NNSE 784
Advanced Analytics Methods

Instructor: F Doyle (CESTM L210)

MW 4:30 – 5:50, NFN 203

Slide Set #2
Pandas DataFrames

Note: some of the content in this slide set has been adapted from the “Python for Data Analysis” course by Katia Oleinik at Boston University

Outline for lecture

- Quick preliminary overview of statistics
  - Before we start pulling data into our computational environment, what is the goal?
    - Descriptive statistics
    - Inferential statistics
- Introduction to Pandas DataFrames
  - Describe the layout for a DataFrame and why we use them
  - Show how to load from a common source type
  - Show a quick example visualization with Seaborn
  - Go over some basic descriptive stats using the DataFrame
  - Demonstrate access and manipulation of DataFrame contents

Descriptive Statistics

- What's the point?
  - Describes the characteristics of a given dataset
  - Allows you to organize and summarize a potentially large amount of data and interpret for meaningful analysis
  - Does not necessarily aim to reach a conclusion or test a hypothesis
  - Can be applied to a full population or a subset (i.e., "sample")
- Three major categories
  1. **Frequency Distribution** – how often do particular values occur (histograms, pie charts, etc.)
  2. **Central Tendency** – what value is most representative (mean, median, mode)
  3. **Variability/Dispersion** – how are values distributed/spread (range, variance, standard deviation)

Inferential Statistics

- What is the point?
  - Helps to draw conclusions and make predictions based on a data set such as in the analysis of experimental results (e.g., “Does my treatment have an effect?”)
- Terminology
  - **Parameter** – descriptive measure computed from a population
  - **Statistic** - descriptive measure computed from a sample
- Some important types
  - **Regression analysis** – shows relationship between one or more independent variables and a dependent variable. Allows you to predict value of the dependent variable for different values of independent variables
  - **Hypothesis Tests** – used to compare populations or assess relationships between variables using samples.
  - **Confidence Intervals** – a main goal in inferential statistics is to estimate population parameters based on sample data. Statistical calculations that consider variability, uncertainty and sampling error allow for an interval estimate to be produced. A confidence interval is a range of values within which the actual population parameter can be expected to fall with some associated probability.

