

# Programming for Data Analytics

## Week8: Introduction to Pandas

Dr. Haithem Afli

[Haithem. afli@cit.ie](mailto:Haithem.afli@cit.ie)

@AfliHaithem

2018/2019



## → Introduction to Pandas Series and Dataframe data structures.

- Reading data into a Dataframe
- Accessing Data from a Dataframe
- Merging and Grouping Data

# Pandas

- NumPy is a great tool for dealing with **numeric matrices** and vectors in Python
  - For more complex data, such as tables it is limited.
- Fortunately, when dealing with complex data we can use the **Python Data Analysis Library** (a.k.a. pandas).
- Pandas is an open source library providing high-performance, easy-to-use data structures for the Python programming language.
  - Used primarily for data manipulation and analysis.
- Resources
  - <http://pandas.pydata.org/pandas-docs/version/0.13.1/pandas.pdf>

# Data Structures in Pandas

- Pandas introduces two new data structures to Python –
  - **Series**
  - **DataFrame**
- Both of which are built on top of NumPy (which means it's very fast).
- A **Series** is a one-dimensional object similar to an array, list, or column in a table.
- Pandas will assign a **labelled index** to each item in the Series.
  - By default, each item will receive an index label from 0 to N, where N is the length of the Series minus one.
- **S = Series(data, index = index)**
  - The data can be many different things such as a NumPy arrays, list of scalar values, dictionary

# Series - Examples

```
import numpy as np
import pandas as pd

s1 = pd.Series(np.random.randn(5))
s2 = pd.Series(np.random.randn(5), index=['a','b','c','d','e'])
# number of indices must match number of data points

print (s1)
print (s2)
```

```
import pandas as pd

# Dictionary with annual car robberies in each Irish city
d = {'Dublin': 245, 'Cork': 150, 'Limerick': 125, 'Galway': 360,
'Belfast': 300}
```

```
# if you pass a dictionary to a series, the keys becomes the
indexes of the Series
cities = pd.Series(d)

print (cities)
```

```
0    0.275735
1   -0.445412
2    0.163060
3   -0.364863
4   -0.069800
dtype: float64
```

```
a    0.068250
b    0.455478
c    1.356175
d    0.484393
e   -0.919080
dtype: float64
```

```
Belfast    300
Cork        150
Dublin      245
Galway      360
Limerick    125
dtype: int64
```

# Series

- You can use the **index** to select specific items from the Series.
  - The first print will print the entire series
  - The second will print the item associated with index 'b' (note you can access one item at time using this method)
  - The third uses **double square brackets** and prints a subset of the original series (**note it returns a independent Series object**)

```
s1 = pd.Series([1, 2, 3, 4, 5], index=['a','b','c','d','e'])
```

```
print (s1)
```

```
print (s1['b'])
```

```
print (s1[['a', 'b']])
```

```
a    1  
b    2  
c    3  
d    4  
e    5  
dtype: int64  
  
2  
  
a    1  
b    2  
dtype: int64
```

# Series

- Another useful feature of a series is using boolean conditions
  - irishCities <200** returns a Series of True/False values, which we then pass to our Series cities, returning the corresponding True items.

```
# Dictionary with annual car robberies in each Irish city  
d = {'Dublin': 245, 'Cork': 150, 'Limerick': 125, 'Galway':  
360, 'Belfast': 300}
```

```
irishCities = pd.Series(d)
```

```
print (irishCities[ irishCities <200 ])
```

```
print (type(irishCities[irishCities <200]))
```

As with NumPy, relational operators return a **separate copy** of the data. The original series and the one returned by the relational operator don't refer to the same copy of the same data.

```
Cork    150  
Limerick 125  
dtype: int64  
<class 'pandas.core.series.Series'>
```

# Series

- It is also very easy to change a value within a series.

```
d = {'Dublin': 245, 'Cork': 150, 'Limerick': 125, 'Galway': 360, 'Belfast': 300}
```

```
irishCities = pd.Series(d)
```

```
print (irishCities)
```

```
irishCities["Cork"] = 180
```

```
irishCities["Kilkenny"] = 120
```

```
print (irishCities)
```

Similar to the syntax we use for adding a key value pair to a dictionary.

```
Galway    360  
Belfast   300  
Cork      150  
Dublin    245  
Limerick   125  
dtype: int64
```

```
Galway    360  
Belfast   300  
Cork    180  
Dublin    245  
Limerick   125  
Kilkenny 120  
dtype: int64
```

# Series



- What does the code below achieve?

```
d = {'Dublin': 245, 'Cork': 150, 'Limerick': 125, 'Galway': 360, 'Belfast': 300}
```

```
irishCities = pd.Series(d)
```

```
print (irishCities)
```

```
irishCities[irishCities<160] = 100
```

```
print (irishCities)
```

This code will go through the Series setting any value that is currently less than 160 to a value of 100.

When you use Boolean selection coupled with assignment it selects the entries in the existing Series object to be changed.

```
Galway    360
Belfast   300
Cork      150
Dublin    245
Limerick   125
dtype: int64
```

```
Galway    360
Belfast   300
Cork      100
Dublin    245
Limerick   100
dtype: int64
```

# Series

- Normal NumPy mathematical operations can be completed on Series objects as well.

```
d = {'Dublin': 245, 'Cork': 150, 'Limerick': 125, 'Galway': 360, 'Belfast':  
300}  
irishCities = pd.Series(d)  
  
print (irishCities*100)  
  
print (np.square(irishCities))
```

Notice in this example we still use NumPys square method but rather than passing it a NumPy array we pass it a Series instead

```
Belfast    30000  
Cork       15000  
Dublin     24500  
Galway     36000  
Limerick   12500  
dtype: int64
```

```
Belfast    90000  
Cork       22500  
Dublin     60025  
Galway     129600  
Limerick   15625  
dtype: int64
```

# Series – len and unique function

- As with all data structures we have seen so far we can use the ***len()*** function to obtain the number of values stored in a Series (this also works for a dataframe, which return the number of rows)
- Another useful function to use with a Series object is the ***unique*** function, which returns all the unique data items in a specific series object (it is returned as a NumPy array).

```
import pandas as pd
```

```
seriesA = pd.Series(['A', 'C', 'B', 'B', 'A'])
```

```
['A' 'C' 'B']
```

```
print (pd.unique(seriesA))
```

