Business Analytics in Python

DS Part IV – Python Hands on

Sasken Training, Adyar , 9840014739 Chennai – 600 020

Agenda

- Python Introduction
- Python environment set up using Anaconda distribution
- Getting started with Python ?
- Basic data types and Container types
- Variables and Control structures
- Functions
- Modules and Packages

What and Why of Python ?

- Created by Guido Van Rossum during 1985-1990 as a hobby project
- Multi purpose scripting language
- Interpreted & Interactive
- Object oriented

Python has extensive framework and libraries for performing various tasks in Data Science

Numpy scipy Pandas Seaborn Matplotlib plotly scikit-learn statsmodels

Which companies that use Python ?

- Search engine and web crawler

- Most of its implementations

- Most of its implementations

- Most of its implementations

- Mission control center to perform auxiliary processing

What can you do with Python ?

- Develop web applications
- Implementing machine learning algorithms such as Scikit-learn, NLTK and Tensor flow .
- Computer vision – Face detection , color detection while using Opencv and python
- Raspberry Pi – You can even build a robot and automate your home.
- Web Scraping – Grab data from a website
- Gui Development – Tkinter or PyQt to build a GUI (desktop) application
- Game development – Create a video game using Pygame module. PyGame applications can run on Android devices


Raspberry Pi 3 Model B

Python Installation

Install Anaconda which comes up with the below packages

- Pandas - provides rich data structures and functions designed to work with structured
  - data fast, easy , and expressive.
- Numpy – Numerical Python
- Matplotlib – this library is for producing plots and other 2D visualizations
- Scipy - Scientific programming
- IPython Notebook
- Scikit learn meant for data science
- Statsmodels – statistical analysis

To install other packages , for instance if would like to install packages like seaborn, you can use the below command from the command line

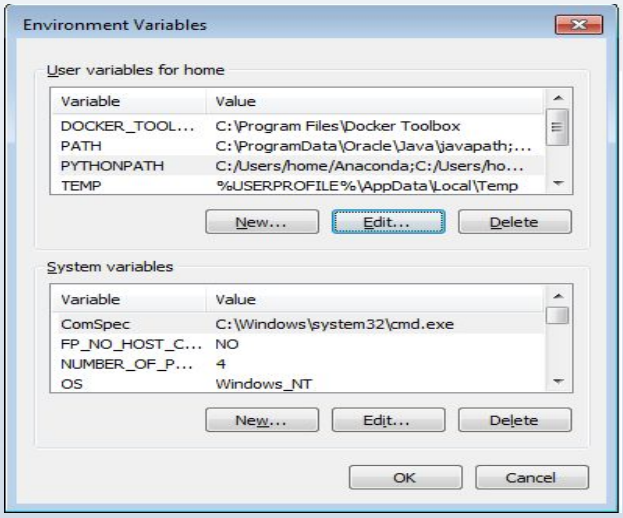
- **conda install seaborn**

```
In [9]: # Numpy is a library for working with Arrays
        import numpy as np
        print ("Numpy version:         %6.6s (need at least 1.7.1)" % np.__version__)

        # SciPy implements many different numerical algorithms
        import scipy as sp
        print ("SciPy version:         %6.6s (need at least 0.12.0)" % sp.__version__)

        # Pandas makes working with data tables easier
        import pandas as pd
        print ("Pandas version:        %6.6s (need at least 0.11.0)" % pd.__version__)

        Numpy version:         1.10.4 (need at least 1.7.1)
        SciPy version:         0.17.0 (need at least 0.12.0)
        Pandas version:        0.17.1 (need at least 0.11.0)
```