Loading Common Data Science Libraries

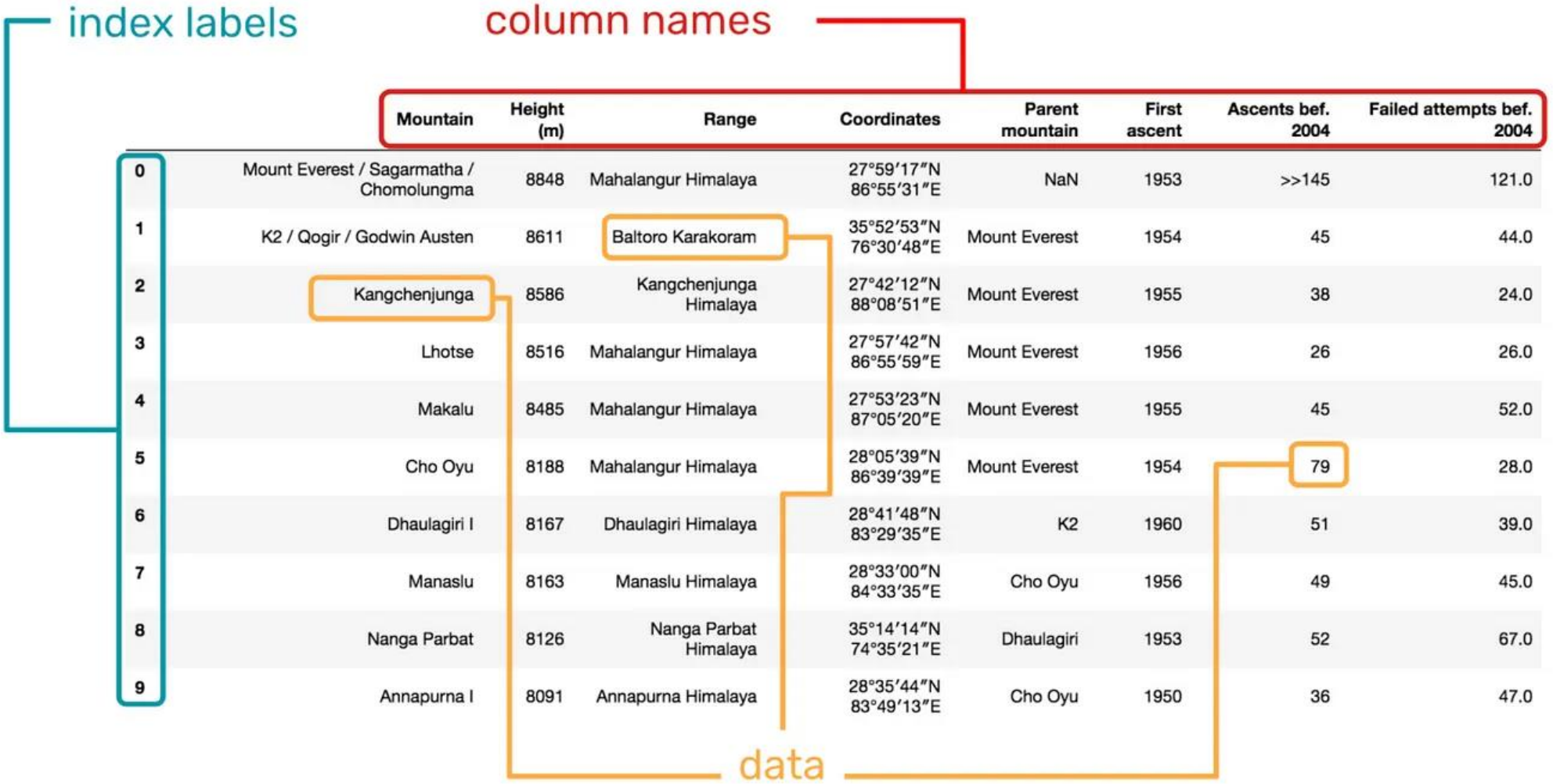
```
#Import Python Libraries
import numpy as np
import scipy as sp
import pandas as pd
import matplotlib as mpl
import seaborn as sns
```

Note: the abbreviations for libraries shown here are those that you will commonly see used.

Pandas DataFrame

- The DataFrame is a “rows & columns” data structure conceptually similar to a spreadsheet or database table and is the main data type used in pandas
- Columns generally represent specific variables
- Rows generally represent a set of related observations for those variables (e.g., a single experiment, etc.)
- DataFrames are the primary data type used for data analysis in a multitude of Python tools, including visualization as well as numerous Machine Learning models that we will investigate later in the course
- Other languages, such as R and Scala also use DataFrames for the same purposes

Pandas DataFrame - example



	Mountain	Height (m)	Range	Coordinates	Parent mountain	First ascent	Ascents bef. 2004	Failed attempts bef. 2004
0	Mount Everest / Sagarmatha / Chomolungma	8848	Mahalangur Himalaya	27°59′17″N 86°55′31″E	NaN	1953	>>145	121.0
1	K2 / Qogir / Godwin Austen	8611	Baltoro Karakoram	35°52′53″N 76°30′48″E	Mount Everest	1954	45	44.0
2	Kangchenjunga	8586	Kangchenjunga Himalaya	27°42′12″N 88°08′51″E	Mount Everest	1955	38	24.0
3	Lhotse	8516	Mahalangur Himalaya	27°57′42″N 86°55′59″E	Mount Everest	1956	26	26.0
4	Makalu	8485	Mahalangur Himalaya	27°53′23″N 87°05′20″E	Mount Everest	1955	45	52.0
5	Cho Oyu	8188	Mahalangur Himalaya	28°05′39″N 86°39′39″E	Mount Everest	1954	79	28.0
6	Dhaulagiri I	8167	Dhaulagiri Himalaya	28°41′48″N 83°29′35″E	K2	1960	51	39.0
7	Manaslu	8163	Manaslu Himalaya	28°33′00″N 84°33′35″E	Cho Oyu	1956	49	45.0
8	Nanga Parbat	8126	Nanga Parbat Himalaya	35°14′14″N 74°35′21″E	Dhaulagiri	1953	52	67.0
9	Annapurna I	8091	Annapurna Himalaya	28°35′44″N 83°49′13″E	Cho Oyu	1950	36	47.0

