Exercise 1: Interacting with the Python Shell Using the IPython Commands

Getting started with the Python shell is simple. Let's follow these steps to interact with the IPython shell:

1.To start the Python shell, type the ipython command in the console:

```
> ipython
In [1]:
```

The IPython shell is now ready and waiting for further commands. First, let's do a simple exercise to solve a sorting problem with one of the basic sorting methods, called straight insertion.

2. In the IPython shell, copy-paste the following code:

```
import numpy as np
vec = np.random.randint(0, 100, size=5)
print(vec)
```

Now, the output for the randomly generated numbers will be similar to the following:

```
[23, 66, 12, 54, 98, 3]
```

3. Use the following logic to print the elements of the vec array in ascending order:

```
for j in np.arange(1, vec.size):
v = vec[j]
i = j
while i > 0 and vec[i-1] > v:
vec[i] = vec[i-1]
i = i - 1
vec[i] = v
```

Use the print(vec) command to print the output on the console:

```
[3, 12, 23, 54, 66, 98]
```

4. Now modify the code. Instead of creating an array of 5 elements, change its parameters so it creates an array with 20 elements, using the up arrow to edit the pasted code. After changing the relevant section, use the down arrow to move to the end of the code and press Enter to execute it.

Exercise 2: Getting Started with the Jupyter Notebook

Let's execute the following steps to demonstrate how to start to execute simple programs in a Jupyter notebook. Working with a Jupyter notebook for the first time can be a little confusing, but let's try to explore its interface and functionality.

Now, start a Jupyter notebook server and work on it by following these steps:
1.To start the Jupyter notebook server, run the following command on the console:

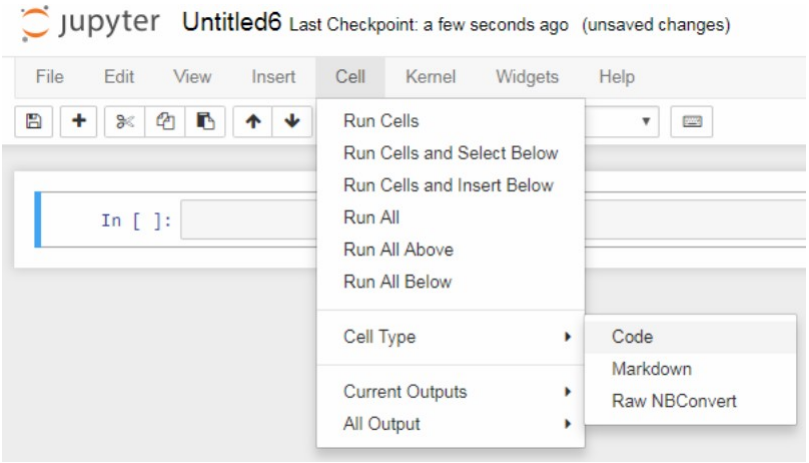
```
> jupyter notebook
```

2. After successfully running or installing Jupyter, open a browser window and navigate to **http://localhost:8888** to access the notebook.

3. You should see a notebook similar to the one shown in the following screenshot:

4. After that, from the top-right corner, click on New and select Python 3 from the list.

5. A new notebook should appear. The first input cell that appears is a Code cell. The default cell type is Code. You can change it via the Cell Type option located under the Cell menu:



6. Now, in the newly generated Code cell, add the following arithmetic function in the first cell:

```
In []: x = 2
print(x*2)
Out []: 4
```

7. Now, add a function that returns the arithmetic mean of two numbers, and then execute the cell:

```
In []: def mean(a,b):
return (a+b)/2
```

8. Let's now use the mean function and call the function with two values, 10 and 20. Execute this cell. What happens? The function is called, and the answer is printed:

```
In []: mean(10,20)
Out[]: 15.0
```

9. We need to document this function. Now, create a new Markdown cell and edit the text in the Markdown cell, documenting what the function does:

The mean function calculates the arithmetic mean of two values.

This is a text cell. It accepts Markdown, Latex and HTML.

Try to use HTML tags to format Markdown Cell. To see the effect you need to „run" it (Ctrl-Enter).

10. Then, include an image from the web. The idea is that the notebook is a document that should register all parts of analysis, so sometimes we need to include a diagram or graph from other sources to explain a point.

