EDA Exercise

1.INTRODUCTION TO EDA

- ❖ Exploratory Data Analysis refers to the critical process of performing initial ***investigations*** on data so as to discover ***patterns***, to spot ***anomalies***, to test ***hypothesis*** and to check ***assumptions*** with the help of summary statistics and graphical representations.

- ❖ It is a good practice to understand the data first and try to gather as many ***insights*** from it.

2. IMPORTANCE OF EDA

- Identifying the most **important variables/features** in your dataset.
- Testing a **hypothesis** or checking assumptions related to the dataset.
- To check the **quality of data** for further processing and cleaning.
- Deliver **data-driven insights** to business stakeholders.
- Verify expected **relationships** actually exist in the data.
- To find **unexpected structure** or insights in the data.

Data Types

Ordinal
Binary
Nominal
Categorical
Variable
Numerical
Discrete
Continuous

Structured Data Types

Categorical - This is any data that isn't a number.

- Ordinal - have a set of order e.g. rating happiness on a scale of 1-10.
- Binary - have only two values .e.g. Male or Female
- Nominal - no set of order e.g. Countries

Numerical – Data inform of numbers

- Continuous - numbers that don't have a logical end to them e.g heights
- Discrete - have a logical end to them e.g. days in the month

Discrete and continuous data

- Discrete data is information that can only take certain values.
- For example:
  - The number of each type of treatment a salon needs to schedule for the week,
  - The number of children attending a nursery each day
  - **the results of rolling 2 dice**
- This type of data is often represented using tally charts, bar charts or pie charts.

- Continuous data is data that can take any value.
- For Example:
  - Height, weight, temperature and length are all examples of continuous data.

Line Graphs in Matplotlib

Basic Line Plot:

Line graphs are helpful for visualizing how a variable changes over time.

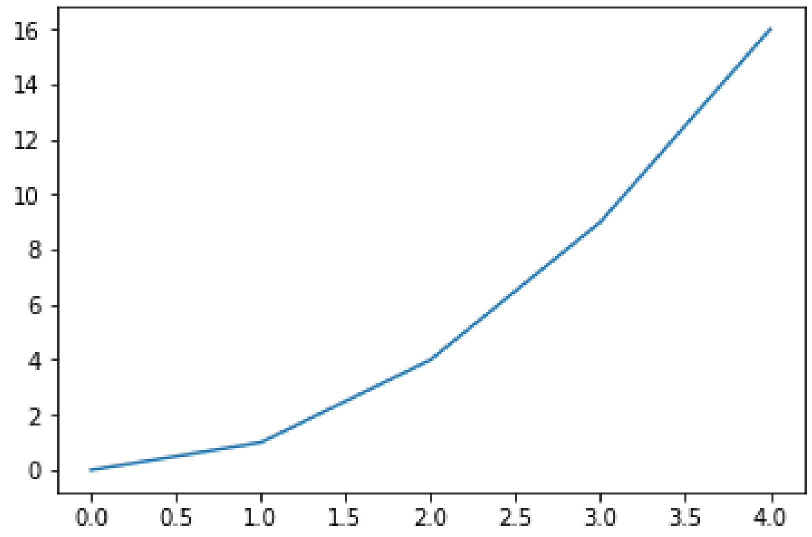
Some possible data that would be displayed with a line graph:

- average prices of gasoline over the past decade
- weight of an individual over the past couple of months
- average temperature along a line of longitude over different latitudes