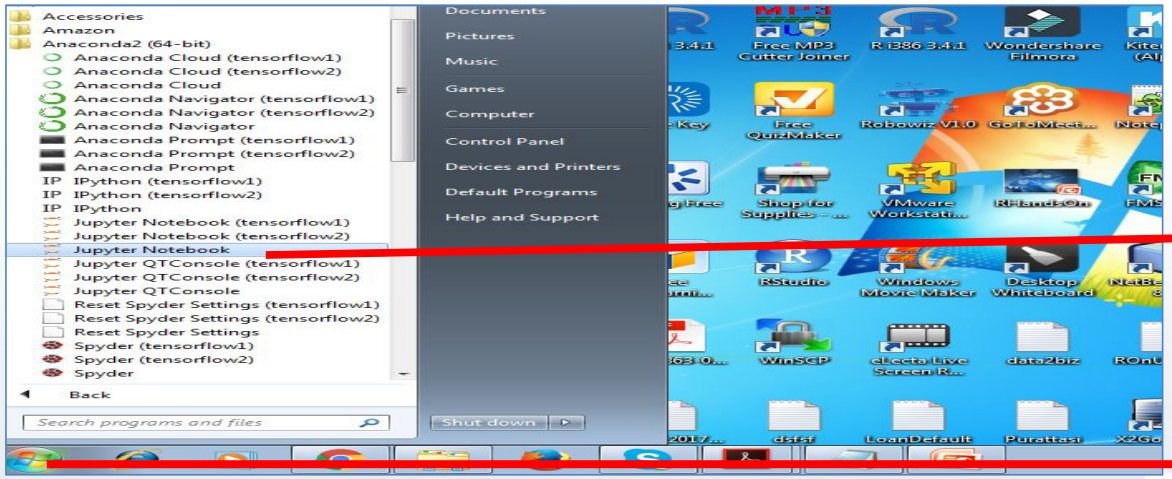
Python Environment Variables

PYTHONPATH ->
C:/Users/home/Anaconda;C:/Users/home/Anaconda/lib;C:/Users/home/Anaconda/DLLs;

PATH ->
C:\users\home\Anaconda;C:\Users\home\Anaconda\Scripts;C:\Users\home\Anaconda\lib

