

Machine Learning Basics Tutorials

Python Basics, and Introduction to Pandas

Contents

1	Acknowledgements	2
2	Introduction	2
2.1	Python and Matlab	2
2.2	The Pandas Library	2
3	Python and Analogous Matlab Commands	3
3.1	Python and Matlab	3
3.2	Basic Arrays and Vectors	3
3.3	Matrix Operations	11
4	Pandas	18
4.1	Pandas Basics	18
4.2	Data Extraction and Manipulation	19
4.3	Column Manipulation and Iteration	21
4.4	Exporting as Separate File	22
5	Concluding Remarks	23

1 Acknowledgements

This tutorial for Machine Learning basics has been created by Danyal Saqib and Talha Mehboob under the supervision of Dr. Wajahat Hussain for the School of Electrical Engineering and Computer Sciences (SEECS) at the National University of Sciences and Technology, NUST. Reporting any errors or omissions within this text shall be appreciated.

2 Introduction

This main purpose of this whole series is to take you through the basics of some of the most popular programming packages used in Python for various tasks. This particular text first introduces some common Python commands with their corresponding Matlab analogies, and then goes on to give a brief overview and tutorial for the Pandas library.

2.1 Python and Matlab

Many people, especially from Engineering backgrounds, are familiar with Matlab already. However, learning some common Python commands can initially be a daunting task. Hence, the first section of this texts focuses on some common Matlab operations, and their analogous commands in Python, thus creating a bridge for ease of transition. We will give Matlab commands that are commonly used, and then show how those same results can be obtained in Python.

2.2 The Pandas Library

Pandas is a library for Python specifically used for data manipulation and analysis. It has easy-to-use data structures and data analysis tools. This works especially well when dealing with data forms that are stored in, let's say, spreadsheets or arrays. This makes Pandas an indispensable tool for data analysis, and invaluable for Machine Learning engineers. Thus, the second major section of this text focuses on the Pandas library and how it can be used for data manipulation, making it much more powerful than traditional spreadsheet management softwares.

3 Python and Analogous Matlab Commands

3.1 Python and Matlab

Matlab is a programming language that is familiar for many Engineers. Thus, this section aims to familiarize you with Python by providing you with an analogy with Matlab. This will help you with the transition from Matlab to Python programming. Note that we will use NumPy library for these coding exercises, and that a tutorial for NumPy has already been provided in the previous text document.

3.2 Basic Arrays and Vectors

We will first see how a simple array is created.

Matlab Example 1:

```
1 % creating vectors
2 x = 1:7;
3 display(x)
```

Output

```
x =
     1     2     3     4     5     6     7
```

Python Example 1:

```
1 import numpy as np
2
3 x = np.arange(1, 8)
4 print(x)
```

Output

```
[1 2 3 4 5 6 7]
```

Note how in Matlab, the last element is included in the array range, whereas in Python it is not.

Matlab Example 2:

```
1 x = [1,2,3,4,5,6,7];  
2 display(x)
```

Output

```
x =  
  
    1    2    3    4    5    6    7
```

Python Example 2:

```
1 x = np.array([1, 2, 3, 4, 5, 6, 7])  
2 print(x)
```

Output

```
[1 2 3 4 5 6 7]
```

Again, we see that the ‘np.array()’ function helps us in creating the same arrays as we did in Matlab.

Matlab Example 3:

```
1 x = 1:2:10;  
2 display(x)
```

Output

```
x =  
  
    1    3    5    7    9
```

Python Example 3:

```
1 x = np.arange(1, 11, 2)  
2 print(x)
```

Output

```
[1 3 5 7 9]
```

There are again 2 important things to note here. Firstly, we see once again that in the ‘np.arange()’ function, the end of the range is not included in the actual array, whereas in Matlab, the ending range is included in the array. Secondly, in Matlab, the order of arguments is ‘start limit, step size, end limit’, whereas for the ‘np.arange()’ function it is ‘start limit, end limit, step size’.

Matlab Example 4:

```
1 x = 10:-2:1
2 display(x)
```

Output

```
x =
    10     8     6     4     2
```

Python Example 4:

```
1 x = np.arange(10, 0, -2)
2 print(x)
```

Output

```
[10  8  6  4  2]
```

Again, the same minor syntax differences between Matlab and Python are very easy to understand.