Using Matplotlib methods, the following code will create a simple line graph using .plot() and display it using .show():

```
x_values = [0, 1, 2, 3, 4]
y_values = [0, 1, 4, 9, 16]
plt.plot(x_values, y_values)
plt.show()
```

plt.plot(x_values, y_values) will create the line graph



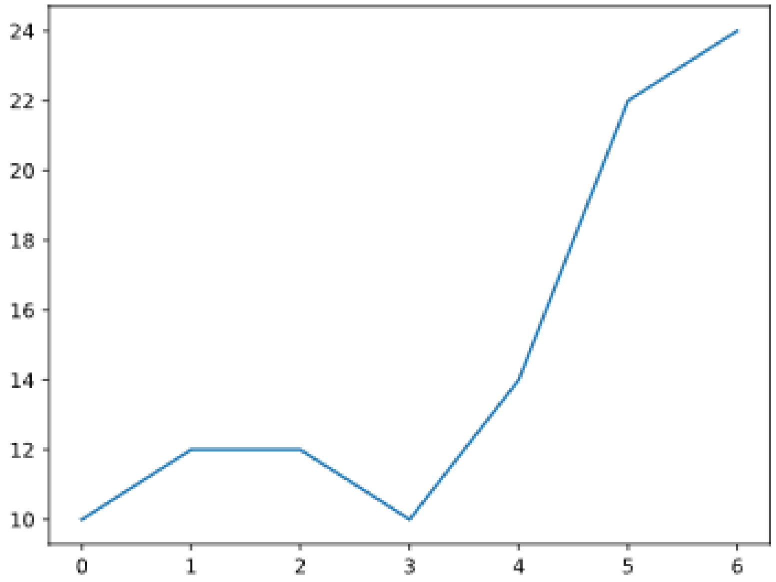
Line Graphs in Matplotlib

Example: We are going to make a simple graph representing someone's spending on lunch over the past week.

First, define two lists, days and money_spent, that contain the following integers:

Days	Money Spent
0	10
1	12
2	12
3	10
4	14
5	22
6	24

```
import codecademylib
from matplotlib import pyplot as plt

days = range(7)  # days = [0, 1, 2, 3, 4,5, 6]

money_spent = [10, 12, 12, 10, 14, 22, 24]

plt.plot(days, money_spent)

plt.show()
```



Plot days on the x-axis and money_spent on the y-axis using plt.plot().

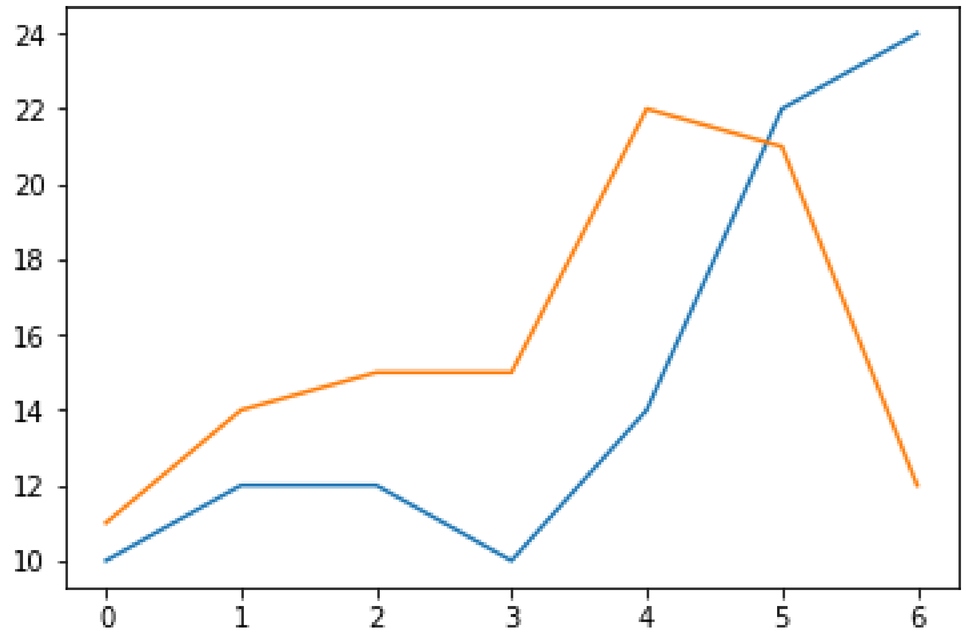
Multiple Line Graphs in Matplotlib

We can also have multiple line plots displayed on the same set of axes.

