: subset = df1.iloc[:, small_range]
         print(subset.head())

       country continent  year  lifeExp       pop
0  Afghanistan      Asia  1952   28.801   8425333
1  Afghanistan      Asia  1957   30.332   9240934
2  Afghanistan      Asia  1962   31.997  10267083
3  Afghanistan      Asia  1967   34.020  11537966
4  Afghanistan      Asia  1972   36.088  13079460
```

Subsetting Columns by Range

```
# create a range from 3 to 5 inclusive
 small_range = list(range(3, 6))
 print(small_range)
```

the values are specified in a way

such that the range is inclusive

on the left, and exclusive on the right

```
In [66]: small_range = list(range(3, 6))
         print(small_range)

         [3, 4, 5]
```

```
subset = df1.iloc[:, small_range]
 print(subset.head())
```

```
In [67]: subset = df1.iloc[:, small_range]
         print(subset.head())

            lifeExp       pop   gdpPercap
         0   28.801   8425333  779.445314
         1   30.332   9240934  820.853030
         2   31.997  10267083  853.100710
         3   34.020  11537966  836.197138
         4   36.088  13079460  739.981106
```

Subsetting Columns by Range

```
# create a range from 0 to 5 inclusive, every other integer
 small_range = list(range(0, 6, 2))
 subset = df.iloc[:, small_range]
 print(subset.head())
```

```
In [67]: subset = df1.iloc[:, small_range]
         print(subset.head())

            lifeExp       pop   gdpPercap
         0   28.801   8425333  779.445314
         1   30.332   9240934  820.853030
         2   31.997  10267083  853.100710
         3   34.020  11537966  836.197138
         4   36.088  13079460  739.981106
```

Subsetting Rows and Columns

- *# using loc*
  print(df.loc[42, 'country'])
- Angola
- *# using iloc*
  print(df.iloc[42, 0])
- Angola
- *# will cause an error*
  print(df.loc[42, 0])

```
In [72]: print(df1.loc[42, 'country'])
Angola

In [73]: print(df1.iloc[42, 0])
Angola
```

```
Traceback (most recent call last):
  File "<ipython-input-1-2b69d7150b5e>", line
2, in <module>
    print(df.loc[42, 0])
TypeError: cannot do label indexing on <class
'pandas.core.indexes.base.Index'> with these
indexers [0] of <class
```

Subsetting Multiple Rows and Columns

 # get the 1st, 100th, and 1000th rows
 # from the 1st, 4th, and 6th columns
 # the columns we are hoping to get are
 # country, lifeExp, and gdpPercap

print(df.iloc[[0, 99, 999], [0, 3, 5]])

```
In [74]: print(df1.iloc[[0, 99, 999], [0, 3, 5]])
```

	country	lifeExp	gdpPercap
0	Afghanistan	28.801	779.445314
99	Bangladesh	43.453	721.186086
999	Mongolia	51.253	1226.041130

 # if we use the column names directly,
 # it makes the code a bit easier to read
 # note now we have to use loc, instead of iloc

print(df.loc[[0, 99, 999], ['country', 'lifeExp', 'gdpPercap']])

```
print(df1.loc[[0, 99, 999], ['country', 'lifeExp', 'gdpPercap']])
```

	country	lifeExp	gdpPercap
0	Afghanistan	28.801	779.445314
99	Bangladesh	43.453	721.186086
999	Mongolia	51.253	1226.041130

using absolute indexes can lead to problems if the column order gets changed for some reason

Subsetting Multiple Rows and Columns

- **print**(df.loc[10:13, ['country', 'lifeExp', 'gdpPercap']])

```
In [76]: print(df1.loc[10:13, ['country', 'lifeExp', 'gdpPercap']])

       country  lifeExp    gdpPercap
10  Afghanistan   42.129   726.734055
11  Afghanistan   43.828   974.580338
12      Albania   55.230  1601.056136
13      Albania   59.280  1942.284244
```

Grouped and Aggregated Calculations

- print(df1.head(n=10))

```
In [77]: print(df1.head(n=10))
```

	country	continent	year	lifeExp	pop	gdpPercap
0	Afghanistan	Asia	1952	28.801	8425333	779.445314
1	Afghanistan	Asia	1957	30.332	9240934	820.853030
2	Afghanistan	Asia	1962	31.997	10267083	853.100710
3	Afghanistan	Asia	1967	34.020	11537966	836.197138
4	Afghanistan	Asia	1972	36.088	13079460	739.981106
5	Afghanistan	Asia	1977	38.438	14880372	786.113360
6	Afghanistan	Asia	1982	39.854	12881816	978.011439
7	Afghanistan	Asia	1987	40.822	13867957	852.395945
8	Afghanistan	Asia	1992	41.674	16317921	649.341395
9	Afghanistan	Asia	1997	41.763	22227415	635.341351

- 1. For each year in our data, what was the average life expectancy? What is the average life expectancy, population, and GDP?
- 2. What if we stratify the data by continent and perform the same calculations?
- 3. How many countries are listed in each continent?

To answer the questions just posed, we need to perform a grouped (i.e., aggregate) calculation.

Grouped Means

 # For each year in our data, what was the average life expectancy?

 # To answer this question,

 # we need to split our data into parts by year

 # then we get the 'lifeExp' column and calculate the mean

print(df.groupby('year')['lifeExp'].mean())