Matlab Example 5:

```
1 x = 2:7;
2 display(x)
3 y = x(1);
4 display(y)
```

Output

```
x =
    2    3    4    5    6    7

y = 2
```

Python Example 5:

```
1 x = np.arange(2, 8)
2 print('x: ', x)
3 y = x[0]
4 print('y: ', y)
```

Output

```
x:  [2  3  4  5  6  7]
y:  2
```

Here, the most important difference to note is that in Matlab, array indexing starts from 1. However, in Python, array indexing starts from 0.

Matlab Example 6:

```
1 x = 2:7;
2 display(x)
3 y = x(end-2 : end);
4 display(y)
```

Output

```
x =
    2    3    4    5    6    7

y =
    5    6    7
```

Python Example 6:

In Python, instead of the ‘end’ keyword, we can simply use ‘len(x)-1’.

```

1 x = np.arange(2, 8)
2 print('x: ', x)
3
4 # Defining the end index for x
5 x_end = len(x) - 1
6
7 y = x[x_end-2 : x_end+1]
8 print('y: ', y)

```

Output

```

x:  [2  3  4  5  6  7]
y:  [5  6  7]

```

Note that the ‘x_end’ variable works just like the ‘end’ keyword in Matlab. However, for the ending limit, we need to define it as ‘x_end + 1’ as we have already seen, in Python the ending limit is not included in the actual array itself.

Matlab Example 7:

```

1 x = 2:7;
2 display(x)
3 x(2)=[];
4 display(x)

```

Output

```

x =
     2     3     4     5     6     7

x =
     2     4     5     6     7

```

Python Example 7:

```

1 x = np.arange(2, 8)
2 print('x: ', x)
3 x = np.delete(x, 1)
4 print('x: ', x)

```

Output

```
x:  [2  3  4  5  6  7]
x:  [2  4  5  6  7]
```

Here, the ‘np.delete’ function is easily used. It takes an array and an index as in input, and outputs that same array, but with that index value removed.

Matlab Example 8:

```
1 x = 2:7;
2 display(x)
3 x=[50, x];
4 display(x)
5 x=[x, 100];
6 display(x)
```

Output

```
x =
     2     3     4     5     6     7

x =
    50     2     3     4     5     6     7

x =
    50     2     3     4     5     6     7    100
```

Python Example 8:

```
1 x = np.arange(2, 8)
2 print('x: ', x)
3 x = np.insert(x, 0, 50)
4 print('x: ', x)
5 x = np.insert(x, len(x), 100)
6 print('x: ', x)
```


Output

```

x:  [2  3  4  5  6  7]
x:  [50  2  3  4  5  6  7]
x:  [ 50  2  3  4  5  6  7 100]

```

The ‘np.insert()’ function takes in 3 primary arguments in this very order: the array to be modified, the specified index, and the value to be inserted. The output is that same array, but with the value inserted **before** the specified index.

So for example in the first modification, we wanted to insert the value of 50 at the very start of the array. Hence, we specified the index to be 0. Thus, the value of 50 was inserted before the first element. Similarly, we wanted to insert the value of 100 at the very end of the array. If we had specified the last index of the array for this i.e ‘len(x) - 1’, the value would have been inserted before the last element, and would have been the second last element. Hence, we have to specify the index to be ‘len(x)’.

Matlab Example 9:

```

1 x = 2:7;
2 display(x)
3 y = [x;x];
4 display(y)

```

Output

```

x =

     2     3     4     5     6     7

y =

     2     3     4     5     6     7
     2     3     4     5     6     7

```

Python Example 9:

```

1 x = np.arange(2, 8)
2 print('x: ', x)
3 y = np.array([x, x])
4 print('y: ', y)

```

Output

```
x:  [2 3 4 5 6 7]
y:  [[2 3 4 5 6 7]
     [2 3 4 5 6 7]]
```

In this example, the Matlab and Python analog is almost identical. We just have to specify that ‘y’ is a NumPy array.

3.3 Matrix Operations

Now we will see how to perform some important matrix functions in Python, with analogous Matlab code.