Source: https://medium.com/epfl-extension-school/selecting-data-from-a-pandas-dataframe-53917dc39953

Reading data into a pandas DataFrame

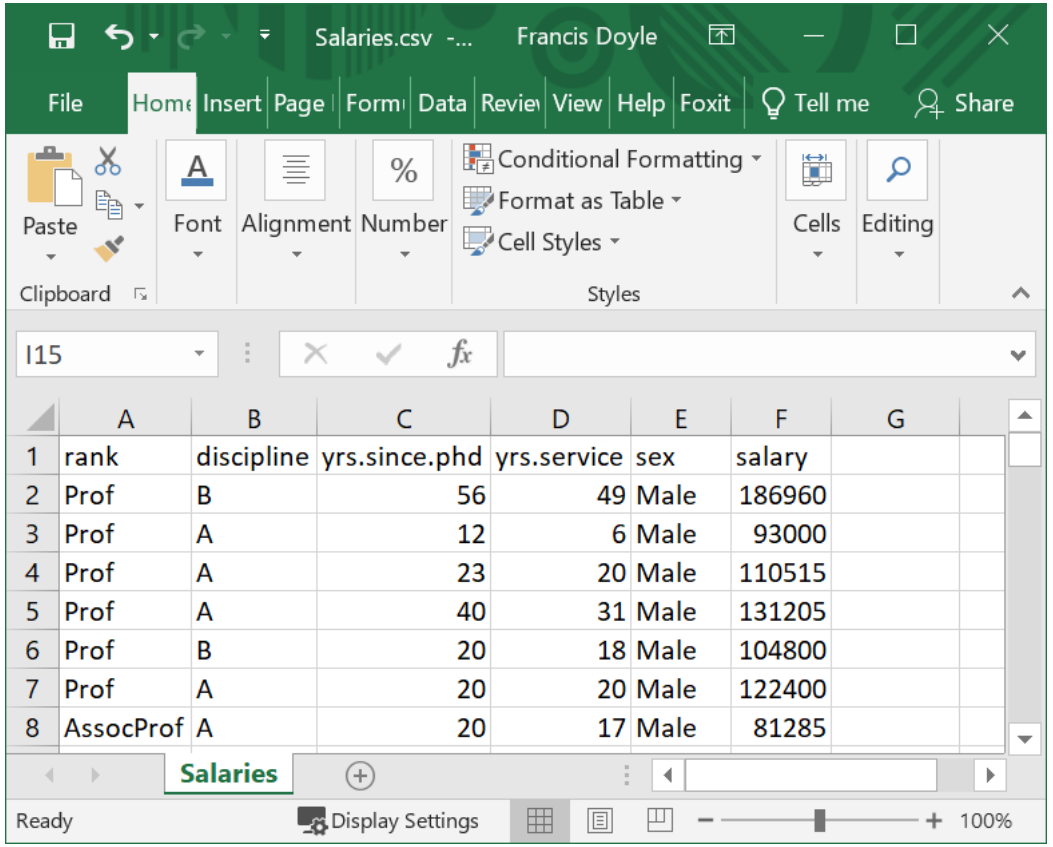
In [2]:
```
#it is common to use the 'as' keyword when importing libraries to provide a more user friendly reference
#in the following code
import pandas as pd
```

In [3]:
```
#the read_csv function has many optional arguments to refine the import process as needed
df = pd.read_csv("./Salaries.csv")
```