This can be very useful if we want to compare two datasets with the same scale and axis categories

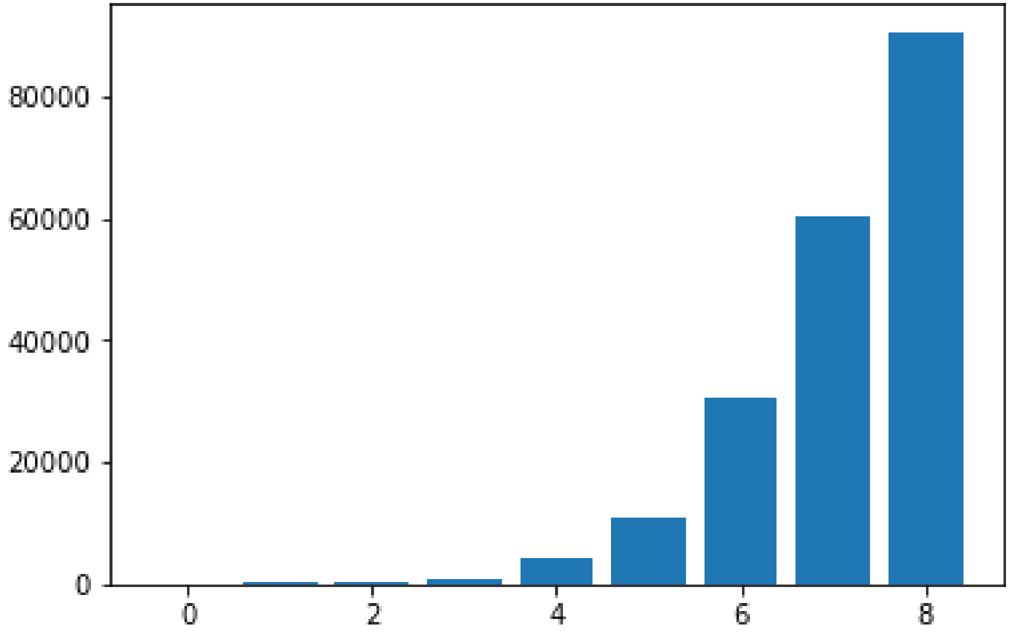
Matplotlib will automatically place the two lines on the same axes and give them different colors if you call plt.plot() twice

```
# Days of the week:
days = [0, 1, 2, 3, 4, 5, 6]
# Your Money:
money_spent = [10, 12, 12, 10, 14, 22, 24]
# Your Friend's Money:
money_spent_2 = [11, 14, 15, 15, 22, 21, 12]
# Plot your money:
plt.plot(days, money_spent)
# Plot your friend's money:
plt.plot(days, money_spent_2)
# Display the result:
plt.show()
```



Simple Bar Chart

```
days_in_year = [88, 225, 365, 687, 4333, 10756, 30687,
60190, 90553]
plt.bar(range(len(days_in_year)), days_in_year)
plt.show()
```



Exercise 1

- We are going to help the cafe MatplotSip analyze some of the sales data they have been collecting. In script.py, we have included a list of drink categories and a list of numbers representing the sales of each drink over the past month.