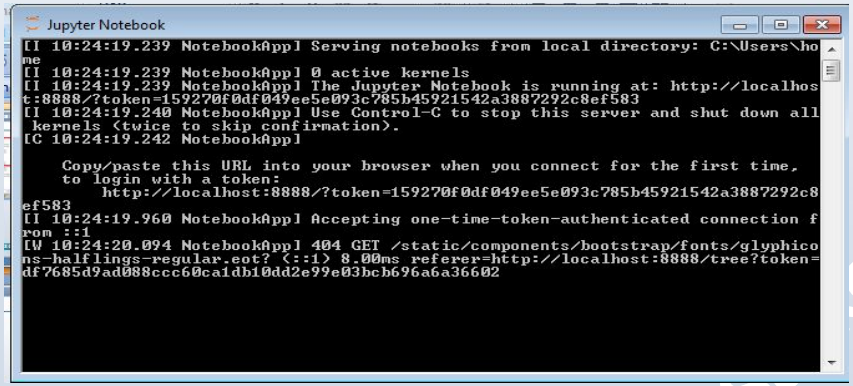


How to start Jupyter Notebook



Step 2
Click on Jupyter Notebook

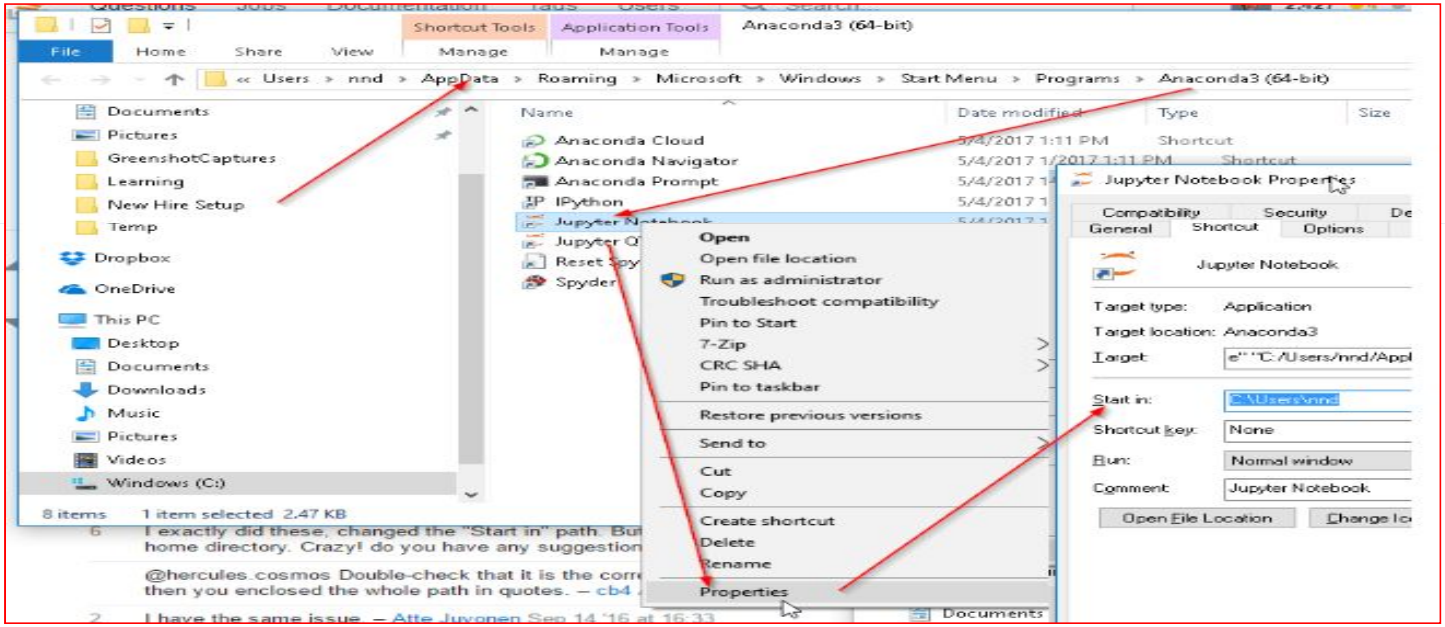
Step 1
Click on the start menu, then All Programs



It will take for a while to start the Jupyter Kernel. Do not close this , Just minimize this window

How to change the Jupyter default start folder name

Click on the *Start Menu*, then *All Programs*
Click on the *Anaconda2 or Anaconda3* folder;
In there you should see *Jupyter Notebook*.
Right-click the correct Jupyter Notebook entry, then click on *Properties*
Enter a path in the *Start in:* box; if the path has spaces in it, you must enclose it in double quotes



How to change the Jupyter default start folder name

se the jupyter notebook config file:

Open cmd (or *Anaconda Prompt*) and run jupyter notebook --generate-config.
This writes a file to C:\Users\username\.jupyter\jupyter_notebook_config.
Change line 179 #c.NotebookApp.notebook_dir = ''
to c.NotebookApp.notebook_dir = 'your path'
Make sure you use forward slashes in your path.

Installing sub packages

- conda install scikit-leann
- pip install ggplot

If Anaconda is already installed , simply run the following commands to update Ipython to the latest version

- Conda update conda
- Conda update ipython

Types of Python IDE

- Jupyter notebook
- Spyder
- PyCharm: This specifies the user interface based on Java Swing
- PyDev: This denotes the user interface based on SWT (works on Eclipse)
- Interactive Editor for Python (IEP)
- Canopy from Enthought: This is based on PyQt
- The Anaconda distribution of Spyder from Continuum Analytics: This is also based on PyQt

Indentation

- Most languages don't care about indentation
- Python uses whitespace and indents to denote blocks of code
  (indent : TAB or control + , dedent CTRL [ )
- Lines of code that begin a block end in a colon:
- Lines within the code block are indented at the same level
- To end a code block, remove the indentation
- You'll want blocks of code that run only when certain conditions are met

```
/*                          */
If ( sku == 500)
    if (stock_on_hand < 50)
                raise_oder(sky,qty) ;
 else
          quit();
```

```
/*              R program        */
If ( skuqty == 500) {
    soh = (soh – ord qty)
    print soh
     raise_oder(sky,qty) ;
       }
else {
 soh = (soh + rec_qty)
 print soh
   }
}
```

```
/*                          */
If ( sku == 500) :
     soh = (soh – ord qty)
     print soh
     raise_oder(sky,qty) ;
 else:
    soh = (soh + rec_qty)
    print soh
```

Indentation

```
In [27]: x=25
         if x> 25:
             print("10")
         else:
             if x < 25:
                 print("Less than 25")
             else:
                   print(" equal to 25")

         equal to 25
```

```
In [28]: x=25
         if x> 25:
             print("10")
         else:
             if x < 25:
                     print("Less than 25")
                 else:
                       print(" equal to 25")

           File "<ipython-input-28-83eb5a43575b>", line 7
             else:
                 ^
         IndentationError: unindent does not match any outer indentation level
```