# Example

- Create a Pandas Series variable to store the data depicted in the table below (we will use the names as indices and the grades as the values).
- Write code that will return a Series containing all those that failed the exam.
- Next write code that will increase any grade less than 40 by 5%

Name	Grade
Jim	78
Elaine	23
Ted	65
Frank	88
Sarah	80
Tim	33

# Example

```
import pandas as pd
```

```
studentDetails = {'Jim':78, 'Elaine':23, 'Ted':65, 'Frank':88, 'Sarah':80, 'Tim':33}  
grades = pd.Series(studentDetails)
```

```
print grades[grades<40]
```

```
grades[grades<40] += 5
```

```
print (grades)
```

```
Elaine    23  
Tim      33  
dtype: int64
```

```
Elaine    28  
Frank    88  
Jim      78  
Sarah    80  
Ted      65  
Tim      38  
dtype: int64
```



It is possible to turn this Series into a one-column DataFrame with the `to_frame` method.

This method will use the Series name as the new column name:

```
>>> director.to_frame()
```

# Discussion



# Data Frame

- A DataFrame is a data structure comprised of **rows and columns** of data.
  - It is similar to a spreadsheet or a database table.
  - You can also think of a DataFrame as a collection of Series objects that share an index
- To create a DataFrame out of common Python data structures, we can pass a dictionary of lists to the DataFrame constructor.
- We can also easily create a dataframe by passing it a 2D NumPy array.
- The syntax for creating a data frame is as follows:
  - ***DataFrame(data, columns=listOfColumns)***
- Using the columns parameter allows us to tell the constructor how we'd like the columns ordered.

# Creating a DataFrame



```
import pandas as pd\n\ndata = {'student': ['Jim Murphy', 'Ted Scully', 'Jason Oakley', 'Pat O'Brien'],\n        'grade': [67, 75, 56, 89],\n        'department': ["Computing", "Chemistry", "Biology", "Maths"]}\n\nstudents = pd.DataFrame(data)\n\nprint students
```

```
department  grade  student\n0 Computing    67 Jim Murphy\n1 Chemistry    75 Ted Scully\n2 Biology      56 Jason Oakley\n3 Maths        89 Pat O'Brien
```

Notice the key becomes the columns headers of the dataframe and the values of the dictionary (the list) populate the column.

# Creating a DataFrame

```
import pandas as pd
data = {'student': ['Jim Murphy', 'Ted Scully', 'Jason Oakley', 'Pat O'Brien'],
        'grade': [67, 75, 56, 89],
        'department': ["Computing", "Chemistry", "Biology", "Maths"]}
students = pd.DataFrame(data, columns=['student', 'grade', 'department'])
print students
```

	student	grade	department
0	Jim Murphy	67	Computing
1	Ted Scully	75	Chemistry
2	Jason Oakley	56	Biology
3	Pat O'Brien	89	Maths

I can directly specify the names of the columns and the order in which they appear by including a columns argument when creating the dataframe. It is important that the names of the columns match the dictionary keys

# Creating a DataFrame

- Rather than using a list as we did in the previous slide we can also create a dataframe by passing a dictionary of Series objects.

```
seriesA = pd.Series(np.random.rand(3), index=['a', 'b', 'c'])  
seriesB = pd.Series(np.random.rand(4), index=['a', 'b', 'c', 'd'])  
seriesC = pd.Series(np.random.rand(3), index=['b', 'c', 'd'])
```

```
df = pd.DataFrame({'one' : seriesA,  
                   'two' : seriesB,  
                   'three' : seriesC})
```

```
print df
```

	one	three	two
a	0.307010	<b>NaN</b>	0.396005
b	0.671142	0.263916	0.532836
c	0.116057	0.839463	0.826531
d	<b>NaN</b>	0.439335	0.984332

# Creating a Dataframe

- In the example below we can easily create a dataframe from a 2D NumPy array. The array is passed as an argument when the dataframe is created.

```
import pandas as pd  
import numpy as np  
  
arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]], float)  
  
df = pd.DataFrame(arr)  
  
print df
```

# Creating a Dataframe

- We can also specify column names when creating the dataframe.

```
import pandas as pd
import numpy as np

arr = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]], float)

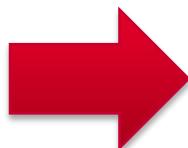
df = pd.DataFrame(arr, columns=['colA', 'colB', 'colC'])
print
print df
```

	colA	colB	colC
0	1	2	3
1	4	5	6
2	7	8	9

# Contents



- Introduction to Pandas Series and Dataframe data structures.



Reading data into a Dataframe

- Accessing Data from a Dataframe
- Merging and Grouping Data



# Dataframe

- The most common way of creating a dataframe is by reading existing data directly into a dataframe
- There are a number of ways of doing this
  - `read_csv`
  - `read_excel`
  - `read_hdf`
  - `read_sql`
  - `read_json`
  - `read_sas ...`
- We will look at how to read from a CSV file.

# Titanic - Dataset



- On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew.
- Although there was some element of luck involved in surviving the sinking, **some groups of people were more likely to survive than others**, such as women, children, and the first-class passengers.
- The dataset we examine contains the details of **891 passengers aboard the titanic**. We will use this as an introduction to the Pandas library.

# Titanic - Dataset



Available as .csv file on Blackboard.

## VARIABLE DESCRIPTIONS:

**survival** Survival

(0 = No; 1 = Yes)

**pclass** Passenger Class

(1 = 1st; 2 = 2nd; 3 = 3rd)

**name** Name

**sex** Sex

**age** Age

**sibsp** Number of Siblings/Spouses Aboard

**parch** Number of Parents/Children Aboard

**ticket** Ticket Number

**fare** Passenger Fare

**cabin** Cabin

**embarked** Port of Embarkation

(C = Cherbourg; Q = Queenstown; S = Southampton)