Matlab Matrix Example 1:

```
1 x = [1, 2, 3; ...  
2     7, 8, 9; ...  
3     10, 11, 12];  
4 display(x)
```

Output

```
x =  
  
     1     2     3  
     7     8     9  
    10    11    12
```

Python Matrix Example 1:

```
1 import numpy as np  
2  
3 a = np.array([[1, 2, 3],  
4              [7, 8, 9],  
5              [10, 11, 12]])  
6  
7 print(a)
```

Output

```
[[ 1  2  3]  
 [ 7  8  9]  
 [10 11 12]]
```

The simple 'np.array' command helps in creation of multi-dimensional arrays.

Matlab Matrix Example 2:

```
1 display(x(1,:))  
2 display(x(:,1))  
3 display(x(1:2,1:2))  
4 display(x(1:2:end,1:2:end))
```

```

5 display(x(:))
6 display(x(4))
7 display(x([1,3],[1 3]))

```

Output

```

1      2      3
      1
      7
     10

1      2
7      8

1      3
10     12

1
7
10
2
8
11
3
9
12

2

1      3
10     12

```

Python Matrix Example 2:

```

1 print(a[0,:])
2 print(a[:,0])
3 print(a[0:2,0:2])
4 print(a[0:4:2,0:4:2])
5 print(a[:])
6 print(a[0,1])
7 end = len(a) - 1
8 print(a[0:end+1:2, 0:end+1:2])

```

Output

```

[1 2 3]

[ 1  7 10]

[[1 2]
 [7 8]]

[[ 1  3]
 [10 12]]

[[ 1  2  3]
 [ 7  8  9]
 [10 11 12]]

2

[[ 1  3]
 [10 12]]

```

Again we are using indexing techniques to reproduce the same results in Python.

Matlab Matrix Example 3:

```

1 y = x>5;
2 display(y)

```

Output

```

display(y)
y =

    0    0    0
    1    1    1
    1    1    1

```

Python Matrix Example 3:

```

1 y = a>5;
2 y = y.astype(np.int)
3 print(y)

```

Output

```
[[0 0 0]
 [1 1 1]
 [1 1 1]]
```

Again, a vet simple analogy between Matlab and Python.

Matlab Matrix Example 4:

```
1 z = x(y);
2 display(z)
```

Output

```
z =

     7
    10
     8
    11
     9
    12
```

Python Matrix Example 4:

```
1 z = a[a > 5]
2 print(z)
```

Output

```
[ 7  8  9 10 11 12]
```

Here again we see that the major differences are in syntax, not the logic.

Matlab Matrix Example 5:

```
1 x = randn(2,2,3);
2 display(x)
```

Output

```

x =

ans(:,:,1) =

    -0.1726    -0.3053
    -2.5119     0.8582

ans(:,:,2) =

    -1.6453     0.7108
     1.1670     0.6831

ans(:,:,3) =

    -1.1788     0.5762
     0.7886    -1.1383

```

Python Matrix Example 5:

```

1 x = np.random.rand(3, 2, 2)
2 print (x)

```

Output

```

[[[0.48822613  0.6558497 ]
  [0.1808548  0.78641941]]

 [[0.9067166  0.54323997]
  [0.67226966 0.3516655 ]]

 [[0.26359573 0.50528116]
  [0.51013627 0.4784581 ]]]

```

The ‘np.random.rand()’ function helps us in creating a multi-dimensional array, initialized with random integers.

Matlab Matrix Example 6:

```

1 x(:,:,1) = [1 2; 3 4];
2 x(:,:,2) = [5 6; 7 8];
3 x(:,:,3) = [9 10;11 12];

```

```

4 x(:,:,4) = [13 14;15 16];
5
6 display(x(:,:,1))
7 display(x(:,2,:))
8 display(size(x(:,2,:)))
9 display(x(:,:,1))

```

Output

```

      1      2
      3      4

ans(:,:,1) =
      2
      4
ans(:,:,2) =
      6
      8
ans(:,:,3) =
     10
     12
ans(:,:,4) =
     14
     16

      2      1      4

      1      2
      3      4

```

Python Matrix Example 6:

```

1 x[0,:,:] = [[1, 2],
2             [3, 4]]
3 x[1,:,:] = [[5, 6],
4             [7, 8]]
5 x[2,:,:] = [[9, 10],
6             [11, 12]]
7
8 print(x[0,:,:])
9 print(x[:, :, 1])
10 print(x[:, :, 1].shape)

```


Output

```
[[1.  2.]  
 [3.  4.]]  
  
[[ 2.  4.]  
 [ 6.  8.]  
 [10. 12.]]  
  
(3, 2)
```

Here, the important thing to note is the order for indexing. In Matlab, for a 3-D Matrix, the dimension is specified as the last argument, whereas in Python, the dimension is specified as the first argument.

