MTU

Ollscoil Teicneolaíochta na Mumhan

Munster Technological University

Machine Learning

Lecture 4: Pandas

Contents

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

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 pandas as pd
Import numpy as np

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

s2 = pd.Series([1, 2, 3, 4, 5], index=['a','b','c','d','e'] )
# number of indices must match number of data points

print (s1)

print (s2)



allValues = s2.values

print (allValues)
print (type(allValues))
```

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

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

[1 2 3 4 5]
<class 'numpy.ndarray'>
```

Series

- You can use the **index to select specific items** from the Series.
  - The first print will print the value associated with the index 'USA'
  - The second uses **double square brackets** and prints a subset of the original series (**note it returns a series**)
  - The third prints the result of the condition statement, which produces a series consisting of Booleans.
  - The fourth print only print those entries that have an associated value greater than 4000000 (again note this returns a series object)

```
#WW2 casualties
ww2_cas = pd.Series( [8700000,4300000,3000000,2100000,400000],
index=['USSR','Germany','China','Japan','USA'])

print ( ww2_cas['USA'] )

print ( ww2_cas[['USA', 'Germany']] )

print ( ww2_cas>4000000 )

print (  ww2_cas[ ww2_cas>4000000 ]  )
```

```
400000

USA        400000
Germany    4300000
dtype: int64

USSR       True
Germany    True
China      False
Japan      False
USA        False
dtype: bool

USSR       8700000
Germany    4300000
dtype: int64
```

Series

- It is also very easy to change a value within a series (Similar to the syntax we use for adding a key value pair to a dictionary.

```
#WW2 casualties
ww2_cas =
pd.Series([8700000,4300000,3000000,2100000,400000],index=['USSR','Germany','Chin
a','Japan','USA'])

ww2_cas['USSR'] = 9000000

print (ww2_cas)

ww2_cas[ ww2_cas>4000000 ]  = 10000000

print (ww2_cas)

print (pd.unique((ww2_cas)))
```

```
USSR       9000000
Germany    4300000
China      3000000
Japan      2100000
USA         400000
dtype: int64


USSR       10000000
Germany    10000000
China       3000000
Japan       2100000
USA          400000
dtype: int64

[10000000  3000000  2100000
400000]
```

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).

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)***

- The data used to create the dataframe can be any one of a rang of data types.
- For example, it could be a 2D NumPy array.
- It could be a a dictionary of lists or a dictionary of Series objects.

- Using the columns parameter allows us to tell the constructor how we'd like the columns ordered.

Creating a DataFrame

- There are many ways of creating a Dataframe. For example below we create a Dataframe directly from a number of Series objects. Notice we pass in a dictionary where the values are Series.
- Notice that Pandas is flexible enough to handle empty values.

```
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

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.
- We can also specify column names when creating the dataframe.
- Note the columns name specified below are an optional argument.

```
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 (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

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 …

Titanic - Dataset

Available as .csv file on Canvas.