L1													Embarked	
	A	B	C	D				E	F	G	H	I	J	K
1	PassengerId	Survived	Pclass	Name				Sex	Age	SibSp	Parch	Ticket	Fare	Cabin
2	1	0	3	Braund, Mr. Owen Harris				male	22	1	0	A/5 21171	7.25	S
3	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Thayer)				female	38	1	0	PC 17599	71.2833	C85
4	3	1	3	Heikkinen, Miss. Laina				female	26	0	0	STON/O2.	7.925	S
5	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)				female	35	1	0	113803	53.1	C123
6	5	0	3	Allen, Mr. William Henry				male	35	0	0	373450	8.05	S
7	6	0	3	Moran, Mr. James				male		0	0	330877	8.4583	Q
8	7	0	1	McCarthy, Mr. Timothy J				male	54	0	0	17463	51.8625	E46
9	8	0	3	Palsson, Master. Gosta Leonard				male	2	3	1	349909	21.075	S
10	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)				female	27	0	2	347742	11.1333	S
11	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)				female	14	1	0	237736	30.0708	C
12	11	1	3	Sandstrom, Miss. Marguerite Rut				female	4	1	1	PP 9549	16.7	G6
13	12	1	1	Bonnell, Miss. Elizabeth				female	58	0	0	113783	26.55	C103
14	13	0	3	Saundcock, Mr. William Henry				male	20	0	0	A/5. 2151	8.05	S
15	14	0	3	Andersson, Mr. Anders Johan				male	39	1	5	347082	31.275	S
16	15	0	3	Vestrom, Miss. Hulda Amanda Adolfina				female	14	0	0	350406	7.8542	S
17	16	1	2	Hewlett, Mrs. (Mary D Kingcome)				female	55	0	0	248706	16	S
18	17	0	3	Rice, Master. Eugene				male	2	4	1	382652	29.125	Q
19	18	1	2	Williams, Mr. Charles Eugene				male		0	0	244373	13	S
20	19	0	3	Vander Planke, Mrs. Julius (Emelia Maria Vandemoortele)				female	31	1	0	345763	18	S
21	20	1	3	Masselmani, Mrs. Fatima				female		0	0	2649	7.225	C
22	21	0	2	Fynney, Mr. Joseph J				male	35	0	0	239865	26	S

# Reading Data from a File

- To pull in the text file, we will use the pandas function *read\_csv* method. Let us take a look at this function and what inputs it takes.
- The *read\_csv* has a very large number of parameters such as specifying the delimiter, included headers, etc

```
# General syntax to import specific functions in a library:  
Import pandas as pd  
  
df = pd.read_csv("titanic.csv")  
  
print type(df)  
  
print df
```

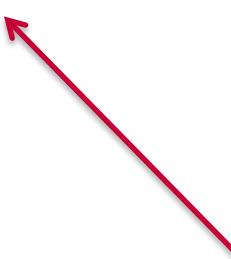
[http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read\\_csv.html](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html)



File Edit Shell Debug Options Windows Help

RESTART

```
>>>
Python version 2.7.5 (default, May 15 2013, 22:43:36) [MSC v.1500 32 bit (Intel)]
Pandas version 0.14.1
<class 'pandas.core.frame.DataFrame'>
   PassengerId  Survived  Pclass \
0              1         0      3
1              2         1      1
2              3         1      3
3              4         1      1
4              5         0      3
5              6         0      3
6              7         0      1
7              8         0      3
8              9         1      3
9             10        1      2
10            11        1      3
11            12        1      1
12            13        0      3
13            14        0      3
14            15        0      3
15            16        1      2
16            17        0      3
17            18        1      2
18            19        0      3
19            20        1      3
20            21        0      2
21            22        1      2
22            23        1      3
23            24        1      1
24            25        0      3
25            26        1      3
26            27        0      3
27            28        0      1
28            29        1      3
29            30        0      3
...
861           862        0      2
862           863        1      1
863           864        0      3
```



The data is read from the .csv file into a pandas data structure called a data frame (same terminology used in R)

You can think of this object as holding the contents of the titanic dataset in a format similar to a database table or an excel spreadsheet.

Each column you see in the dataframe is a **Series** object



# Describing a DataFrame

- DataFrame's have a very useful **describe** method, which is used for seeing **basic statistics** about the dataset's numeric columns.
  - It will return information on all columns of a numeric datatype, therefore some of the data may not be of use .
  - The data type of what is returned is itself a dataframe

```
df = pd.read_csv("titanic.csv")  
  
print type(df)  
  
print df.describe()
```

```
PassengerId    Survived     Pclass      Age     SibSp  \
count    891.000000  891.000000  891.000000  714.000000  891.000000
mean     446.000000   0.383838   2.308642   29.699118   0.523008
std      257.353842   0.486592   0.836071   14.526497   1.102743
min      1.000000   0.000000   1.000000   0.420000   0.000000
25%     223.500000   0.000000   2.000000  20.125000   0.000000
50%     446.000000   0.000000   3.000000  28.000000   0.000000
75%     668.500000   1.000000   3.000000  38.000000   1.000000
max     891.000000   1.000000   3.000000  80.000000   8.000000

Parch      Fare
count    891.000000  891.000000
mean      0.381594  32.204208
std       0.806057  49.693429
min      0.000000   0.000000
25%     0.000000   7.910400
50%     0.000000  14.454200
75%     0.000000  31.000000
max      6.000000  512.329200
```

We can easily see the average age of the passengers is 29.6 years old, with the youngest being 0.42 and the oldest being 80. The median age is 28, with the youngest quartile of users being 20 or younger, and the oldest quartile being at least 38

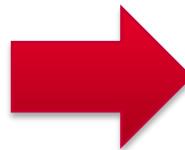
# Discussion



# Contents



- Introduction to Pandas Series and Dataframe data structures.
- Reading data into a Dataframe



Accessing Data from a Dataframe

- Merging and Grouping Data

# Accessing Column Data

- To select a column, we index with the name of the column:
- dataframe['columnName']**

```
df = pd.read_csv("titanic.csv")  
  
print (df['Age'])
```

- Note this column is returned as a **Series object**

Alternatively, a column of data may be accessed using the dot notation with the column name as an attribute (df.Age). Although it works with this particular example, it is not best practice and is prone to error and misuse. Column names with spaces or special characters cannot be accessed in this manner.

```
76 Python 2.7.5 Shell  
File Edit Shell Debug Options Windows Help  
630 male 80.0 0 0 27042 30.00  
851 male 74.0 0 0 347060 7.77  
493 male 71.0 0 0 PC 17609 49.50  
96 male 71.0 0 0 PC 17754 34.65  
116 male 70.5 0 0 370369 7.75  
===== RESTART =====  
>>>  
>>>  
0 22  
1 38  
2 26  
3 35  
4 35  
5 NaN  
6 54  
7 2  
8 27  
9 14  
10 4  
11 58  
12 20  
13 39  
14 14  
...  
876 20  
877 19  
878 NaN  
879 56  
880 25  
881 33  
882 22  
883 28  
884 25  
885 39  
886 27  
887 19  
888 NaN  
889 26  
890 32  
Name: Age, Length: 891, dtype: float64  
>>> |
```

# Accessing Row Data

- To get the first 5 rows of a dataframe, we can use a slice: df[0:5] or df[:5].

```
df = pd.read_csv("titanic.csv")
```

```
firstEntries = df[:5]
```

```
print(firstEntries)
```

As with NumPy

a slice returns a view of the original data.