```
In [78]: print(df1.groupby('year')['lifeExp'].mean())

year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

Grouped Means

- *# the backslash allows us to break up 1 long line of Python code*
  # into multiple lines
  # df.groupby(['year', 'continent'])[['lifeExp', 'gdpPercap']].mean()
  # is the same as the following code
  multi_group_var = df.\
  groupby(['year', 'continent'])\
  [['lifeExp', 'gdpPercap']].\
  mean()
  print(multi_group_var)
- (OR)
- multi_group_var = df1.groupby(['year', 'continent'])[['lifeExp', 'gdpPercap']].mean()
- print(multi_group_var)

Grouped Means

year	continent	lifeExp	gdpPercap
1952	Africa	39.135500	1252.572466
	Americas	53.279840	4079.062552
	Asia	46.314394	5195.484004
	Europe	64.408500	5661.057435
	Oceania	69.255000	10298.085650
1957	Africa	41.266346	1385.236062
	Americas	55.960280	4616.043733
	Asia	49.318544	5787.732940
	Europe	66.703067	6963.012816
	Oceania	70.295000	11598.522455
1962	Africa	43.319442	1598.078825
	Americas	58.398760	4901.541870
	Asia	51.563223	5729.369625
	Europe	68.539233	8365.486814
	Oceania	71.085000	12696.452430
1967	Africa	45.334538	2050.363801
	Americas	60.410920	5668.253496
	Asia	54.663640	5971.173374
	Europe	69.737600	10143.823757
	Oceania	71.310000	14495.021790

Grouped Means

- If you need to “flatten” the dataframe, you can use the reset_index method.
- flat = multi_group_var.reset_index()
  print(flat.head(15))

```
In [85]: flat = multi_group_var.reset_index()
         print(flat.head(15))
```

	year	continent	lifeExp	gdpPercap
0	1952	Africa	39.135500	1252.572466
1	1952	Americas	53.279840	4079.062552
2	1952	Asia	46.314394	5195.484004
3	1952	Europe	64.408500	5661.057435
4	1952	Oceania	69.255000	10298.085650
5	1957	Africa	41.266346	1385.236062
6	1957	Americas	55.960280	4616.043733
7	1957	Asia	49.318544	5787.732940
8	1957	Europe	66.703067	6963.012816
9	1957	Oceania	70.295000	11598.522455
10	1962	Africa	43.319442	1598.078825
11	1962	Americas	58.398760	4901.541870
12	1962	Asia	51.563223	5729.369625
13	1962	Europe	68.539233	8365.486814
14	1962	Oceania	71.085000	12696.452430

Grouped Frequency Counts

- use the nunique and value_counts methods, respectively, to get counts of unique values and frequency counts on a Pandas Series.
- **print**(df1.groupby('continent')['country'].nunique())

```
print(df1.groupby('continent')['country'].nunique())
```

```
continent
Africa      52
Americas    25
Asia        33
Europe      30
Oceania      2
Name: country, dtype: int64
```

- print(df1.groupby('continent')['country'].value_counts())

```
In [88]: print(df1.groupby('continent')['country'].value_counts())
```

```
continent  country
Africa     Algeria           12
           Angola            12
           Benin             12
           Botswana          12
           Burkina Faso      12
                             ..
Europe     Switzerland       12
           Turkey            12
           United Kingdom    12
Oceania    Australia         12
           New Zealand       12
Name: country, Length: 142, dtype: int64
```

Basic Plot

- Visualizations are extremely important in almost every step of the data process. They help us identify trends in data when we are trying to understand and clean the data, and they help us convey our final findings.
- global_yearly_life_expectancy = df.groupby('year')['lifeExp'].mean()
- **print**(global_yearly_life_expectancy)
- global_yearly_life_expectancy.plot()

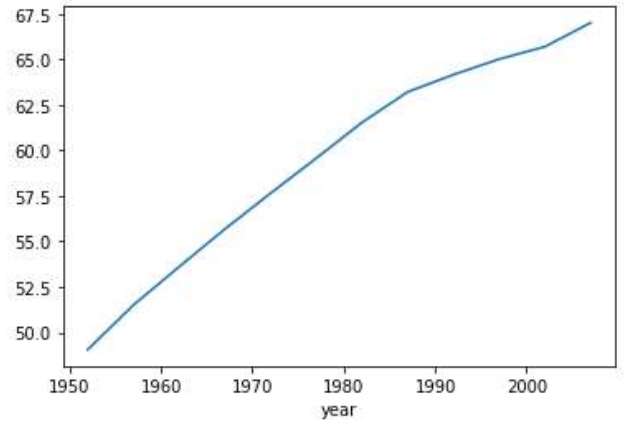
```
In [89]: global_yearly_life_expectancy = df1.groupby('year')['lifeExp'].mean()

In [90]: print(global_yearly_life_expectancy)

year
1952    49.057620
1957    51.507401
1962    53.609249
1967    55.678290
1972    57.647386
1977    59.570157
1982    61.533197
1987    63.212613
1992    64.160338
1997    65.014676
2002    65.694923
2007    67.007423
Name: lifeExp, dtype: float64
```

```
In [92]: global_yearly_life_expectancy.plot()

Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0x215cfde14c8>
```




ngiyabonga
danke
謝謝
teşekkür ederim
спасибо
thank you
gracias
dank je
tapadh leat
bedankt
dziękuję
mochchakkeram
go raibh maith agat
obrigado
sukriya
kop khun krap
arigatō
takk
dakujem
grazie
terima kasih
merci
நன்றி