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

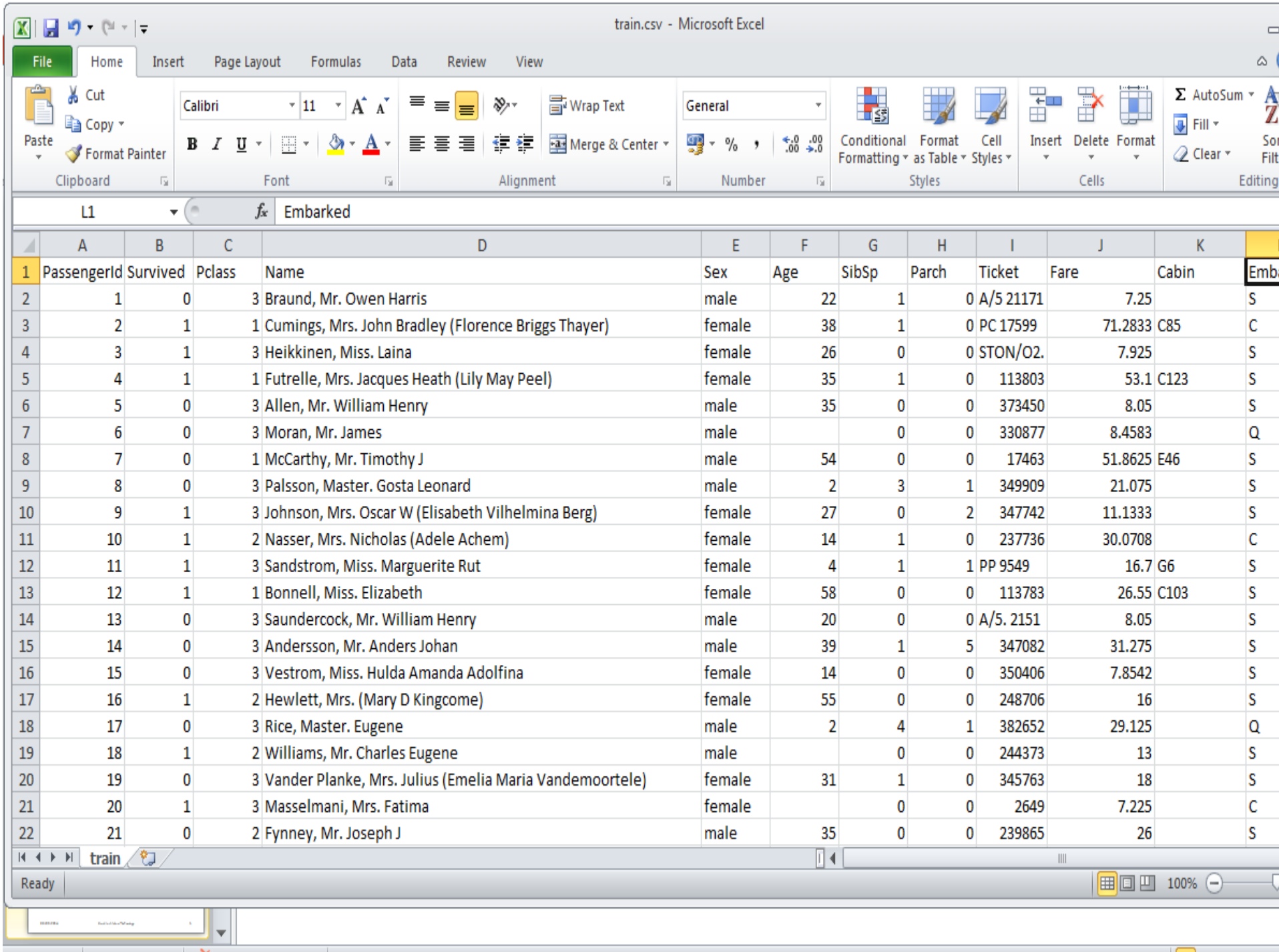
ticket Ticket Number

fare Passenger Fare

cabin Cabin

embarked Port of Embarkation
(C = Cherbourg; Q = Queenstown; S = Southampton)

https://www.kaggle.com/c/titanic



	A	B	C	D	E	F	G	H	I	J	K	
1	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emb
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	Saundercock, 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
- The head function returns a dataframe contains the first 5 rows of the original dataframe. Note we can provide an int argument to head to specify the number of rows we want to extract (note there is also a tail function that allows you to access the last 5 rows of the dataset).

```
Import pandas as pd

import numpy as np
import pandas as pd

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

```
   Survived  Pclass                                                Name  \
0         0       3                             Braund, Mr. Owen Harris
1         1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...
2         1       3                              Heikkinen, Miss. Laina
3         1       1        Futrelle, Mrs. Jacques Heath (Lily May Peel)
4         0       3                            Allen, Mr. William Henry

      Sex   Age  SibSp  Parch            Ticket     Fare Cabin Embarked
0    male  22.0      1      0         A/5 21171   7.2500   NaN        S
1  female  38.0      1      0          PC 17599  71.2833   C85        C
2  female  26.0      0      0  STON/O2. 3101282   7.9250   NaN        S
3  female  35.0      1      0            113803  53.1000  C123        S
4    male  35.0      0      0            373450   8.0500   NaN        S
```

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 (df.describe())
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Contents

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

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**

```
0      22.0
1      38.0
2      26.0
3      35.0
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
20     35.0
21     34.0
22     15.0
23     28.0
```

Accessing Rows and Individual Data Items

- We can access individual rows and values by specifying the column followed by the row index.

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