Any changes made to view will be reflected in the original dataframe

	PassengerId	Survived	Pclass	\
0		1	0	3
1		2	1	1
2		3	1	3
3		4	1	1
4		5	0	3

	Name	Sex	Age	SibSp	\
0	Braund, Mr. Owen Harris	male	22	1	
1	Cumings, Mrs. John Bradley (Florence Briggs Th... Heikkinen, Miss. Laina	female	38	1 0	
2	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	26		
3	Allen, Mr. William Henry	male	35	0	

	Parch	Ticket	Fare	Cabin	Embarked
0	0	A/5 21171	7.2500	NaN	S
1	0	PC 17599	71.2833	C85	C
2	0	STON/O2. 3101282	7.9250	NaN	S
3	0	113803	53.1000	C123	S
4	0	373450	8.0500	NaN	S

# Accessing Rows and Individual Data Items



- We can combine the techniques we saw in the previous slides in order to get the first 10 rows of a specific column (age in this case):

```
df = pd.read_csv("titanic.csv")
print (df['Age'][:10])
```

To access a specific data item within a data frame we can use the following  
**df['columnName'][rowNumber]**

```
df = pd.read_csv("titanic.csv")
print (df['Age'][11])
```

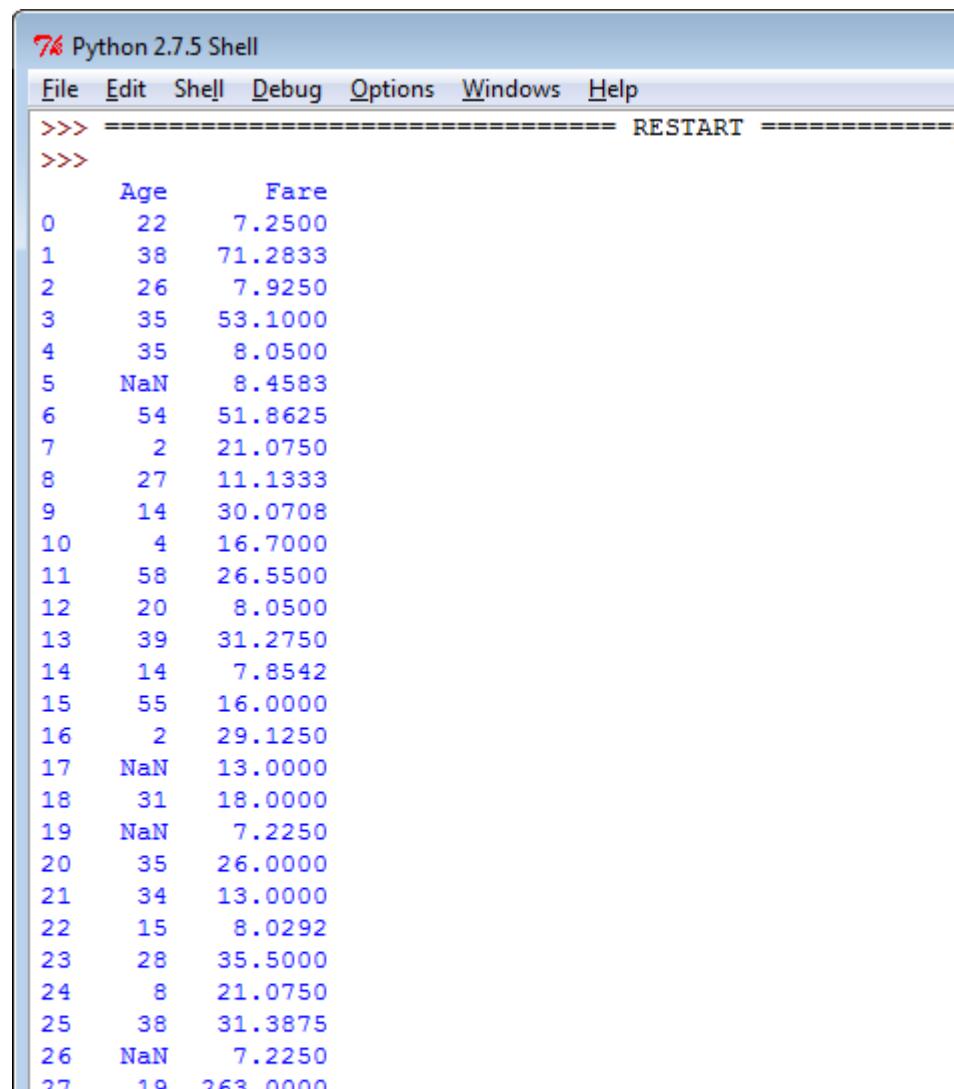
```
>>>
0    22
1    38
2    26
3    35
4    35
5    NaN
6    54
7     2
8    27
9    14
Name: Age, dtype: float64
```

```
>>>
58.0
>>>
```

# Selecting Multiple Columns

- Pandas makes it really easy to select a subset of the columns: just index which list of columns you want.
- Note this returns another dataframe
  - (note the **double square brackets**)

```
df = pd.read_csv("titanic.csv")
print (df[['Age', 'Fare']])
```



A screenshot of the Python 2.7.5 Shell interface. The title bar says "Python 2.7.5 Shell". The menu bar includes File, Edit, Shell, Debug, Options, Windows, and Help. Below the menu is a command line window. The first two lines show the Python prompt: >>> followed by a series of equals signs and the word "RESTART". The third line shows another prompt: >>>. The main area displays a table of data with two columns: "Age" and "Fare". The data consists of 27 rows, indexed from 0 to 26. The "Age" column contains values like 22, 38, 26, etc., and the "Fare" column contains values like 7.2500, 71.2833, 7.9250, etc.