11. Now, finally, include the mathematical expression in LaTex in the same Markdown cell:

$$f(x) = \int_{-\infty}^{\infty} e^{-2ikx} f(k)$$

Sample LaTex (you must use $$): $ c = \sqrt{a^2 + b^2} $

Activity 1: IPython and Jupyter

Let's demonstrate common Python development in IPython and Jupyter. We will import NumPy, define a function, and iterate the results.

1. To import NumPy use:

```
import numpy as np
```

2. Create a function `squarePlus(x,c)`, which calculates value:

`y=x^2+c`

3. Run your code in IPython and Jupyter

Exercise 3: Reading Data with Pandas

How can an analyst start data analysis without data? We need to learn how to get data from an internet source into our notebook so that we can start our analysis. Let's demonstrate how pandas can read CSV data from an internet source so we can analyze it:
1. Import pandas library.

```
import pandas as pd
```

2. Read the Automobile mileage dataset: imports-85.data. Convert it to csv.
2. To read the CSV file we have to use **the read_csv()** function. It reads the file and returns Pandas DataFrame.

```
Sample code : df = pd.read_csv("/path/to/imports-85.csv", names = columns)
```

4. Use the function read_csv from pandas and show the first rows calling the method head on the DataFrame:

```
import pandas as pd
df = pd.read_csv("imports-85.csv")
df.head()
```

The output is as follows:

	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.60	...	130	mpfi	3.47	2.68	9.00	111	5000	21	27	13495
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
1	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
2	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
3	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450
4	2	?	audi	gas	std	two	sedan	fwd	front	99.8	...	136	mpfi	3.19	3.40	8.5	110	5500	19	25	15250

Pandas can read more formats:

- JSON
- Excel
- HTML
- HDF5
- Parquet (with PyArrow)
- SQL databases
- Google Big Query

Try to read other formats from pandas, such as Excel sheets.

Selecting Data

To select a column, just use the name of the column (projection):

```
df['State']
```
or
```
df.State
```

To select rows with given attribute value (selection):

```
df[df.State == "MN"]
```

To use logic operations, eg. select all rows that have State equal to AK and a Location of Nome, use the & operator:

```
df[(df.State == "AK") & (df.Location == "Nome")]
```

Selecting using rows range or index:

```
df.Location[2]
df.Location[1:3]
```

Display rows starting from first (first five are selected by default):

```
df.head()
df.head(10)
```

Get list of column names:

```
df.columns
```

Display columns datatypes:

```
df.dtypes
```

Exercise 4: Data Selection and the .loc Method

As we saw before, selecting data, separating variables, and viewing columns and rows of interest is fundamental to the analysis process. Let's say we want to analyze the radiation from I-131 in the state of Minnesota:

1.Import the NumPy and pandas libraries using the following command in the Jupyter notebook:

```
import numpy as np
import pandas as pd
```

2. Read the RadNet dataset from the EPA (U.S. Environmental Protection Agency), available from the Socrata project from file: `RadNet_Laboratory_Analysis.csv`

```
df = pd.read_csv('<PATH TO FILE>')
```

3. Start by selecting a column using the ['<name of the column>'] notation. Use the State column:

```
df['State'].head()
```

The output is as follows:

```
0    ID
1    ID
2    AK
3    AK
4    AK
Name: State, dtype: object
```

4. Now filter the selected values in a column using the MN column name:

```
df[df.State == "MN"]
```

The output is as follows:

	State	Location	Date Posted	Date Collected	Sample Type	Unit	Ba-140	Co-60	Cs-134	Cs-136	Cs-137	I-131	I-132	I-133
367	MN	St. Paul	04/08/2011	03/28/2011	Drinking Water	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect
368	MN	St. Paul	04/22/2011	04/13/2011	Drinking Water	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	0.16	Non-detect	Non-detect
380	MN	Welch	04/08/2011	03/29/2011	Drinking Water	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect
381	MN	Welch	06/01/2011	04/14/2011	Drinking Water	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect
555	MN	St. Paul	04/04/2011	03/22/2011	Precipitation	pCi/l	Non-detect	Non-detect	Non-detect	NaN	Non-detect	32.3	Non-detect	Non-detect
556	MN	St. Paul	04/10/2011	03/29/2011	Precipitation	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	16	Non-detect	Non-detect
557	MN	Welch	04/04/2011	03/17/2011	Precipitation	pCi/l	Non-detect	Non-detect	Non-detect	NaN	Non-detect	Non-detect	Non-detect	Non-detect
558	MN	Welch/510	04/13/2011	04/04/2011	Precipitation	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	9.1	Non-detect	Non-detect

