

Practical Machine Learning



Practical Machine Learning

Lecture: Pandas Part 1: Series and DataFrame Objects

Ted Scully

Pandas

- NumPy is a great tool for dealing with **numeric matrices** and vectors in Python
 - For more complex data, it is limited.
- Fortunately, when dealing with complex data we can use the **powerful Python data analysis toolkit** (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
 - <https://pandas.pydata.org/docs/reference/index.html>

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

You will notice the syntax and functionality used in a Series object is quite similar to that of a NumPy array.

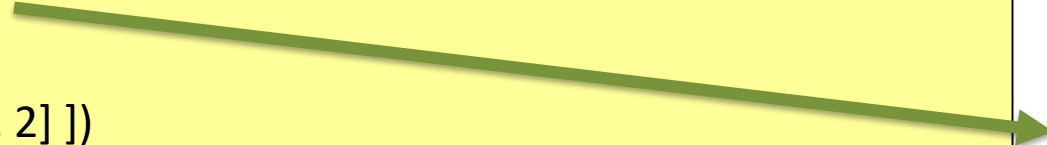
```
import numpy as np
import pandas as pd
```

```
s1 = pd.Series(np.random.randn(5))
```

```
print (s1)
```



```
print (s1[1])
```



```
print (s1[ [1, 2] ])
```

```
print (s1[[True,False, False, False, True]])
```

```
print (s1[0:3])
```

```
print (np.square(s1))
```

0 -0.635304

1 -1.314937

2 -0.006525

3 -1.223736

4 0.188021

dtype: float64

-1.3149367389155941

Series - Examples

You will notice the syntax and functionality used in a Series object is quite similar to that of a NumPy array.

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

```
s1 = pd.Series(np.random.randn(5))
```

```
print (s1)
```

```
print (s1[1])
```

```
print (s1[ [1, 2] ])
```

```
print (s1[[True,False, False, False, True]])
```

```
print (s1[0:3])
```

```
print (np.square(s1))
```

```
1 -1.314937
2 -0.006525
dtype: float64
```

```
0 -0.635304
4  0.188021
dtype: float64
```

Series - Examples

You will notice the syntax and functionality used in a Series object is quite similar to that of a NumPy array.

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

```
s1 = pd.Series(np.random.randn(5))
```

```
print (s1)
```

```
print (s1[1])
```

```
print (s1[ [1, 2] ])
```

```
print (s1[[True,False, False, False, True]])
```

```
print (s1[0:3])
```

```
print (np.square(s1))
```

```
0 -0.635304
1 -1.314937
2 -0.006525
dtype: float64
```

```
0  0.403612
1  1.729059
2  0.000043
3  1.497530
4  0.035352
dtype: float64
```

Series

- We can use conditional statements in the same way as we saw with NumPy
 - **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] ) )
```

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

As with NumPy, Boolean indexing like this creates 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'>
```

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.
- 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 from a NumPy Array

- In the example below we 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)
```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 2.0 | 3.0 |
| 1 | 4.0 | 5.0 | 6.0 |
| 2 | 7.0 | 8.0 | 9.0 |

Creating a DataFrame

- Remember we mentioned you can view a dataset as a group of Series object. Here create a DataFrame by passing it a number 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 | NaN | 0.396005 |
| b | 0.671142 | 0.263916 | 0.532836 |
| c | 0.116057 | 0.839463 | 0.826531 |
| d | NaN | 0.439335 | 0.984332 |

Revert from DataFrame to NumPy Array

- It is very easy to convert from a DataFrame object to a NumPy array using `.to_numpy()`.
- We can also convert a **Series object** to a NumPy array in the same way!