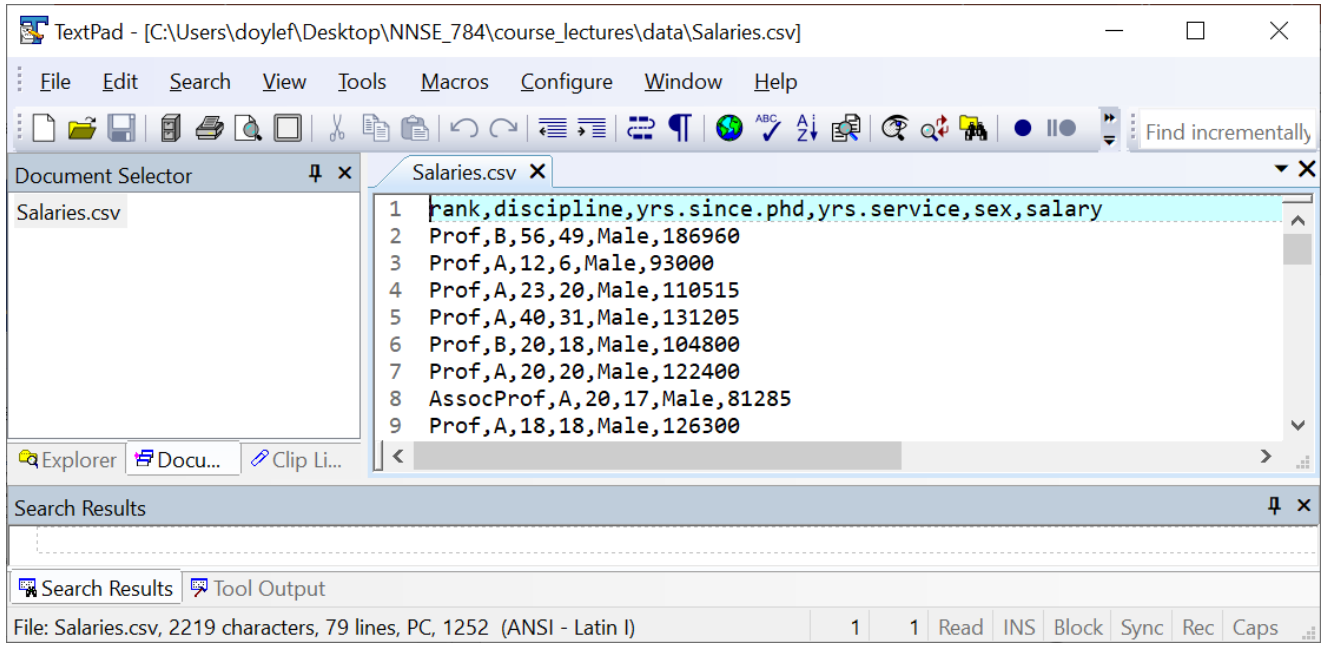
There is a number of pandas commands to read other data formats:

```
pd.read_excel('myfile.xlsx',sheet_name='Sheet1', index_col=None, na_values=['NA'])
pd.read_stata('myfile.dta')
pd.read_sas('myfile.sas7bdat')
pd.read_hdf('myfile.h5','df')
```

.csv files – “comma separated values”
ASCII text file – can be produced or manipulated with a spreadsheet program, text editor, etc…



	A	B	C	D	E	F	G
1	rank	discipline	yrs.since.phd	yrs.service	sex	salary	
2	Prof	B	56	49	Male	186960	
3	Prof	A	12	6	Male	93000	
4	Prof	A	23	20	Male	110515	
5	Prof	A	40	31	Male	131205	
6	Prof	B	20	18	Male	104800	
7	Prof	A	20	20	Male	122400	
8	AssocProf	A	20	17	Male	81285	



Exploring a DataFrame

```
In [4]: df.head()
```

Out[4]:

	rank	discipline	yrs.since.phd	yrs.service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800

```
In [5]: df.head(8)
```