	Age	Fare
0	22	7.2500
1	38	71.2833
2	26	7.9250
3	35	53.1000
4	35	8.0500
5	NaN	8.4583
6	54	51.8625
7	2	21.0750
8	27	11.1333
9	14	30.0708
10	4	16.7000
11	58	26.5500
12	20	8.0500
13	39	31.2750
14	14	7.8542
15	55	16.0000
16	2	29.1250
17	NaN	13.0000
18	31	18.0000
19	NaN	7.2250
20	35	26.0000
21	34	13.0000
22	15	8.0292
23	28	35.5000
24	8	21.0750
25	38	31.3875
26	NaN	7.2250
27	19	263.0000

# Accessing Column (Series) Data

- We mentioned in a previous slide that you can also think of a DataFrame as a **group of Series objects** that share an index. When you access an individual column from a dataframe the datatype returned is a series.
  - Note if you extract multiple columns the data type returned is still a dataframe

```
df = pd.read_csv("titanic.csv")
```

```
ages = df['Age']  
print type(ages)
```

```
moreInfo = df[['Age', 'Name']]  
print (type(moreInfo))
```

```
<class 'pandas.core.series.Series'>  
<class 'pandas.core.frame.DataFrame'>
```

# Using Head and Tail

- To view a small sample of a Series or DataFrame object, use the head (start) and tail (end) methods. The default number of elements to display is five, but you can pass a number as an argument.

```
df = pd.read_csv("titanic.csv")
freqAges = df['Age']
print(freqAges.head())
print
print(freqAges.tail())
```

```
>>>
0    22
1    38
2    26
3    35
4    35
Name: Age, dtype: float64

886    27
887    19
888    NaN
889    26
890    32
Name: Age, dtype: float64
>>>
```

- If I want to capture the last 7 age values in the dataset

```
df = pd.read_csv("titanic.csv")
print(df["Age"].tail(7))
```

# Accessing Data - .head and .tail

- It is important to understand that if you **extract a column** from a dataframe you are working with a **view of the same data**.
- Both the dataframe and the column you have extracted refer to the same data.
- In the example below you will see that the change made to allAges will be reflected in the dataframe age column.

```
import pandas as pd  
  
df = pd.read_csv('titanic.csv')  
  
print (df['Age'].head(5))  
  
allAges = df['Age']  
  
allAges[0] = 877  
print (df['Age'].head(5))
```

```
0    22  
1    38  
2    26  
3    35  
4    35  
Name: Age, dtype: float64  
0    877  
1    38  
2    26  
3    35  
4    35  
Name: Age, dtype: float64
```

# Counting – value\_counts()

- A very useful method **value\_counts()** can be used to count the **number of occurrences of each entry** in a column (it returns a Series object)
- It presents the results in **descending** order
- For examples, how many males and females are represented in dataset

```
df = pd.read_csv("titanic.csv")
print (df['Sex'].value_counts())
```

```
male    577
female   314
dtype: int64
```

## Example 1

- Read data in from the titanic dataset and determine the four most common ages represented.

```
df = read_csv("titanic.csv")
freqAges = df['Age']
print (freqAges.value_counts().head(4))
```

```
>>>
24    30
22    27
18    26
28    25
dtype: int64
```



# Performing Operations

- We can perform the same mathematical operations in Pandas as we could in NumPy

```
import pandas as pd
import numpy as np

df = pd.read_csv("titanic.csv")
print ("Average age", np.mean(df["Age"]))

print (df["Age"].head(5))
df["Age"] += 5

print (df["Age"].head(5))
```

Average age 29.6991176471

0 22

1 38

2 26

3 35

4 35

Name: Age, dtype: float64

0 27

1 43

2 31

3 40

4 40

Name: Age, dtype: float64

# Querying the Dataset

- You can combine multiple queries within the [] after the dataset. Think of the square brackets as a way of refining the data you want.
- In the code we find the names of all those people that did not survive the sinking of the titanic

```
df = read_csv("titanic.csv")  
  
print (df['Survived']==0)  
  
names = df['Name'][df['Survived']==0]  
print (names)
```

```
0      True  
1     False  
2     False  
3     False  
4      True  
5      True  
6      True  
7      True  
8     False  
9     False  
10    False  
11    False  
12      True  
13      True  
14      True  
...  
876     True  
877     True  
878     True
```

```
0      Braund, Mr. Owen Harris  
4      Allen, Mr. William Henry  
5          Moran, Mr. James  
6      McCarthy, Mr. Timothy J  
7      Palsson, Master. Gosta Leonard  
12      Saundercock, Mr. William Henry  
13      Andersson, Mr. Anders Johan  
14      Vestrom, Miss. Hulda Amanda Adolfina  
16          Rice, Master. Eugene  
18      Vander Planke, Mrs. Julius (Emelia Maria Vande...  
20          Fynney, Mr. Joseph J  
24      Palsson, Miss. Torborg Danira
```

## Exercise 2

- I want to determine the name and age of all those that died on the titanic that were under 10 years of age.

```
freqYoungAge = df[['Name', 'Age']][df['Survived']==0][df['Age']<10]
```

	Name	Age
7	Palsson, Master. Gosta Leonard	2
16	Rice, Master. Eugene	2
24	Palsson, Miss. Torborg Danira	8
50	Panula, Master. Juha Niilo	7
63	Skoog, Master. Harald	4
119	Andersson, Miss. Ellis Anna Maria	2
147	Ford, Miss. Robina Maggie "Ruby"	9
164	Panula, Master. Eino Viljami	1
171	Rice, Master. Arthur	4
182	Asplund, Master. Clarence Gustaf Hugo	9
205	Strom, Miss. Telma Matilda	2
278	Rice, Master. Eric	7
297	Allison, Miss. Helen Loraine	2
374	Palsson, Miss. Stina Viola	3
386	Goodwin, Master. Sidney Leonard	1
480	Goodwin, Master. Harold Victor	9
541	Andersson, Miss. Ingeborg Constanzia	9
634	Skoog, Miss. Mabel	9
642	Skoog, Miss. Margit Elizabeth	2
787	Rice, Master. George Hugh	8
813	Andersson, Miss. Ebba Iris Alfrida	6
824	Panula, Master. Urho Abraham	2
850	Andersson, Master. Sigvard Harald Elias	4
852	Boulos, Miss. Nourelain	9