```
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 )

dataArr = df.to_numpy()

print (dataArr)
print (type(dataArr))
```

| | 0 | 1 | 2 |
|---|-----|-----|-----|
| 0 | 1.0 | 2.0 | 3.0 |
| 1 | 4.0 | 5.0 | 6.0 |
| 2 | 7.0 | 8.0 | 9.0 |

```
[[ 1.  2.  3.]
 [ 4.  5.  6.]
 [ 7.  8.  9.]]
```

```
<class 'numpy.ndarray'>
```

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



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) |

File Home Insert Page Layout Formulas Data Review View



Calibri 11 A A

B I U

Font



Alignment

Wrap Text

Merge & Center

General

%

Number



Conditional Formatting



Format as Table



Cell Styles



Insert



Delete



Format



AutoSum



Fill

Editing

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 | Embarked |
| 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 | C |
| 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 | S |
| 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 | S |
| 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 | S |
| 13 | 12 | 1 | 1 | Bonnell, Miss. Elizabeth | female | 58 | 0 | 0 | 113783 | 26.55 | C103 | S |
| 14 | 13 | 0 | 3 | Saunderscock, 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 |

train

Ready

100%

Describing a DataFrame

- To pull in the text file, we will use the pandas function [read_csv](#) method.
- The `read_csv` has a very large number of parameters such as specifying the delimiter, included headers, etc
- Typically it's not very useful to print out an entire dataframe.
- However, there are some useful functions you can use to get summary data.

```
import pandas as pd

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

print (df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId      891 non-null int64
Survived         891 non-null int64
Pclass           891 non-null int64
Name             891 non-null object
Sex              891 non-null object
Age              714 non-null float64
SibSp            891 non-null int64
Parch           891 non-null int64
Ticket           891 non-null object
Fare             891 non-null float64
Cabin           204 non-null object
Embarked         889 non-null object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
None
```

Describing a DataFrame

- DataFrame's also have a useful **describe** method, which is used for viewing **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

Practical Machine Learning



Practical Machine Learning

Lecture: Pandas Part 2: Accessing Data

Ted Scully

Accessing Column Data

- To select a column in a dataframe, 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.

```
4      35.0  
5      NaN  
6      54.0  
7       2.0  
8      27.0  
9      14.0  
10     4.0  
11     58.0  
12     20.0  
13     39.0  
14     14.0  
15     55.0  
16       2.0  
17      NaN  
18     31.0  
19      NaN
```

```
...  
867    31.0  
868     NaN  
869      4.0  
870    26.0  
871    47.0  
872    33.0  
873    47.0  
874    28.0  
875    15.0  
876    20.0  
877    19.0  
878     NaN  
879    56.0  
880    25.0  
881    33.0  
882    22.0  
883    28.0  
884    25.0  
885    39.0  
886    27.0  
887    19.0  
888     NaN  
889    26.0  
890    32.0
```

Name: Age, Length: 891, dtype: float64

Accessing Columns

- 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 data type 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 (freqAges.tail())
```

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

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

```
>>>
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
>>>
```

Using loc and iloc

- A DataFrame consists of **both rows and columns**, and as a result has constructs to select data from specific rows and columns.
- We have already seen the use of `[]` but Pandas also allows us to access data using `.loc[]`, and `.iloc[]`.
- The `.loc` function is primarily label based indexing
- The `iloc` function is used for integer-location based indexing and is similar to what we used in NumPy
- Both `.loc` and `.iloc` work with Series and DataFrame objects.

Using loc

- The **.loc** function is primarily label based indexing
- Pandas loc allows us to access a group of rows and columns by label(s) or using a Boolean array.
- The allowable inputs can be (from [Pandas API](#)):
 - A single label, e.g. 5 or 'a', (note that 5 is interpreted as a label of the index, and never as an integer position along the index).
 - A list or array of labels, e.g. ['a', 'b', 'c'].
 - A slice object with labels, e.g. 'a':'f'. (Note that contrary to usual python slices, **both** the start and the stop are included)
 - A boolean array of the same length as the axis being sliced, e.g. [True, False, True].
- Note, in the titanic dataset we use an **integer index** but the value passed to loc could also be a **String** index if applicable.