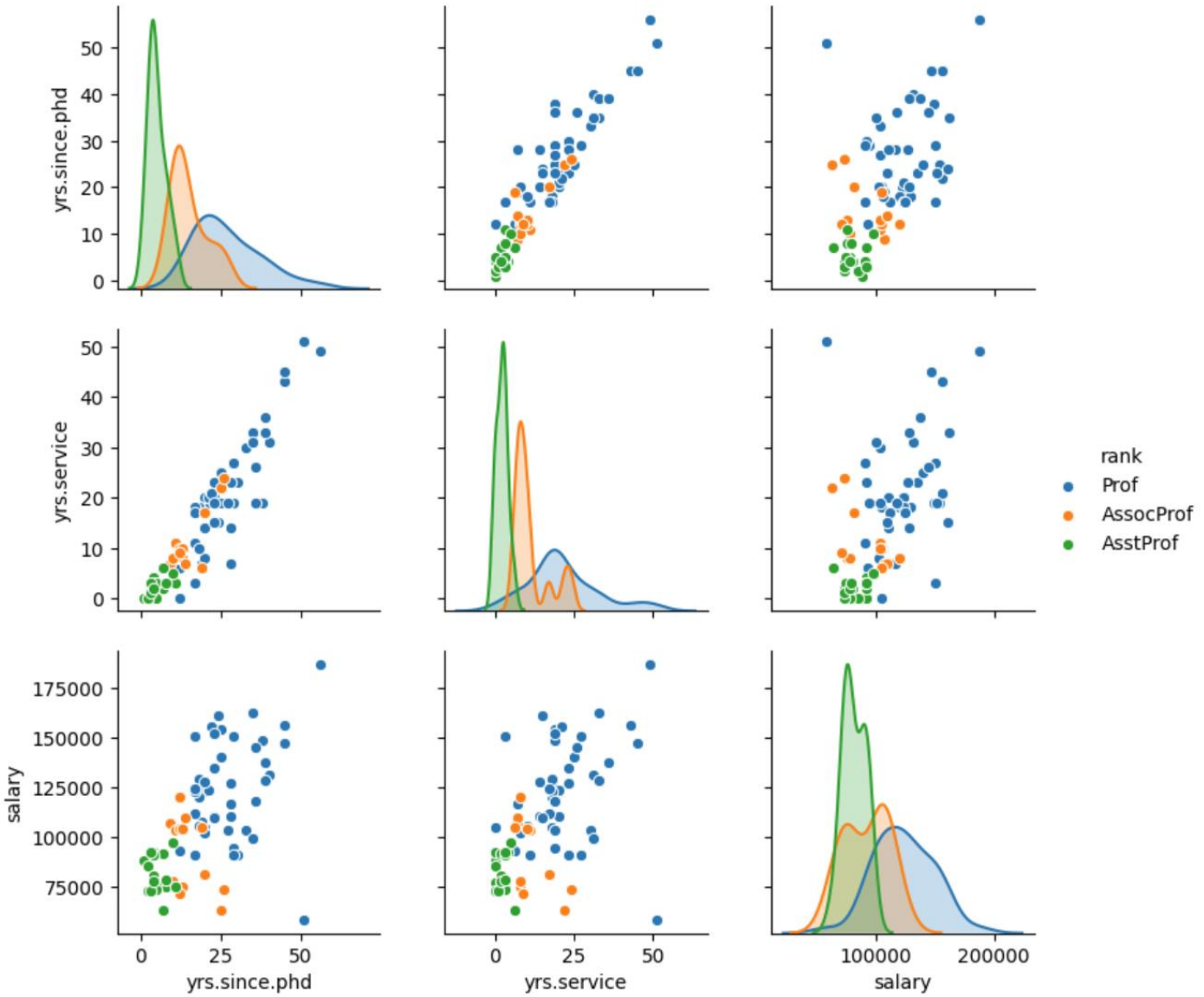
Out[5]:

	rank	discipline	yrs.since.phd	yrs.service	sex	salary
0	Prof	B	56	49	Male	186960
1	Prof	A	12	6	Male	93000
2	Prof	A	23	20	Male	110515
3	Prof	A	40	31	Male	131205
4	Prof	B	20	18	Male	104800
5	Prof	A	20	20	Male	122400
6	AssocProf	A	20	17	Male	81285
7	Prof	A	18	18	Male	126300

A Quick Example of Data Visualization and Exploratory Data Analysis

```
In [17]: import seaborn as sns
         sns.pairplot(df, hue='rank')
```

```
Out[17]: <seaborn.axisgrid.PairGrid at 0x21c5595dca0>
```



Data types in the DataFrame

Pandas dtype	Python type	NumPy type	Usage
object	str or mixed	string_, unicode_, mixed types	Text or mixed numeric and non-numeric values
int64	int	int_, int8, int16, int32, int64, uint8, uint16, uint32, uint64	Integer numbers
float64	float	float_, float16, float32, float64	Floating point numbers
bool	bool	bool_	True/False values
datetime64	NA	datetime64[ns]	Date and time values
timedelta[ns]	NA	NA	Differences between two datetimes
category	NA	NA	Finite list of text values