Again if you were to print out the result of these two conditions it would be an array of booleans (as demonstrated in the previous slide). Only where the boolean entry is true for both survived and age will that row be selected from the dataset

# Example

- In the following example I want to print the **number of males** and **females** that survived and those that didn't from each pclass.

```
import pandas as pd

def pClassSurvivorDetails(pClass, data):

    print "\nResults for Pclass =", pClass, "-----"
    print "The following did not survive"
    notSurvive = df['Sex'][df['Survived']==0][df['Pclass']==pClass]
    print notSurvive.value_counts()

    print "The following did survive"
    survive = df['Sex'][df['Survived']==1][df['Pclass']==pClass]
    print survive.value_counts()

def main():
    df = pd.read_csv("titanic.csv")
    for value in [1, 2, 3]:
        pClassSurvivorDetails(value, df)
```

**Results for Pclass = 1**

The following did not survive

male 77

female 3

dtype: int64

The following did survive

female 91

male 45

dtype: int64

**Results for Pclass = 2**

The following did not survive

male 91

female 6

dtype: int64

The following did survive

female 70

male 17

dtype: int64

**Results for Pclass = 3**

The following did not survive

male 300

female 72

dtype: int64

The following did survive

female 72

male 47

dtype: int64

# Combining Conditions using & and |

- It is also very useful to use & and | to combine conditions.
  - For example I want to search the data to return all cases that satisfy all of these conditions.
    - All those that have pclass =1
    - All those that boarded in Southampton
    - All those older than 20 years

```
pClass = df['Pclass']==1  
sBoard = df['Embarked']=="S"  
ages = df['Age']>20  
  
print df[pClass & sBoard & ages]
```

# Combining Conditions using & and |

- I can easily introduce an or connective by using | to link the various condition I use.

```
pClass = df['Pclass']==1  
sBoard = df['Embarked']=="S"  
ages = df['Age']>20  
  
print df[["Pclass", "Embarked", "Age"]][pClass | sBoard | ages]
```

0	3	S	22
1	1	C	38
2	3	S	26
3	1	S	35
4	3	S	35
6	1	S	54
7	3	S	2
8	3	S	27
10	3	S	4
11	1	S	58
12	3	S	20
13	3	S	39
14	3	S	14
15	2	S	55
17	2	S	NaN
18	3	S	31
20	2	S	35
21	2	S	34
23	1	S	28 .....
24	3	S	8
25	3	S	38

# Converting Series to NumPy Array - .values

- We already mentioned that Dataframe is composed on multiple Series object.
- However, a Series object is internally a NumPy array.
- If you add values to the end of any Series, you'll get its internal numpy array
- We can also add .values to a dataframe to produce a 2D numpy array
- To do this you need to ensure there are only numerical contents in all columns

```
df = pd.read_csv("titanic.csv")

ages = df['Age']
print type(ages)

nAges = ages.values

print type(nAges)
```

```
<class 'pandas.core.series.Series'>

<type 'numpy.ndarray'>
```

# Example

- In this short program we will show the number of first, second and third class passengers that died when the titanic sunk.

```
bDeaths = df['Survived']==0  
allDeaths = df[bDeaths]  
  
print allDeaths['Pclass'].value_counts()
```

3	372
2	97
1	80
dtype: int64	

# Exercise

- Of course it might be more accurate to measure the number of 1, 2 and 3<sup>rd</sup> class passengers that died against the number of such passengers that boarded.

```
bDeaths = df['Survived']==0  
allDeaths = df[bDeaths]  
  
deathsFreq = allDeaths['Pclass'].value_counts()  
allPassengers = df['Pclass'].value_counts()  
  
print (deathsFreq*100)/ allPassengers )
```

```
1 37.037037  
2 52.717391  
3 75.763747  
dtype: float64
```

# Data Frame Analysis - Sorting

- The sort function is very useful. It's general syntax is
  - Sort(['Column1', 'Column2', ...], ascending=[True, False, ...])
- To sort the details of all passengers in terms of ascending age, we can sort the dataframe in ascending order

```
df = read_csv("titanic.csv")  
  
sorted = df.sort(['Age'], ascending=[False])  
  
print sorted[:6]
```

# Data Frame Analysis - Sorting

```
PassengerId  Survived  Pclass          Name \
630          631       1      1  Barkworth, Mr. Algernon Henry Wilson
851          852       0      3           Svensson, Mr. Johan
493          494       0      1  Artagaveytia, Mr. Ramon
96           97       0      1  Goldschmidt, Mr. George B
116         117       0      3        Connors, Mr. Patrick
672          673       0      2  Mitchell, Mr. Henry Michael

   Sex   Age  SibSp  Parch     Ticket     Fare Cabin Embarked
630  male  80.0      0      0    27042  30.0000    A23      S
851  male  74.0      0      0    347060  7.7750    NaN      S
493  male  71.0      0      0  PC 17609  49.5042    NaN      C
96   male  71.0      0      0  PC 17754  34.6542     A5      C
116  male  70.5      0      0    370369  7.7500    NaN      Q
672  male  70.0      0      0    C.A. 24580  10.5000    NaN      S
>>>
```

# Data Frame Analysis - Sorting

- If I wanted to sort the data in terms of ascending age and descending fare.

```
sorted = df.sort(['Age', 'Fare'], ascending=[True, False])
```