Using loc


- **Rows** can be retrieved via an index label value using **.loc[]** on an entire dataframe
- It is important to understand that when we access `df.loc[0]` below it looks for the **matching label** (it doesn't just return the row with position 0)
- Below we can access multiple row directly by pass `loc[]` a list of row labels we want returned. (Single row is returned as a Series, two or more rows is returned as a dataframe)

```
import pandas as pd
```

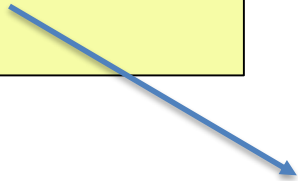
```
df = pd.read_csv("titanic.csv")
```

```
print ( df.loc[0] )
```

```
print (df.loc[ [1, 20] ])
```



| | |
|------------------------|-------------------------|
| PassengerId | 1 |
| Survived | 0 |
| Pclass | 3 |
| Name | Braund, Mr. Owen Harris |
| Sex | male |
| Age | 22 |
| SibSp | 1 |
| Parch | 0 |
| Ticket | A/5 21171 |
| Fare | 7.25 |
| Cabin | NaN |
| Embarked | S |
| Name: 0, dtype: object | |



| PassengerId | Survived | Pclass | ... | Fare | Cabin | Embarked | |
|-------------|----------|--------|-----|---------|---------|----------|---|
| 1 | 2 | 1 | ... | 71.2833 | C85 | C | |
| 20 | 21 | 0 | 2 | ... | 26.0000 | NaN | S |

Using loc on a DataFrame

- To properly illustrate the use of loc on a DataFrame when we read in the DataFrame we **index it using the name column** as shown below (previously it Pandas automatically generated an integer based index starting at 0)

```
import pandas as pd
```

```
df = pd.read_csv("titanic.csv", index_col='Name')
```

```
print (df["Fare"])
```

| Name | |
|---|---------|
| Braund, Mr. Owen Harris | 7.2500 |
| Cumings, Mrs. John Bradley (Florence Briggs Thayer) | 71.2833 |
| Heikkinen, Miss. Laina | 7.9250 |
| Futrelle, Mrs. Jacques Heath (Lily May Peel) | 53.1000 |
| Allen, Mr. William Henry | 8.0500 |
| Moran, Mr. James | 8.4583 |
| McCarthy, Mr. Timothy J | 51.8625 |
| Palsson, Master. Gosta Leonard | 21.0750 |
| Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg) | 11.1333 |
| Nasser, Mrs. Nicholas (Adele Achem) | 30.0708 |
| Sandstrom, Miss. Marguerite Rut | 16.7000 |
| Bonnell, Miss. Elizabeth | 26.5500 |
| Saunderscock, Mr. William Henry | 8.0500 |
| Andersson, Mr. Anders Johan | 31.2750 |
| Vestrom, Miss. Hulda Amanda Adolfina | 7.8542 |
| Hewlett, Mrs. (Mary D Kingcome) | 16.0000 |
| Rice, Master. Eugene | 29.1250 |

Using loc on a DataFrame

- To properly illustrate the use of loc on a DataFrame when I read in the DataFrame I index is using the name column as shown below

```
df = pd.read_csv("titanic.csv", index_col='Name')  
  
print (df.loc[ "Moran, Mr. James" ])  
  
print (df.loc[ "Moran, Mr. James": "Bonnell, Miss. Elizabeth" ])
```

Using loc on a DataFrame

- To properly illustrate the use of loc on a DataFrame when I read in the DataFrame I index is using the name column as shown below

```
df = pd.read_csv("titanic.csv", index_col='Name')
```

```
print (df.loc[ "Moran, Mr. James" ])
```

```
print (df.loc[ "Moran, Mr. James": "Bonnell, Miss. Elizabeth" ])
```

```
PassengerId      6
Survived         0
Pclass           3
Sex              male
Age             NaN
SibSp            0
Parch            0
Ticket           330877
Fare             8.4583
Cabin            NaN
Embarked         Q
Name: Moran, Mr. James, dtype: object
```