This brings us to the end of this particular section. For those of you more familiar with Matlab, this section will hopefully help you in better understanding Python and its programming practices.

In the next section we will take a look at the Pandas library, and how it can be an incredibly powerful tool for data manipulation.

4 Pandas

4.1 Pandas Basics

The very first thing to do is to download the dataset we are going to use in this section. Go ahead and click the link below:

https://drive.google.com/file/d/1WGlAHA0joY-vQq9jmd_LXJxkxCg0JYuU/view?usp=sharing

After downloading the ‘test_data.csv’ file, you can open it on your computer using excel (or some other spreadsheet program) to inspect it.

	A	B	C	D	E	F
1	Student Name	Student ID	Math Score	Physics Score	Chemistry Score	English Score
2	Elin Cotton	31	45	67	54	57
3	Rafael Blake	90	76	58	61	71
4	Fahim Hartley	86	51	56	48	54
5	Gabrielle Khan	70	86	56	44	79
6	Sofia Briggs	48	60	82	47	82
7	Siya Walls	88	54	66	100	43
8	Iman Padilla	29	86	81	93	54
9	Emily Marks	87	53	60	59	81
10	Conor Ferrell	20	69	62	87	93
11	Christopher George	12	46	46	63	91
12	Linda William	85	47	61	94	49
13	Ruqayyah Gallegos	48	43	56	76	71

Figure 1: test_data.csv

Now, try and save this data into the same directory that you have your python file in. For Google Colab, you can upload this into your Google Drive, and access it from there.

In Pandas, the most commonly used data structure is a dataframe. It is somewhat similar to a NumPy array in structure, but Pandas allows us to do lots of different types of data manipulation that wouldn’t otherwise be possible. We will now import Pandas, and read this file into a dataframe object:

Code: pandasBasic.py

```
1 # Import Pandas using its common nickname
2 import pandas as pd
3
4 # Reading test_data.csv as a dataframe
5 df = pd.read_csv('test_data.csv')
6 print(df)
```

The dataframe object should be displayed.

4.2 Data Extraction and Manipulation

Now, let us look at some common data extraction and manipulation functions in Pandas:

Code: dataExtraction.py

```
1 # Import Pandas using its common nickname
2 import pandas as pd
3
4 # Reading test_data.csv as a dataframe
5 df = pd.read_csv('test_data.csv')
6 # Reading Headings
7 print("Headings/Column Names: \n", df.columns)
8 # Accessing a specific column
9 print("\nNames: \n", df['Student Name'])
10 # Accessing a specific row
11 print("\nStudent 2 Info: \n", df.iloc[1])
12 # Accessing a specific item
13 print("\nStudent 10 Physics score: \n", df.iloc[9, 3])
14 # Finding students whose Math scores are over 80
15 mth = df.loc[df['Math Score'] > 80]
16 # Printing only their names, Student IDs, and Math scores
17 print("\nStudent with Math scores greater than 80: \n", mth[['Student Name',
    'Student ID', 'Math Score']])
```

Output

```
Headings/Column Names:
Index(['Student Name', 'Student ID', 'Math Score',
      'Physics Score', 'Chemistry Score', 'English Score'],
      dtype='object')
Names:
0          Elin Cotton
1        Rafael Blake
2        Fahim Hartley
3    Gabrielle Khan
4        Sofia Briggs
5        Siya Walls
6        Iman Padilla
7        Emily Marks
8        Conor Ferrell
9    Christopher George
10       Linda William
11    Ruqayyah Gallegos
```

```

12      Eryn Snyder
13      Nancie Walters
14      Sebastian Waters
15      Fionn Jacobs
16      Ronan Ferry
17      Theodore Newton
18      Ashlea William
19      Codie Novak

Student 2 Info:
  Student Name      Rafael Blake
Student ID              90
Math Score             76
Physics Score          58
Chemistry Score        61
English Score          71

Student 10 Physics score:
46

Student with Math scores greater than 80:
  Student Name  Student ID  Math Score
3  Gabrielle Khan         70         86
6    Iman Padilla         29         86
13  Nancie Walters         31         89
15    Fionn Jacobs         80         86
16    Ronan Ferry          11         92
17  Theodore Newton         29         90
18  Ashlea William         42         98
19    Codie Novak          15         86

```

We can get a lot of useful information from this dataset using the command `df.describe()`, such as the mean, standard deviation, various quartiles, etc. You are encouraged to try it out on this dataset.