	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	\
803	male	0.42	0	1	2625	8.5167	NaN	
755	male	0.67	1	1	250649	14.5000	NaN	
469	female	0.75	2	1	2666	19.2583	NaN	
644	female	0.75	2	1	2666	19.2583	NaN	
78	male	0.83	0	2	248738	29.0000	NaN	
831	male	0.83	1	1	29106	18.7500	NaN	
305	male	0.92	1	2	113781	151.5500	C22 C26	
386	male	1.00	5	2	CA 2144	46.9000	NaN	
164	male	1.00	4	1	3101295	39.6875	NaN	
183	male	1.00	2	1	230136	39.0000	F4	
827	male	1.00	0	2	S.C./PARIS 2079	37.0042	NaN	
788	male	1.00	1	2	C.A. 2315	20.5750	NaN	
381	female	1.00	0	2	2653	15.7417	NaN	
172	female	1.00	1	1	347742	11.1333	NaN	
297	female	2.00	1	2	113781	151.5500	C22 C26	
824	male	2.00	4	1	3101295	39.6875	NaN	
119	female	2.00	4	2	347082	31.2750	NaN	
16	male	2.00	4	1	382652	29.1250	NaN	
642	female	2.00	3	2	347088	27.9000	NaN	
340	male	2.00	1	1	230080	26.0000	F2	
530	female	2.00	1	1	26360	26.0000	NaN	
7	male	2.00	3	1	349909	21.0750	NaN	
479	female	2.00	0	1	3101298	12.2875	NaN	
205	female	2.00	0	1	347054	10.4625	G6	
43	female	3.00	1	2	SC/Paris 2123	41.5792	NaN	

Notice the data is arranged in ascending age but in descending fare

# Storing the Data in a File

- I want to extract three columns (name, age, survived) from the dataset and store them as a new dataset file. To write a dataframe to a file you can just use the `to_csv` function that takes in the name of the file you want to write the data to.
- In the example below we extract a subset of the titanic dataset (age, name and survived columns and write that data to the csv file)

```
df = read_csv("titanic.csv")

shortDataframe = df[['Age', 'Name', 'Survived']]

shortDataframe.to_csv('titanic_short.csv')
```

titanic\_short.csv - Microsoft Excel

File Home Insert Page Layout Formulas Data Review View

Cut Copy Paste Format Painter Clipboard

Font Alignment Number Styles

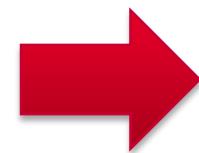
A1 fx

	A	B	C	D	E	F	G	H	I	J	K
1	Age	Name	Survived								
2	0 Braund, Mr. Owen Harris		22	0							
3	1 Cumings, Mrs. John Bradley (Florence Brig		38	1							
4	2 Heikkinen, Miss. Laina		26	1							
5	3 Futrelle, Mrs. Jacques Heath (Lily May Pee		35	1							
6	4 Allen, Mr. William Henry		35	0							
7	5 Moran, Mr. James			0							
8	6 McCarthy, Mr. Timothy J		54	0							
9	7 Palsson, Master. Gosta Leonard		2	0							
10	8 Johnson, Mrs. Oscar W (Elisabeth Vilhelmi		27	1							
11	9 Nasser, Mrs. Nicholas (Adele Achem)		14	1							
12	10 Sandstrom, Miss. Marguerite Rut		4	1							
13	11 Bonnell, Miss. Elizabeth		58	1							
14	12 Saundercock, Mr. William Henry		20	0							
15	13 Andersson, Mr. Anders Johan		39	0							
16	14 Vestrom, Miss. Hulda Amanda Adolfina		14	0							
17	15 Hewlett, Mrs. (Mary D Kingcome)		55	1							
18	16 Rice, Master. Eugene		2	0							
19	17 Williams, Mr. Charles Eugene			1							
20	18 Vander Planke, Mrs. Julius (Emelia Maria \		31	0							
21	19 Masselmani, Mrs. Fatima			1							
22	20 Fynney, Mr. Joseph J		35	0							

# Contents



- Introduction to Pandas Series and Dataframe data structures.
- Reading data into a Dataframe
- Accessing Data from a Dataframe



Merging and Grouping Data

# Merging Data

- **pandas.merge** allows two DataFrames to be joined on one or more keys. The function provides a series of parameters allowing you to specify the columns or indexes on which to join

how : {'left', 'right', 'outer', 'inner'}, default 'inner'

left: use only keys from left frame (SQL: left outer join)

right: use only keys from right frame (SQL: right outer join)

outer: use union of keys from both frames (SQL: full outer join)

inner: use intersection of keys from both frames (SQL: inner join)

# Merging Data

- This kind of merging is very useful if you have separate datasets with a common key
- Default join is an inner join (selects all rows from both tables as long as there is a match between the columns in both tables)

```
left = pd.DataFrame({'names': ['john', 'tim', 'tom'], 'colLeft': [1, 2, 3]})  
right = pd.DataFrame({'names': ['fred', 'tim', 'tom'], 'colRight': [4, 5, 6]})
```

```
print left  
print right  
print  
print pd.merge(left, right, on='names', how='inner')
```

Notice that we lose values from both frames since certain keys do not match up

	colLeft	names
0	1	john
1	2	tim
2	3	tom

	colRight	names
0	4	fred
1	5	tim
2	6	tom

	colLeft	names	colRight
0	2	tim	5
1	3	tom	6

# Merging Data – Outer Join

- The following is an example of an **outer join**.
- We keep everything from both frames, regardless of whether or not there was a match on both sides. Where there was not a match, the values corresponding to that key are NULL (NaN).

```
left = pd.DataFrame({'names': ['john', 'tim', 'tom'], 'colLeft': [1, 2, 3]})  
right = pd.DataFrame({'names': ['fred', 'tim', 'tom'], 'colRight': [4, 5, 6]})
```

```
print left  
print  
print right  
print  
print merge(left, right, on='names', how='outer')
```

	colLeft	names
0	1	john
1	2	tim
2	3	tom

	colRight	names
0	4	fred
1	5	tim
2	6	tom

	colLeft	names	colRight
0	1	john	NaN
1	2	tim	5
2	3	tom	6
3	NaN	fred	4

# Merging Data

- The following is an example of a left outer join.
  - We keep everything from the left frame, pulling in the value from the right frame where the keys match up. The right\_value is NULL where keys do not match (NaN).

```
left = pd.DataFrame({'names': ['john', 'tim', 'tom'], 'colLeft': [1, 2, 3]})  
right = pd.DataFrame({'names': ['fred', 'tim', 'tom'], 'colRight': [4, 5, 6]})
```

```
print left  
print  
print right  
print  
print merge(left, right, on='names', how='left')
```

	colLeft	names
0	1	john
1	2	tim
2	3	tom

	colRight	names
0	4	fred
1	5	tim
2	6	tom

	colLeft	names	colRight
0	1	john	NaN
1	2	tim	5
2	3	tom	6

# Merging Data

- The following is an example of a right outer join.
  - This time we've kept everything from the right frame with the left\_value being NULL where the right frame's key did not find a match.

```
print merge(left, right, on='names', how='right')
```