Using loc on a DataFrame

- To properly illustrate DataFrame indexing

| | PassengerId | Survived | \ |
|---|-------------|----------|---|
| Name | | | |
| Moran, Mr. James | 6 | 0 | |
| McCarthy, Mr. Timothy J | 7 | 0 | |
| Palsson, Master. Gosta Leonard | 8 | 0 | |
| Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg) | 9 | 1 | |
| Nasser, Mrs. Nicholas (Adele Achem) | 10 | 1 | |
| Sandstrom, Miss. Marguerite Rut | 11 | 1 | |
| Bonnell, Miss. Elizabeth | 12 | 1 | |

| | Pclass | Sex | Age | \ |
|---|--------|--------|------|---|
| Name | | | | |
| Moran, Mr. James | 3 | male | NaN | |
| McCarthy, Mr. Timothy J | 1 | male | 54.0 | |
| Palsson, Master. Gosta Leonard | 3 | male | 2.0 | |
| Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg) | 3 | female | 27.0 | |
| Nasser, Mrs. Nicholas (Adele Achem) | 2 | female | 14.0 | |
| Sandstrom, Miss. Marguerite Rut | 3 | female | 4.0 | |
| Bonnell, Miss. Elizabeth | 1 | female | 58.0 | |

| | SibSp | Parch | Ticket | \ |
|-------------------------|-------|-------|--------|---|
| Name | | | | |
| Moran, Mr. James | 0 | 0 | 330877 | |
| McCarthy, Mr. Timothy J | 0 | 0 | 17463 | |

```
df = pd.read_csv(
```

```
print (df.loc[ "Mo
```

```
print (df.loc[ "Moran, Mr. James": "Bonnell, Miss. Elizabeth" ])
```

Selecting DataFrame rows and columns simultaneously - loc

Up to now we have used loc to extract entire rows, here we extract rows and selected columns

```
import pandas as pd

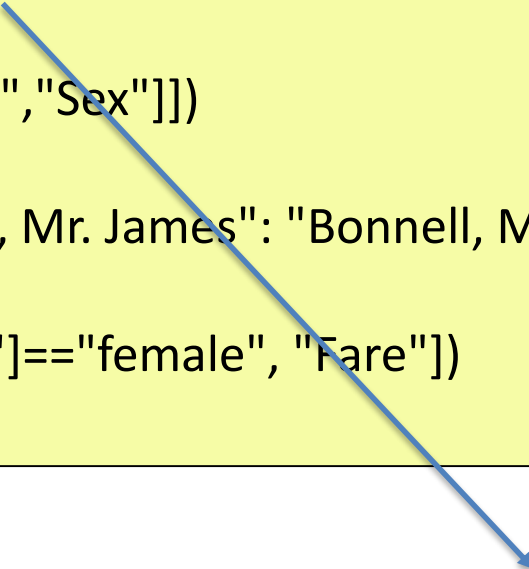
df = pd.read_csv("titanic.csv", index_col='Name')

print (df.loc["Moran, Mr. James", "Fare"])

print (df.loc[:, ["Fare", "Sex"]])

print (df.loc["Moran, Mr. James": "Bonnell, Miss. Elizabeth", ["Fare", "Sex"]])

print (df.loc[df["Sex"]=="female", "Fare"])
```



Retrieves the row indexed “Moran, Mr. James”, and the column “Fare”, which returns the value 8.4583

Selecting DataFrame rows and columns simultaneously - loc

```
import pandas as pd

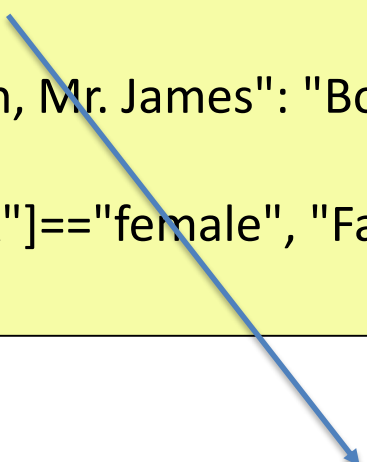
df = pd.read_csv("titanic.csv", index_col='Name')

print (df.loc["Moran, Mr. James", "Fare"])

print (df.loc[:, ["Fare", "Sex"]])

print (df.loc["Moran, Mr. James": "Bonnell, Miss. Elizabeth", ["Fare", "Sex"]])

print (df.loc[df["Sex"]=="female", "Fare"])
```