5. Select more than one column per condition. Add the Sample Type column for filtering:

```
df[(df.State == 'CA') & (df['Sample Type'] == 'Drinking Water')]
```

The output is as follows:

	State	Location	Date Posted	Date Collected	Sample Type	Unit	Ba-140	Co-60	Cs-134	Cs-136	Cs-137	I-131	I-132	I-133	Te-129
305	CA	Los Angeles	04/10/2011	04/04/2011	Drinking Water	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	0.39	Non-detect	Non-detect	Non-detec
306	CA	Los Angeles	06/01/2011	04/12/2011	Drinking Water	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	0.18	Non-detect	Non-detect	Non-detec
356	CA	Richmond	04/09/2011	03/29/2011	Drinking Water	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detec
357	CA	Richmond	06/01/2011	04/13/2011	Drinking Water	pCi/l	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detect	Non-detec

6. Next, select the MN state and the isotope I-131:

```
df[(df.State == "MN") ]["I-131"]
```

The output is as follows (The radiation in the state of Minnesota with **ID 555** is the highest.):

```
367    Non-detect
368          0.16
380    Non-detect
381    Non-detect
555          32.3
556            16
557    Non-detect
558           9.1
Name: I-131, dtype: object
```

7. We can do the same more easily with the `.loc` method, filtering by state and selecting a column on the same .loc call:

```
df_rad=df
df_rad.loc[df_rad.State == "MN", "I-131"]
```

Note
The result of the .loc filter is a series and not a DataFrame. This depends on the operation and selection done on the DataFrame and not is caused only by .loc . Because the DataFrame can be understood as a 2D combination of series, the selection of one column will return a series. To make a selection and still return a DataFrame, use double brackets e.g.:

```
df[['I-132']].head()
```

Applying a Function to a Column

Data is never clean. There are always cleaning tasks that have to be done before a dataset can be analyzed. One of the most common tasks in data cleaning is applying a function to a column, changing a value to a more adequate one. In our example dataset, when no concentration was measured, the non-detect value was inserted. As this column is a numerical one, analyzing it could become complicated. We can apply a transformation over a column, changing from non-detect to numpy.NaN, which makes manipulating numerical values more easy, filling with other values such as the mean, and so on.

To apply a function to more than one column, use the applymap method, with the same logic as the apply method. For example, another common operation is removing spaces from strings. Again, we can use the apply and applymap functions to fix the data. We can also apply a function to rows instead of to columns, using the axis parameter (0 for rows, 1 for columns).

`apply()` method syntax sample (for all rows):
df – Pandas Dataframe

```
df['column_name'].apply(<lambda function>)
```

applymap() method syntax sample (for all rows):

```
df.loc[:, <[list_of_column_names]>]=df.loc[:,
[<list_of_column_names>]].applymap(<lambda function>)
```

Activity 2: Working with Data Problems

Before starting an analysis, we need to check for data problems, and when we find them (which is very common!), we have to correct the issues by transforming the DataFrame. One way to do that, for instance, is by applying a function to a column, or to the entire DataFrame. It's common for some numbers in a DataFrame, when it's read, to not be converted correctly to floating-point numbers. Let's fix this issue by applying functions:

1. Import pandas and numpy library.
2. Read the RadNet dataset from the U.S. Environmental Protection Agency.
3. Create a list with numeric columns for radionuclides in the RadNet dataset.
4. Use the apply method on one column, with a lambda function that compares the Non-detect string
5. Use applymap method to replace text values "Non-detect" with .np.nan in all numeric columns at once, do the same with "ND" value

6. Use the same applymap method to remove all spaces (from begining and end) in all other columns (non-numeric). Use strip() method to remove spaces.

Additionally:

7. Change datatypes of columns using methods (use df.dtypes() to check datatypes):

```
pd.to_datetime()
pd.to_numeric()
```