```
0    22.0
1    38.0
2    26.0
3    35.0
4    35.0
5     NaN
6    54.0
7     2.0
8    27.0
9    14.0
Name: Age, dtype: float64
```

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])
```

```
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**)
  - returns a dataframe

```
import pandas as pd
df = pd.read_csv("titanic.csv")
print (df[['Age', 'Fare']] [0:10] )
```

	Age	Fare
0	22.0	7.2500
1	38.0	71.2833
2	26.0	7.9250
3	35.0	53.1000
4	35.0	8.0500
5	NaN	8.4583
6	54.0	51.8625
7	2.0	21.0750
8	27.0	11.1333
9	14.0	30.0708

Pandas iloc

- The iloc methods is used for integer-location based indexing and is similar to what we used in NumPy
- The syntax is data.iloc[<row selection>, <column selection>]

```
import pandas as pd

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

print (df.iloc[0])

print (df.iloc[:, 0])

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

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

1. Print out the first row
2. Print out the first column
3. Print out the first, second and third column
4. Print out the first and fourth columns

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 = pd.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
```

Performing Operations

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

```
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

```
import pandas as pd

df = pd.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
15    False
16     True
17    False
18     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
26                                  Emir, Mr. Farred Chehab
27                           Fortune, Mr. Charles Alexander
```

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  ……..
```

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 passangers 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
>>>
```