Accesses all rows but just the columns Fare and Sex

Selecting DataFrame rows and columns simultaneously - loc

```
import pandas as pd


df = pd.read_csv("titanic.csv", index_col='Name')

print (df.loc["Moran, Mr. James", "Fare"])

print (df.loc[:, ["Fare", "Sex"]])

print (df.loc["Moran, Mr. James": "Bonnell, Miss. Elizabeth", ["Fare", "Sex"]])

print (df.loc[df["Sex"]=="female", "Fare"])
```



Retrieves row starting from the index Moran, Mr. James up to and including the index Bonnell, Miss. Elizabeth but just the columns Fare and Sex

Selecting DataFrame rows and columns simultaneously - loc

```
import pandas as pd

df = pd.read_csv("titanic.csv", index_col='Name')

print (df.loc["Moran, Mr. James", "Fare"])

print (df.loc[:, ["Fare", "Sex"]])

print (df.loc["Moran, Mr. James": "Bonnell, Miss. Elizabeth", ["Fare", "Sex"]])

print (df.loc[df["Sex"]=="female", "Fare"])
```



This will return the Fare column for all rows that have Sex == female

Using iloc on a DataFrame

- We can also access specific rows using the iloc function.
- Again we specify an integer positional index (not label) and it returns the corresponding row.
- Allowed inputs are:
 - An integer, e.g. 5.
 - A list or array of integers, e.g. [4, 3, 0].
 - A slice object with ints, e.g. 1:7.
 - A Boolean array.

Using iloc on a DataFrame


```
import pandas as pd

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


# prints the entire first row
#(returned as a Series object)
print ( df.iloc[ 0 ] )
```

```
# selects the row with index 1
#and 3 (DataFrame obj)
print ( df.iloc[ [1, 3] ] )
```

```
# prints the index 4, 7 and 10
print ( df.iloc[ 4:11:3 ] )
```



```
PassengerId      1
Survived          0
Pclass            3
Name      Braund, Mr. Owen Harris
Sex              male
Age             22
SibSp            1
Parch            0
Ticket      A/5 21171
Fare           7.25
Cabin          NaN
Embarked         S
Name: 0, dtype: object
```



| | PassengerId | Survived | Pclass | ... | Fare | Cabin | Embarked |
|---|-------------|----------|--------|-----|---------|-------|----------|
| 1 | 2 | | 1 | ... | 71.2833 | C85 | C |
| 3 | 4 | 1 | 1 | ... | 53.1000 | C123 | S |

[2 rows x 12 columns]

Using iloc on a DataFrame

```
import pandas as pd
```

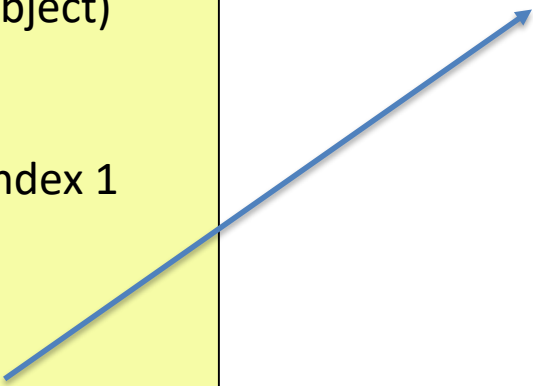
```
df = pd.read_csv("titanic.csv")
```

```
# prints the entire first row  
#(returned as a Series object)  
print (df.iloc[0])
```

```
# selects the row with index 1  
#and 3 (DataFrame obj)  
print (df.iloc[[1, 3]])
```

```
# prints the index 4, 7 and 10  
print (df.iloc[4:11:3])
```