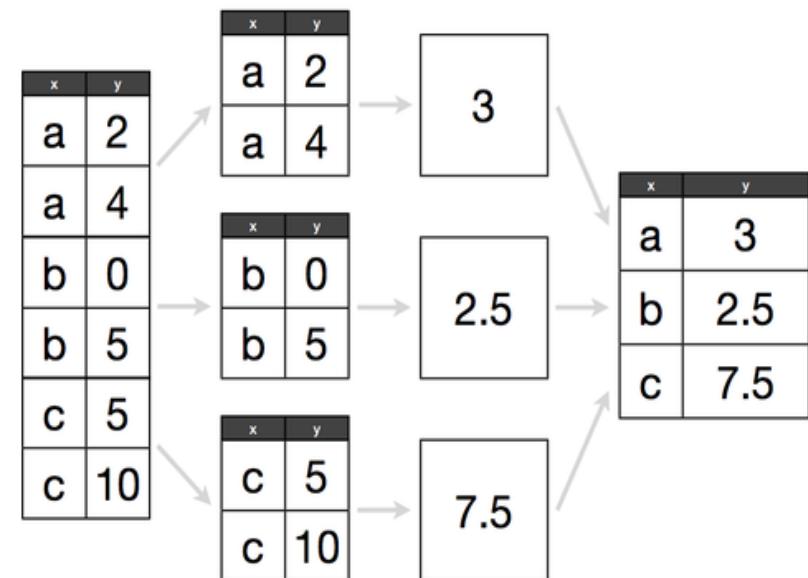
```
      colLeft names
0          1   john
1          2    tim
2          3   tom

      colRight names
0          4   fred
1          5    tim
2          6   tom

      colLeft names  colRight
0          2    tim        5
1          3    tom        6
2         NaN   fred        4
```

# Using GroupBy

- When approaching a data analysis problem, you'll often break it apart into manageable pieces, perform some operations on each of the pieces, and then put everything back together again.
- Pandas groupby method is useful for this as it facilitates the following steps:
  - Splitting the data into groups based on some criteria
  - Applying a function to each group independently
  - Combining the results into a data structure





# Using GroupBy

- Pandas groupby function returns a DataFrameGroupBy object which has a variety of methods
  - Count returns the total number of NOT NULL values within each column.
  - For example 186 of the entries in the Age column have a Pclass of 1

```
df = read_csv("titanic.csv")

sorted = df.groupby('Pclass')
print sorted.count()
```

```
<pandas.core.groupby.DataFrameGroupBy object at 0x0629F0B0>
    PassengerId  Survived  Name  Sex  Age  SibSp  Parch  Ticket  Fare \
Pclass
1           216       216   216   216   216   186     216      216   216
2           184       184   184   184   184   173     184      184   184
3           491       491   491   491   491   355     491      491   491

    Cabin  Embarked
Pclass
1         176       214
2          16       184
3          12       491
```

# Using GroupBy

- You can perform mathematical operations such as sum, median, mean on the groupby object.
- For example, lets try to find out the average price paid by first class, second class and third class passengers

```
groupDF = df.groupby('Pclass')
print groupDF['Fare']
print groupDF['Fare'].mean()
```

```
print
print groupDF['Age'].mean()
```

```
<pandas.core.groupby.SeriesGroupBy object at 0x062460D0>
Pclass
1      84.154687
2      20.662183
3      13.675550
Name: Fare, dtype: float64

Pclass
1      38.233441
2      29.877630
3      25.140620
Name: Age, dtype: float64
```

# GroupBy

- The size method is a simple but useful methods that returns the number of entries in each group.
- For example lets determine the top 10 most common ages on the titanic voyage.

```
groupDF = df.groupby('Age')  
  
print groupDF.size().order(ascending=False).head(10)
```



The .size method will return the number of entries for each unique age in the dataset. For example, 30 of the entries have an age of 24.

Age	
24	30
22	27
18	26
30	25
19	25
28	25
21	24
25	23
36	22
29	20
	<b>dtype: int64</b>

# Using GroupBy

- In the example below we use the GroupBy object to group according to the attributes Pclass and Sex

```
df = read_csv("titanic.csv")  
  
sorted = df.groupby(['Pclass', 'Sex'])  
  
print sorted.count()
```

		PassengerId	Survived	Name	Age	SibSp	Parch	Ticket	Fare	\
Pclass	Sex									
1	female	94	94	94	85	94	94	94	94	
	male	122	122	122	101	122	122	122	122	
2	female	76	76	76	74	76	76	76	76	
	male	108	108	108	99	108	108	108	108	
3	female	144	144	144	102	144	144	144	144	
	male	347	347	347	253	347	347	347	347	
		Cabin	Embarked							
Pclass	Sex									
1	female	81	92							
	male	95	122							
2	female	10	76							
	male	6	108							
3	female	6	144							
	male	6	347							

# URL



- <http://pandas.pydata.org/pandas-docs/stable/groupby.html>
- <http://synesthesiam.com/posts/an-introduction-to-pandas.html#getting-started>
- <http://www.gregreda.com/2013/10/26/working-with-pandas-dataframes/>
  
- Need to include material on matplotlib with pandas as well.



# Titanic - Dataset

## SPECIAL NOTES:

Pclass is a proxy for socio-economic status (SES)

1st ~ Upper; 2nd ~ Middle; 3rd ~ Lower

Age is in Years; Fractional if Age less than One (1)

If the Age is Estimated, it is in the form xx.5

With respect to the family relation variables (i.e. sibsp and parch) some relations were ignored. The following are the definitions used for sibsp and parch.

Sibling: Brother, Sister, Stepbrother, or Stepsister of Passenger Aboard Titanic

Spouse: Husband or Wife of Passenger Aboard Titanic (Mistresses and Fiances Ignored)

Parent: Mother or Father of Passenger Aboard Titanic

Child: Son, Daughter, Stepson, or Stepdaughter of Passenger Aboard Titanic

# Discussion



# Thank you

[Haithem. afli@cit.ie](mailto:Haithem.afli@cit.ie)

[@AfliHaithem](https://twitter.com/AfliHaithem)