Ln: 586 Col: 58

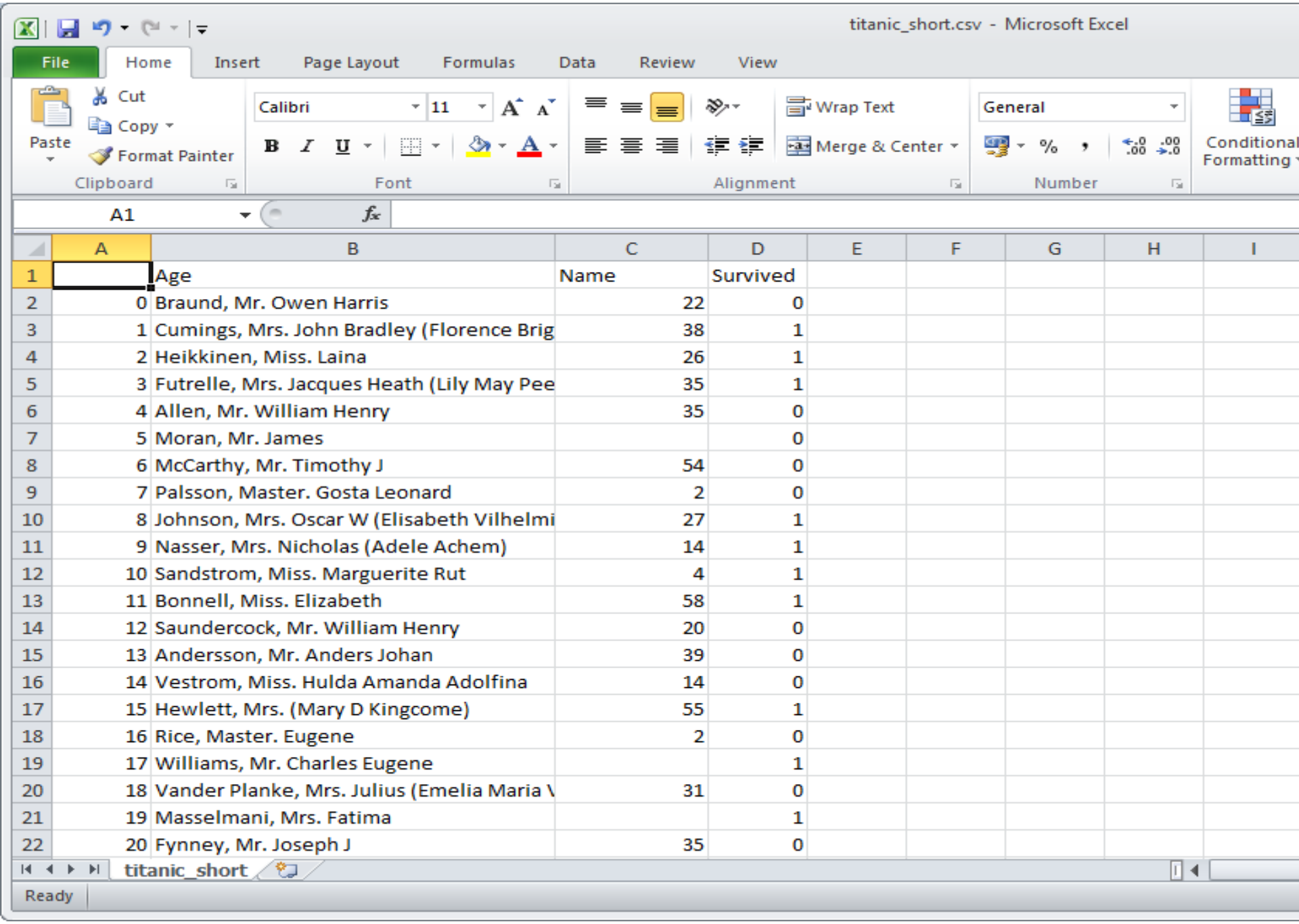
Storing the Data in a File

- 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')
```



	A	B	C	D	E	F	G	H	I
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					

Dropping Columns in a Dataframe

- Be able to remove a column from a dataframe is very useful in Pandas.
- We can the .drop function as shown below.
- Note we can drop more than one label at a time by specify a list

```
import pandas as pd

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

df = df.drop('Survived', axis=1)

print (df.columns.values)
```

```
['Pclass' 'Name' 'Sex' 'Age' 'SibSp'
'Parch' 'Ticket' 'Fare' 'Cabin'
 'Embarked']
```

Mapping

- A common pre-processing task, when using Scikit-Learn, is to encode categorical values as numerical values.
- We can easily do this using the map method available in Pandas.

```
import pandas as pd
import numpy as np

seriesA = pd.Series(['A', 'C', 'B'])
seriesB = pd.Series([21, 18, 19])
seriesC = pd.Series([4, 1, 1])
seriesD = pd.Series(['Computing', 'Biology', 'Biology'])

df = pd.DataFrame({'Grade' : seriesA,  'Age' : seriesB, 'DegreeYear' : seriesC,
          'Department' : seriesD})
print df

grade_mapping = {'F':0, 'D':1, 'C':2, 'B':3, 'A':4}

df['Grade'] = df['Grade'].map(grade_mapping)

print df
```

```
   Age  DegreeYear Department Grade
0   21           4  Computing     A
1   18           1    Biology     C
2   19           1  Chemistry     B

   Age  DegreeYear Department  Grade
0   21           4  Computing      4
1   18           1    Biology      2
2   19           1  Chemistry      3
```

```
import pandas as pd
import numpy as np

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' : series
print df
print df.isnull()
print df.isnull().sum()
```

```
        one     three          two
a  0.376060     NaN         0.111221
b  0.400020     0.164076    0.548106
c  0.507972     0.325337    0.137571
d       NaN     0.823270    0.816618

    one  three    two
a  False   True  False
b  False  False  False
c  False  False  False
d   True  False  False

one      1
three    1
two      0
dtype: int64
```

As we have seen Pandas can accommodate missing values and mark them as null. We need to be aware of null values in our data set. The call isnull().sum() on a dataframe will allow us to check the number of null values in each column

Dealing with Missing Values

- One of the easiest ways to deal with missing values is to simply remove the corresponding features (columns) or rows from the dataset entirely.

  - **df.dropna()** will remove any rows that contain a missing value.
  - Note we can run the dropna function on both a dataframe or series object.

  - **df.dropna(subset=['A'])** only drop rows where missing values appear in a specific column, in this case column A

```
import pandas as pd
import numpy as np

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)

newColOne = ( df['one'].dropna() )
print (newColOne)
print (type(newColOne))

print (df.dropna(subset=['one']))
```

```
 one     three      two
a  0.639647      NaN        0.663694
b  0.194625    0.601401   0.555457
c  0.521288    0.303332   0.152948
d       NaN       0.583547   0.613421

a    0.639647
b    0.194625
c    0.521288
Name: one, dtype: float64
<class 'pandas.core.series.Series'>

       one           three      two
a  0.639647      NaN        0.663694
b  0.194625    0.601401    0.555457
c  0.521288    0.303332    0.152948
```

Here we drop any row from our dataframe that has a null value in the column with label one

Thank you for your attention