| | PassengerId | Survived | Pclass | Name | Sex | \ | |
|----|-------------|----------|--------|---------------------------------|--------|-------|----------|
| 4 | 5 | 0 | 3 | Allen, Mr. William Henry | male | | |
| 7 | 8 | 0 | 3 | Palsson, Master. Gosta Leonard | male | | |
| 10 | 11 | 1 | 3 | Sandstrom, Miss. Marguerite Rut | female | | |
| | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
| 4 | 35.0 | 0 | 0 | 373450 | 8.050 | NaN | S |
| 7 | 2.0 | 3 | 1 | 349909 | 21.075 | NaN | S |
| 10 | 4.0 | 1 | 1 | PP 9549 | 16.700 | G6 | S |



Selecting DataFrame rows and columns simultaneously - iloc

- The method of using iloc is similar to how we selected data in NumPy
- The syntax is `data.iloc[<row selection>, <column selection>]`

```
print (df.iloc[:, 0])
```

```
print ( df.iloc[:, 0:3])
```

```
print ( df.iloc[:, [0,3]])
```

1. Print out the first column (Series object)
2. Print out the first, second and third column (DataFrame Object)
3. Print out the first and fourth columns (DataFrame Object)

Selecting DataFrame rows and columns simultaneously - iloc

- When using a **Boolean indexing** (masking) with loc it supports either a Boolean Series object or Boolean array.
- In contrast iloc only supports a **Boolean array** so you will have to convert the Boolean Series into an array using to_numpy()

```
import pandas as pd

df = pd.read_csv("titanic.csv",
index_col='Name')

bool_series = df["Sex"]=="female"

print (df.iloc[bool_series.to_numpy(), 2])
```

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))
```

```
24.0    30
22.0    27
18.0    26
19.0    25
Name: Age, dtype: int64
```

Missing Values

- ▶ It is common in real-world applications that some features are missing one or more values for various reasons. There could be an **error in the data collection process**, certain fields may have been **left blank in a survey, power outage**, etc.
- The following will return the number of missing NaN values in in each column of your dataframe.
- **df.isnull().sum()**

```
import pandas as pd
```

```
seriesA = pd.Series( ['Jim', 'Jean', 'Ted', 'Elizabeth'] )
```

```
seriesB = pd.Series( ['H1', 'H2.1', 'H1'] )
```

```
seriesC = pd.Series( ['BSc', 'MSc', 'PhD', 'BSc'] )
```

```
# Define a dictionary containing Students data
```

```
data = {'Name': seriesA,
```

```
       'Grade': seriesB,
```

```
       'Qualification': seriesC}
```

```
df = pd.DataFrame(data)
```

```
print (df)
```

```
print (df.isnull().sum())
```

| | Name | Grade | Qualification |
|---|-----------|-------|---------------|
| 0 | Jim | H1 | BSc |
| 1 | Jean | H2.1 | MSc |
| 2 | Ted | H1 | PhD |
| 3 | Elizabeth | NaN | BSc |

| | |
|---------------|---|
| Name | 0 |
| Grade | 1 |
| Qualification | 0 |

dtype: int64

Dealing with Missing Values - Removal

- ▶ One of the easiest ways to deal with missing values is to simply remove the corresponding features (columns) or rows from the dataset entirely.
 - ▶ **Rows**
 - ▶ **df.dropna()** will remove any rows that contain a missing value.
 - ▶ **df.dropna(subset=['A'])** only drop rows where missing values appear in a specific column in this case column A.
 - ▶ **df.dropna(thresh=3)** the parameter thresh specifies the number of non-NAN values (non missing values) that a row must have in order to be **retained**.
 - ▶ **Columns**
 - ▶ **df.dropna(axis = 1)** will drop **columns** that have at least one missing value
 - ▶ if you want to drop a column of a specific name you can call **df.drop(['ColumnName'], axis=1)**
- ▶ Each of the above will return a **separate copy** of the dataset with the new changes (if you want the changes to take place on the current dataframe then you can use the parameter **inplace=True**)

Missing Values