Pandas DataFrame attributes

Reminder - Python objects have *attributes* and *methods*.

df.attribute	description
dtypes	list the types of the columns
columns	list the column names
axes	list the row labels and column names
ndim	number of dimensions
size	number of elements
shape	return a tuple representing the dimensionality
values	numpy representation of the data

Pandas DataFrame methods

… *parentheses denote these are methods.*
All attributes and methods can be listed with a *dir()* function: `dir(df)`

df.method()	description
head([n]), tail([n])	first/last n rows
describe()	generate descriptive statistics (for numeric columns only)
max(), min()	return max/min values for all numeric columns
mean(), median()	return mean/median values for all numeric columns
std()	standard deviation
sample([n])	returns a random sample of the data frame
dropna()	drop all the records with missing values

Quick Examination of a DataFrame's Layout

```
In [5]: df.dtypes

Out[5]: rank              object
        discipline        object
        yrs.since.phd      int64
        yrs.service        int64
        sex               object
        salary             int64
        dtype: object

In [9]: df.columns

Out[9]: Index(['rank', 'discipline', 'yrs.since.phd', 'yrs.service', 'sex', 'salary'],
        dtype='object')

In [11]: df.size

Out[11]: 468
```

Quick Descriptive Statistics Summary of a DataFrame

```
In [14]: df.describe()
```

Out[14]:

	yrs.since.phd	yrs.service	salary
count	78.000000	78.000000	78.000000
mean	19.705128	15.051282	108023.782051
std	12.498425	12.139768	28293.661022
min	1.000000	0.000000	57800.000000
25%	10.250000	5.250000	88612.500000
50%	18.500000	14.500000	104671.000000
75%	27.750000	20.750000	126774.750000
max	56.000000	51.000000	186960.000000

The unique() Method

```
In [10]: df['rank'].unique()

Out[10]: array(['Prof', 'AssocProf', 'AsstProf'], dtype=object)
```

We use the unique() function on a specific dataframe column (a pandas series object) to examine the set of distinct values represented in the column.

Selecting a column in a Data Frame

Method 1: Subset the data frame using column name:

```
df['sex']
```

Method 2: Use the column name as an attribute:

```
df.sex
```

Note: there is a *rank* attribute in pandas data frames, so to select a column with a name "rank" we should use method 1.

Data Frames *groupby* method

Using "group by" method we can:

- Split the data into groups based on some criteria
- Calculate statistics (or apply a function) to each group
- Similar to some dplyr() function capabilities in R

In [ ]:
```
#Group data using rank
df_rank = df.groupby(['rank'])
```

In [ ]:
```
#Calculate mean value for each numeric column per each group
df_rank.mean()
```

rank	phd	service	salary
AssocProf	15.076923	11.307692	91786.230769
AsstProf	5.052632	2.210526	81362.789474
Prof	27.065217	21.413043	123624.804348

Data Frames *groupby* method

Once groupby object is created we can calculate various statistics for each group:

```
In [ ]: #Calculate mean salary for each professor rank:
        df.groupby('rank')[['salary']].mean()
```

rank	salary
AssocProf	91786.230769
AsstProf	81362.789474
Prof	123624.804348

Note: If single brackets are used to specify the column (e.g. salary), then the output is Pandas Series object. When double brackets are used the output is a Data Frame

Data Frames *groupby* method

groupby performance notes:

- no grouping/splitting occurs until it's needed. Creating the *groupby* object only verifies that you have passed a valid mapping for group key (i.e., 'rank')
- by default the group keys are sorted during the *groupby* operation. You may want to pass sort=False for potential speedup:

In [ ]:
```
#Calculate mean salary for each professor rank:
df.groupby(['rank'], sort=False)[['salary']].mean()
```

Data Frame: filtering

To subset the data we can apply Boolean indexing. This indexing is commonly known as a filter. For example if we want to subset the rows in which the salary value is greater than $120K:

In [ ]:
```
#Select only the rows with a salary greater than 120000
df_sub = df[ df['salary'] > 120000 ]
```

Any Boolean operator can be used to subset the data:

> greater; >= greater or equal;

< less; <= less or equal;

== equal; != not equal;

In [ ]:
```
#Select only those rows that contain female professors:
df_f = df[ df['sex'] == 'Female' ]
```

Data Frame: filtering

We can apply a more complicated filter with a combination of boolean expressions. Note the use of the parantheses surrounding the individual expressions.