Indentation

```
pwd = 'jackson'
if pwd == 'jackson':
    print('Logging on ...')
else:
    print('Incorrect password.')

print('All done!')
```

```
Logging on ...
All done!
```

```
age = int(input('How old are you? '))
if age <= 2:
    print(' free')
elif 2 < age < 13:
    print(' child fare')
else:
    print('adult fare')
```

```
How old are you? 1
 free
```

- elif is short for else if, and you can use as many elif-blocks as needed. As usual, each of the code blocks in an if/elif-statement must be consistently indented the same amount. Not only does Python require this indentation so that it can recognize the code blocks,

Comments

 # Traditional one line comment

“ “ “

Any string not assigned to a variable is considered a comment

This is an example of a multi-line comment.

“ “ “

```
"""
Any string not assigned to a variable is considered a comment
This is an example of a multi-line comment.
"""
x = 55
print x
```

```
55
```

Line Continuation

```
In [ ]: (1.5+13.2) + 50*
        2

  File "<ipython-input-60-4ddf00a158f8>", line 1
    (1.5+13.2) + 50*
                   ^
SyntaxError: invalid syntax


In [ ]: (1.5+13.2) + 50* \
        2

Out[ ]: 114.7
```

Arithmetic Operators

Operator Name	Operator	Description	Example
Addition	+	Adds values on either side of the operator	a + b will give 60
Subtraction	-	Subtracts right hand operand from left hand operand	a - b will give 20
Multiplication	*	Multiplies values on either side of the operator	a * b will give 800
Division	/	Divides left hand operand by right hand operand	a / b will give 2
Modulus	%	Divides left hand operand by right hand operand and returns remainder	b % a will give 0
Exponent	**	Performs exponential (power) calculation on operators	a**b will give 40 to the power 20
Floor Division	//	The division of operands where the result is the quotient in which the digits after the decimal point are removed.	9//2 is equal to 4 and 9.0//2.0 is equal to 4.0

Comparison and logical operators

Operator Name	Operator	Description	Example (a=40 and b=20)
Equals	==	Checks if the value of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
Not Equals	!=	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.
Not Equals	<>	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a <> b) is true. This is similar to != operator.
Greater Than	>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is true.
Less Than	<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is not true.
Greater Than Equals To	>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is true.
Less Than Equals To	<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is not true.

Operator	Description	Example a = 40 b = 20
and	Called Logical AND operator. If both the operands are true then then condition becomes true.	(a and b) is true.
or	Called Logical OR Operator. If any of the two operands are non zero then then condition becomes true.	(a or b) is true.
not	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	not(a and b) is false.

Membership operators

Operator	Description	Example
in	Evaluates to true if it finds a variable in the specified sequence and false otherwise.	x in y, here in results in a 1 if x is a member of sequence y.
not in	Evaluates to true if it does not finds a variable in the specified sequence and false otherwise.	x not in y, here not in results in a 1 if x is not a member of sequence y.

Variables

```
In [4]: 4+5
        print("Welcome to Python... So excited to work with ")

        Welcome to Python... So excited to work with
```

From command line

```
#!/usr/bin/python

print "Hello, Python!"
```

```
# Add two numbers together
x = 3
y = 8
res = x + y
print("Output #2: Four plus five equals {0:d}.".format(res))
```

Python is case sensitive

Standard data types

Basic types
Numbers
Strings

Complex Numbers

Container types
Lists
Dictionaries
Tuples

Integer 22

Float 22.876

Python strings are immutable

A complex number is made up of two floating-point values, one each for the real and imaginary parts. You can access the parts of a complex object z as read-only attributes z.real and z.imag. You can specify an imaginary literal as a floating point or decimal literal followed by a j or J: 0j, 0.j, 0.0j, .0j, 1j, 1.j, 1.0j, 1e0j, 1.e0j, 1.0e0j. For example, to denote the complex number that equals one, use expressions like1+0j or 1.0+0.0j. Python performs the addition or subtraction at compile time.