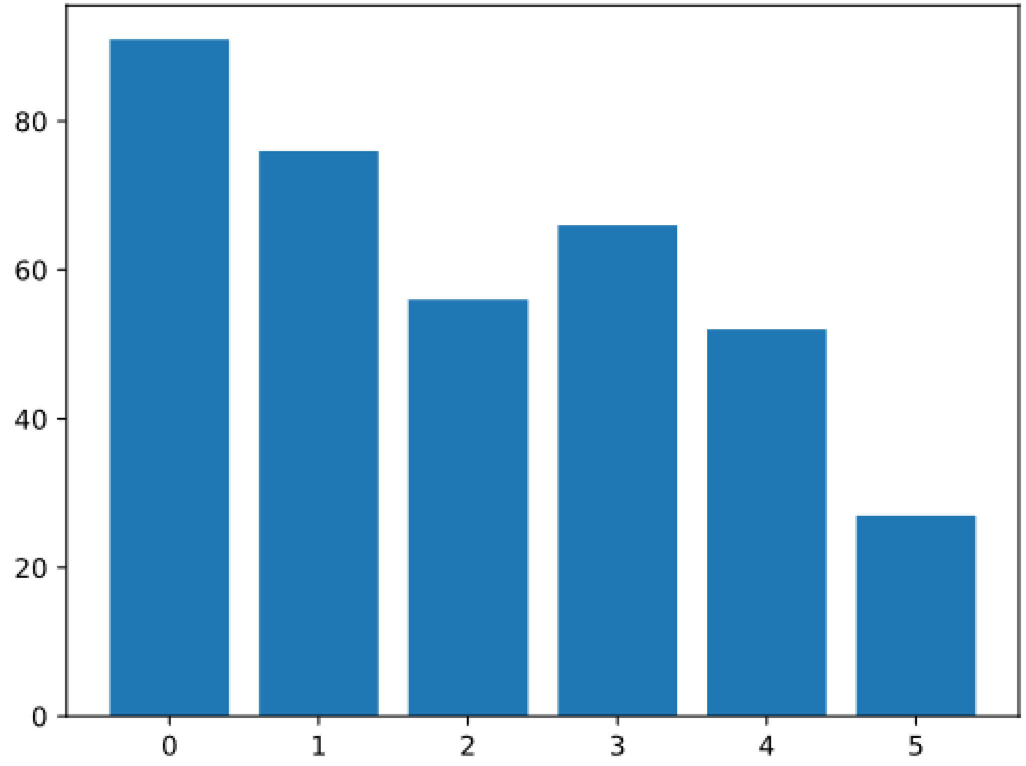
- Use plt.bar to plot numbers of drinks sold on the y-axis. The x-values of the graph should just be the list [0, 1 ... , n-1], where n is the number of categories (drinks) we are plotting. So at x=0, we'll have the number of cappuccinos sold.

- Show the plot and examine it. At this point, we can't tell which bar corresponds to which drink, so this chart is not very helpful. We'll fix this in the next section.

```
from matplotlib import pyplot as plt

drinks = ["cappuccino", "latte", "chai", "americano", "mocha", "espresso"]
sales =  [91, 76, 56, 66, 52, 27]
```

Solution:

```
import codecademylib
from matplotlib import pyplot as plt

drinks = ["cappuccino", "latte", "chai", "americano", "mocha", "espresso"]
sales =  [91, 76, 56, 66, 52, 27]

plt.bar(range(len(sales)), sales)
plt.show()
```



Exercise:

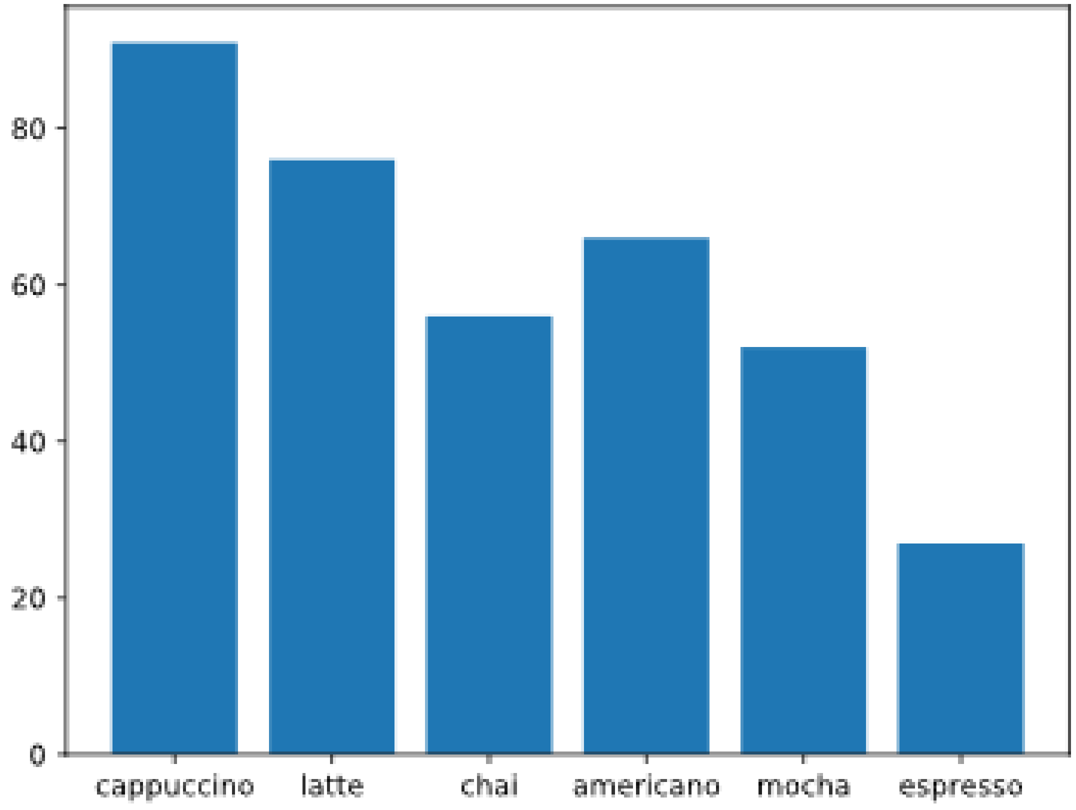
- The list drinks represents the drinks sold at MatplotSip. We are going to set x-tick labels on the chart you made with plt.bar in the last exercise.

- First, create the axes object for the plot and store it in a variable called ax.
- Set the x-axis ticks to be the numbers from 0 to the length of drinks.
- Use the strings in the drinks list for the x-axis ticks of the plot you made with plt.bar.

```
from matplotlib import pyplot as plt

drinks = ["cappuccino", "latte", "chai", "americano", "mocha", "espresso"]
sales =  [91, 76, 56, 66, 52, 27]

plt.bar(range(len(drinks)), sales)

#create your ax object here
plt.show()
```

Solution:

```
from matplotlib import pyplot as plt

drinks = ["cappuccino", "latte", "chai", "americano", "mocha",
"espresso"]
sales =  [91, 76, 56, 66, 52, 27]

plt.bar(range(len(drinks)), sales)

ax = plt.subplot()
ax.set_xticks(range(6))
ax.set_xticklabels(drinks)
plt.show()
```



Python for Data Analysis

Research Computing Services

Katia Oleinik (koleinik@bu.edu)

BOSTON UNIVERSITY

Tutorial Content

Overview of Python Libraries for Data Scientists

Reading Data; Selecting and Filtering the Data; Data manipulation, sorting, grouping, rearranging

Plotting the data

Descriptive statistics

Inferential statistics

Python Libraries for Data Science

Many popular Python toolboxes/libraries:

- NumPy
- SciPy
- Pandas
- SciKit-Learn

All these libraries are installed on the SCC

Visualization libraries

- matplotlib
- Seaborn

and many more …

Python Libraries for Data Science

NumPy:

- introduces objects for multidimensional arrays and matrices, as well as functions that allow to easily perform advanced mathematical and statistical operations on those objects
- provides vectorization of mathematical operations on arrays and matrices which significantly improves the performance
- many other python libraries are built on NumPy

Link: http://www.numpy.org/

Python Libraries for Data Science

SciPy:

- collection of algorithms for linear algebra, differential equations, numerical integration, optimization, statistics and more

- part of SciPy Stack

- built on NumPy

Link: https://www.scipy.org/scipylib/

Python Libraries for Data Science

Pandas:

- adds data structures and tools designed to work with table-like data (similar to Series and Data Frames in R)

- provides tools for data manipulation: reshaping, merging, sorting, slicing, aggregation etc.

- allows handling missing data

Link: http://pandas.pydata.org/

Python Libraries for Data Science

SciKit-Learn:

- provides machine learning algorithms: classification, regression, clustering, model validation etc.