```
import pandas as pd

seriesA = pd.Series( ['Jim', 'Jean', 'Ted', 'Elizabeth'] )
seriesB = pd.Series( ['H1', 'H2.1', 'H1'] )
seriesC = pd.Series( ['BSc', 'MSc', 'PhD', 'BSc'] )

# Define a dictionary containing Students data
data = {'Name': seriesA,
        'Grade': seriesB,
        'Qualification': seriesC}

df = pd.DataFrame(data)
df.dropna(inplace=True)
print (df)
```

| | Name | Grade | Qualification |
|---|-----------|-------|---------------|
| 0 | Jim | H1 | BSc |
| 1 | Jean | H2.1 | MSc |
| 2 | Ted | H1 | PhD |
| 3 | Elizabeth | NaN | BSc |

| | Name | Grade | Qualification |
|---|------|-------|---------------|
| 0 | Jim | H1 | BSc |
| 1 | Jean | H2.1 | MSc |
| 2 | Ted | H1 | PhD |

Adding and Deleting Entire Columns

```
import pandas as pd
```

```
# Define a dictionary containing Students data
data = {'Name': ['Jim', 'Jean', 'Ted', 'Elizabeth'],
        'Grade': ['H1', 'H2.1', 'Pass', 'H1'],
        'Qualification': ['BSc', 'MSc', 'PhD', 'BSc']}
```

```
df = pd.DataFrame(data)
print (df)
```

```
age = [21, 34, 34, 53]
```

```
df["Age"] = age
print (df)
```

```
df.drop(["Qualification", "Grade"], axis = 1, inplace = True)
print (df)
```

| | Name | Grade | Qualification |
|---|-----------|-------|---------------|
| 0 | Jim | H1 | BSc |
| 1 | Jean | H2.1 | MSc |
| 2 | Ted | Pass | PhD |
| 3 | Elizabeth | H1 | BSc |

Adding and Deleting Columns

```
import pandas as pd
```

```
# Define a dictionary containing Students data  
data = {'Name': ['Jim', 'Jean', 'Ted', 'Elizabeth'],  
        'Grade': ['H1', 'H2.1', 'Pass', 'H1'],  
        'Qualification': ['BSc', 'MSc', 'PhD', 'BSc']}
```

```
df = pd.DataFrame(data)  
print (df)
```

```
age = [21, 34, 34, 53]
```

```
df["Age"] = age  
print (df)
```

```
df.drop(["Qualification", "Grade"], axis = 1, inplace = True)  
print (df)
```

| | Name | Grade | Qualification | Age |
|---|-----------|-------|---------------|-----|
| 0 | Jim | H1 | BSc | 21 |
| 1 | Jean | H2.1 | MSc | 34 |
| 2 | Ted | Pass | PhD | 34 |
| 3 | Elizabeth | H1 | BSc | 53 |

Adding and Deleting Columns

```
import pandas as pd

# Define a dictionary containing Students data
data = {'Name': ['Jim', 'Jean', 'Ted', 'Elizabeth'],
        'Grade': ['H1', 'H2.1', 'Pass', 'H1'],
        'Qualification': ['BSc', 'MSc', 'PhD', 'BSc']}

df = pd.DataFrame(data)
print (df)

age = [21, 34, 34, 53]

df["Age"] = age
print (df)

df.drop(["Qualification", "Grade"], axis = 1, inplace = True)
print (df)
```

| | Name | Age |
|---|-----------|-----|
| 0 | Jim | 21 |
| 1 | Jean | 34 |
| 2 | Ted | 34 |
| 3 | Elizabeth | 53 |

Summary

- NumPy 2D Arrays
 - [row, column] access
 - Slice operations [start:stop:step]
 - Performing operations of a specific axis (`np.sum(arr1, axis = 0)`)
 - Comparison Operators
 - Advanced Index (Boolean index with comparison operation, integer list)
 - Logical Operators
- Pandas
 - Series and DataFrame
 - Accessing Columns
 - Using label based indexing (`loc`) and integer based indexing (`iloc`)