We can easily sort our data according to some parameter of our choice in Pandas:

Code: sortData.py

```

1 # Import Pandas using its common nickname
2 import pandas as pd
3
4 # Reading test_data.csv as a dataframe
5 df = pd.read_csv('test_data.csv')

```

```

6 # Find useful characteristics about your data
7 eng = df.sort_values('English Score')
8 # Only print their Names, IDs, and English scores
9 print(eng[['Student Name', 'Student ID', 'English Score']])

```

Output

	Student Name	Student ID	English Score
18	Ashlea William	42	40
5	Siya Walls	88	43
10	Linda William	85	49
19	Codie Novak	15	54
2	Fahim Hartley	86	54
6	Iman Padilla	29	54
0	Elin Cotton	31	57
16	Ronan Ferry	11	61
13	Nancie Walters	31	69
15	Fionn Jacobs	80	70
1	Rafael Blake	90	71
11	Ruqayyah Gallegos	48	71
3	Gabrielle Khan	70	79
7	Emily Marks	87	81
4	Sofia Briggs	48	82
14	Sebastian Waters	40	86
17	Theodore Newton	29	89
9	Christopher George	12	91
8	Conor Ferrell	20	93
12	Eryn Snyder	58	96

You can set the 'ascending' parameter of 'df.sort_values' to false, and get a descending list.

4.3 Column Manipulation and Iteration

We are now going to add another column to our dataframe, that shows average scores for each student:

Code: colManipulation.py

```

1 # Import Pandas using its common nickname
2 import pandas as pd
3
4 # Reading test_data.csv as a dataframe
5 df = pd.read_csv('test_data.csv')
6

```

```

7 # Adding another column for average scores
8 df['Average Score'] = (df['Math Score'] + df['Physics Score'] + df['Chemistry
   Score'] + df['English Score']) / 4
9 print(df[['Student Name', 'Average Score']])

```

Output

	Student Name	Average Score
0	Elin Cotton	55.75
1	Rafael Blake	66.50
2	Fahim Hartley	52.25
3	Gabrielle Khan	66.25
4	Sofia Briggs	67.75
5	Siya Walls	65.75
6	Iman Padilla	78.50
7	Emily Marks	63.25
8	Conor Ferrell	77.75
9	Christopher George	61.50
10	Linda William	62.75
11	Ruqayyah Gallegos	61.50
12	Eryn Snyder	72.25
13	Nancie Walters	67.50
14	Sebastian Waters	64.00
15	Fionn Jacobs	70.50
16	Ronan Ferry	74.50
17	Theodore Newton	80.25
18	Ashlea William	77.25
19	Codie Novak	70.25

Interestingly, we didn't need to manually type out every single column to be added to the 'Average Score' column. We could use a shorter method to average the scores:

```

1 # Short Method
2 df['Average Score'] = (df.iloc[:, 2:6].sum(axis=1)) / 4

```

Try running this code, and see if you can figure out how this line of code is equivalent to the previous one.

4.4 Exporting as Separate File

Finally, let's learn how to export this modified dataframe as it's own csv file:

Code: exportFile.py

```
1 # Import Pandas using its common nickname
2 import pandas as pd
3
4 # Reading test_data.csv as a dataframe
5 df = pd.read_csv('test_data.csv')
6
7 # Adding another column for average scores
8 df['Average Score'] = (df['Math Score'] + df['Physics Score'] + df['Chemistry
   Score'] + df['English Score']) / 4
9 # Short Method
10 df['Short Method Average'] = (df.iloc[:, 2:6].sum(axis=1)) / 4
11
12 # Exporting into a separate csv file
13 df.to_csv('modified.csv', index = False)
```

And indeed, you will have created a new csv file, with the new columns added in. Remember that they will be located in the same location as your Python file.

5 Concluding Remarks

This brings us to the end of this particular Machine Learning basics tutorial. They will hopefully give you a strong starting point in getting comfortable with Python in general and the Pandas Library in particular. These tutorial documents combined with the assignments will hopefully be a very effective and valuable learning experience for you.