- built on NumPy, SciPy and matplotlib

Link: http://scikit-learn.org/

Python Libraries for Data Science

matplotlib:

- python 2D plotting library which produces publication quality figures in a variety of hardcopy formats
- a set of functionalities similar to those of MATLAB
- line plots, scatter plots, barcharts, histograms, pie charts etc.
- relatively low-level; some effort needed to create advanced visualization

Link: https://matplotlib.org/

Python Libraries for Data Science

Seaborn:

- based on matplotlib

- provides high level interface for drawing attractive statistical graphics

- Similar (in style) to the popular ggplot2 library in R

Link: https://seaborn.pydata.org/

Login to the Shared Computing Cluster

- Use your SCC login information if you have SCC account

- If you are using tutorial accounts see info on the blackboard

Note: Your password will not be displayed while you enter it.

Selecting Python Version on the SCC

```
# view available python versions on the SCC
[scc1 ~] module avail python

# load python 3 version
[scc1 ~] module load python/3.6.2
```

Download tutorial notebook

On the Shared Computing Cluster

```
[scc1 ~] cp /project/scv/examples/python/data_analysis/dataScience.ipynb .
```

On a local computer save the link:

http://rcs.bu.edu/examples/python/data_analysis/dataScience.ipynb

Start Jupyter nootebook

On the Shared Computing Cluster

```
[scc1 ~] jupyter notebook
```

Loading Python Libraries

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

Press `Shift+Enter` to execute the *jupyter* cell

Reading data using pandas

```
In [ ]: #Read csv file
        df = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/Salaries.csv")
```

Note: The above command has many optional arguments to fine-tune the data import process.