In [ ]:
```
#example of a more complex multi boolean expression filter to find some
"middle range"
df_sub =  df[(df['salary'] > 60000) & (df['salary'] < 120000)]
```

Data Frames: Subsetting

There are a number of ways to subset the Data Frame:

- one or more columns
- one or more rows
- a subset of rows and columns

Rows and columns can be selected by their position or label

Data Frames: Subsetting (columns)

When selecting one column, it is possible to use single set of brackets, but the resulting object will be a Series (not a DataFrame):

In [ ]:
```
#Select column salary:
df['salary']
```

When we need to select more than one column and/or make the output to be a DataFrame, we should use double brackets:

In [ ]:
```
#Select column salary:
df[['rank','salary']]
```

Data Frames: Selecting rows (slicing)

If we need to select a range of rows, we can specify the range using ":"

In [ ]:
```
#Select rows by their position:
df[10:20]
```

Notice that the first row has a position 0, and the last value in the range is omitted:
So for 0:10 range the first 10 rows are returned with the positions starting with 0 and ending with 9

Data Frames: method loc

If we need to select a range of rows and/or columns, using their labels we can use method loc:

In [ ]:
```
#Select rows by their labels:
df_sub.loc[10:20,['rank','sex','salary']]
```

Out[ ]:

	rank	sex	salary
10	Prof	Male	128250
11	Prof	Male	134778
13	Prof	Male	162200
14	Prof	Male	153750
15	Prof	Male	150480
19	Prof	Male	150500

Data Frames: method iloc

If we need to select a range of rows and/or columns, using their positions we can use method iloc:

In [ ]:
```
#Select rows and columns by their positions:
df_sub.iloc[10:20,[0, 3, 4, 5]]
```

Out[ ]:

	rank	service	sex	salary
26	Prof	19	Male	148750
27	Prof	43	Male	155865
29	Prof	20	Male	123683
31	Prof	21	Male	155750
35	Prof	23	Male	126933
36	Prof	45	Male	146856
39	Prof	18	Female	129000
40	Prof	36	Female	137000
44	Prof	19	Female	151768
45	Prof	25	Female	140096

Data Frames: method iloc (summary)

```
df.iloc[0]  # First row of a data frame
df.iloc[i]  #(i+1)th row
df.iloc[-1] # Last row
```

```
df.iloc[:, 0]  # First column
df.iloc[:, -1] # Last column
```

```
df.iloc[0:7]        #First 7 rows
df.iloc[:, 0:2]     #First 2 columns
df.iloc[1:3, 0:2]   #Second through third rows and first 2 columns
df.iloc[[0,5], [1,3]]  #1st and 6th rows and 2nd and 4th columns
```

Data Frames: Sorting

We can sort the data by a value in the column. By default the sorting will occur in ascending order and a new data frame is returned.

In [ ]:
```
# Create a new data frame from the original sorted by the column Salary
df_sorted = df.sort_values( by ='service')
df_sorted.head()
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
55	AsstProf	A	2	0	Female	72500
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000

Data Frames: Sorting

We can sort the data using 2 or more columns:

In [ ]:
```
df_sorted = df.sort_values( by =['service', 'salary'], ascending = [True, False])
df_sorted.head(10)
```

Out[ ]:

	rank	discipline	phd	service	sex	salary
52	Prof	A	12	0	Female	105000
17	AsstProf	B	4	0	Male	92000
12	AsstProf	B	1	0	Male	88000
23	AsstProf	A	2	0	Male	85000
43	AsstProf	B	5	0	Female	77000
55	AsstProf	A	2	0	Female	72500
57	AsstProf	A	3	1	Female	72500
28	AsstProf	B	7	2	Male	91300
42	AsstProf	B	4	2	Female	80225
68	AsstProf	A	4	2	Female	77500