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')
```

Exploring data frames

In [3]:
```
#List first 5 records
df.head()
```

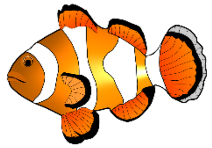
Out[3]:

	rank	discipline	phd	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

Hands-on exercises

- ✓ Try to read the first 10, 20, 50 records;
- ✓ Can you guess how to view the last few records;

Hint:



Data Frame data types

Pandas Type	Native Python Type	Description
object	string	The most general dtype. Will be assigned to your column if column has mixed types (numbers and strings).
int64	int	Numeric characters. 64 refers to the memory allocated to hold this character.
float64	float	Numeric characters with decimals. If a column contains numbers and NaNs(see below), pandas will default to float64, in case your missing value has a decimal.
datetime64, timedelta[ns]	N/A (but see the datetime module in Python's standard library)	Values meant to hold time data. Look into these for time series experiments.

Data Frame data types

In [4]:
```
#Check a particular column type
df['salary'].dtype
```

Out[4]: `dtype('int64')`

In [5]:
```
#Check types for all the columns
df.dtypes
```

Out[4]:
```
rank          object
discipline    object
phd           int64
service       int64
sex           object
salary        int64
dtype: object
```

Data Frames attributes

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

Hands-on exercises

- ✓ Find how many records this data frame has;
- ✓ How many elements are there?
- ✓ What are the column names?
- ✓ What types of columns we have in this data frame?

Data Frames methods

Unlike attributes, python methods have *parenthesis.*
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

Hands-on exercises

- ✓ Give the summary for the numeric columns in the dataset
- ✓ Calculate standard deviation for all numeric columns;
- ✓ What are the mean values of the first 50 records in the dataset? ***Hint:*** use head() method to subset the first 50 records and then calculate the mean

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 an attribute *rank* for pandas data frames, so to select a column with a name "rank" we should use method 1.

Hands-on exercises

- ✓ Calculate the basic statistics for the *salary* column;
- ✓ Find how many values in the *salary* column (use *count* method);
- ✓ Calculate the average salary;

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 dplyr() function 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 create we can calculate various statistics for each group:

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

	salary
rank	
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
- 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 [ ]: #Calculate mean salary for each professor rank:
        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 Frames: Slicing

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: Slicing

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

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, 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 by their labels:
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 return.

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

Missing Values

Missing values are marked as NaN

In [ ]:
```
# Read a dataset with missing values
flights = pd.read_csv("http://rcs.bu.edu/examples/python/data_analysis/flights.csv")
```

In [ ]:
```
# Select the rows that have at least one missing value
flights[flights.isnull().any(axis=1)].head()
```

Out[ ]:

	year	month	day	dep_time	dep_delay	arr_time	arr_delay	carrier	tailnum	flight	origin	dest	air_time	distance	hour	minute
330	2013	1	1	1807.0	29.0	2251.0	NaN	UA	N31412	1228	EWR	SAN	NaN	2425	18.0	7.0
403	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EHAA	791	LGA	DFW	NaN	1389	NaN	NaN
404	2013	1	1	NaN	NaN	NaN	NaN	AA	N3EVAA	1925	LGA	MIA	NaN	1096	NaN	NaN
855	2013	1	2	2145.0	16.0	NaN	NaN	UA	N12221	1299	EWR	RSW	NaN	1068	21.0	45.0
858	2013	1	2	NaN	NaN	NaN	NaN	AA	NaN	133	JFK	LAX	NaN	2475	NaN	NaN

Missing Values

There are a number of methods to deal with missing values in the data frame:

df.method()	description
dropna()	Drop missing observations
dropna(how='all')	Drop observations where all cells is NA
dropna(axis=1, how='all')	Drop column if all the values are missing
dropna(thresh = 5)	Drop rows that contain less than 5 non-missing values
fillna(0)	Replace missing values with zeros
isnull()	returns True if the value is missing
notnull()	Returns True for non-missing values

Missing Values

- When summing the data, missing values will be treated as zero
- If all values are missing, the sum will be equal to NaN
- cumsum() and cumprod() methods ignore missing values but preserve them in the resulting arrays
- Missing values in GroupBy method are excluded (just like in R)
- Many descriptive statistics methods have *skipna* option to control if missing data should be excluded . This value is set to *True* by default (unlike R)

Aggregation Functions in Pandas

Aggregation - computing a summary statistic about each group, i.e.

- compute group sums or means
- compute group sizes/counts

Common aggregation functions:

min, max
count, sum, prod
mean, median, mode, mad
std, var

Aggregation Functions in Pandas

agg() method are useful when multiple statistics are computed per column:

In [ ]:
```
flights[['dep_delay','arr_delay']].agg(['min','mean','max'])
```

Out[ ]:

	dep_delay	arr_delay
min	-16.000000	-62.000000
mean	9.384302	2.298675
max	351.000000	389.000000

Basic Descriptive Statistics

df.method()	description
describe	Basic statistics (count, mean, std, min, quantiles, max)
min, max	Minimum and maximum values
mean, median, mode	Arithmetic average, median and mode
var, std	Variance and standard deviation
sem	Standard error of mean
skew	Sample skewness
kurt	kurtosis

Graphics to explore the data

Seaborn package is built on matplotlib but provides high level interface for drawing attractive statistical graphics, similar to ggplot2 library in R. It specifically targets statistical data visualization

To show graphs within Python notebook include inline directive:

```
In [ ]: %matplotlib inline
```

Graphics

	description
distplot	histogram
barplot	estimate of central tendency for a numeric variable
violinplot	similar to boxplot, also shows the probability density of the data
jointplot	Scatterplot
regplot	Regression plot
pairplot	Pairplot
boxplot	boxplot
swarmplot	categorical scatterplot
factorplot	General categorical plot

Basic statistical Analysis

statsmodel and scikit-learn - both have a number of function for statistical analysis

The first one is mostly used for regular analysis using R style formulas, while scikit-learn is more tailored for Machine Learning.

statsmodels:

- linear regressions
- ANOVA tests
- hypothesis testings
- many more ...

scikit-learn:

- kmeans
- support vector machines
- random forests
- many more ...

See examples in the Tutorial Notebook

Conclusion

Thank you for attending the tutorial.

Please fill the evaluation form:

http://scv.bu.edu/survey/tutorial_evaluation.html

Questions:

email: koleinik@bu.edu (Katia Oleinik)