The constructors int(), long(), float(), and complex() can be used to produce numbers of a specific type

Numeric and String variables

```
x = 1
y = 2
print( x * y)
```

```
2
```

```
x = '1'
y = '2'
print( x + y)
```

```
12
```

```
x = '1'
y = '2'
print( x * y)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-f43b88a5d763> in <module>()
      1 x = '1'
      2 y = '2'
----> 3 print( x * y)

TypeError: can't multiply sequence by non-int of type 'str'
```

Multiplication of a string type by another String type using this * operator is not allowed

Type()

```
x = 1
y = 2.3
print( x * y)
print type(x)
print type(y)
```

```
2.3
<type 'int'>
<type 'float'>
```

```
z = 10.5
w = long(z)
print long(z)
print type(w)
```

```
10
<type 'long'>
```

```
bools = True
type(bools)
```

```
bool
```

Type conversions

```
>>> float(22//5)
4.0
>>> int(4.5)
4
>>> int(3.9)
3
>>> round(3.9)
4
>>> round(3)
3
```

Strings

```
# Escape sequences to accommodate quotes and backslash
print '\" Double quote escape sequence \"'
print '\' Single quote escape sequence \''
print ' Backslash \\  escape sequence \''
```

```
" Double quote escape sequence "
' Single quote escape sequence '
 Backslash \  escape sequence '
```

Working with String data types

```
x1 = 'Hello World!'
x2 = "Python Programming"
print "x1[0]: ", x1[0]
print "x2[1:5]: ", x2[1:5]
```

```
x1[0]:  H
x2[1:5]:  ytho
```

```
var1 = 'Hello World!'

print "Updated String :- ", x1[:6] + 'Python World '
```

Updated string

```
Updated String :-  Hello Python World
```

```
x1 = 'Hello World!'
x2 = "Python Programming"
type(x1)
type(x2)
```

```
str
```

```
# formatted string
print "My name is %s and weight is %d kg!" % ('Sasken', 31)
```

Formatted string

```
My name is Sasken and weight is 31 kg!
```

Strings

```
# Escape sequences to accommodate quotes and backslash
print '\" Double quote escape sequence \"'
print '\' Single quote escape sequence \''
print ' Backslash \\  escape sequence \''
```

```
" Double quote escape sequence "
' Single quote escape sequence '
 Backslash \  escape sequence '
```

Dates and Times

- The built-in Python datetime module provides datetime, date, and time types.
- The datetime type as you may imagine combines the information stored in date and time and is the most commonly used:

```
from datetime import datetime, date, time
dt = datetime(2011, 10, 29, 20, 30, 21)
```

```
print dt
```

```
2011-10-29 20:30:21
```

```
dt.time()
```

```
datetime.time(20, 30, 21)
```

```
# The strftime method formats a datetime as a string:
dt.strftime('%m/%d/%Y %H:%M')
```

```
'10/29/2011 20:30'
```

```
# Strings can be converted (parsed) into datetime objects using the strptime function:
datetime.strptime('20091031', '%Y%m%d')
```

```
datetime.datetime(2009, 10, 31, 0, 0)
```

```
# Given a datetime instance, you can extract the equivalent date and time objects by calling methods on the datetime of the sa
dt.time()
```

```
datetime.time(20, 30, 21)
```

Lists

Lists are the most basic data structure in Python

Lists are mutable

What is mutable ?

Once the elements are stored in lists that can be modified later

```
a = [7.1,12,2.76,13]
print(type(a))
a[0]

print(' the first value of list a is ',a[0])
print(' the last value of list a is ',a[-1])
a[1] = 24
print('after modifying the 2nd element of 12 by 24', a)
```

```
<type 'list'>
(' the first value of list a is ', 7.1)
(' the last value of list a is ', 13)
('after modifying the 2nd element of 12 by 24', [7.1, 24, 2.76, 13])
```

As can be seen here,
The 2nd element got modified
From 12 to 24

Lists

```
height = [ 143.52, 129.82, 201.52, 217.32]
tmpHeight = []
tmpHeight.append( int (height[0]))
tmpHeight.append( int(height[1]))
tmpHeight.append( int(height[2]))
tmpHeight.append( int(height[3]))
print(tmpHeight)
```

```
[143, 129, 201, 217]
```

Accessing values stored in Lists – Slice operator [] and [:]

x = ‘computer’

Code	Result	Explanation
x[1:4]	‘omp’	Item from 1 to 3
x[3:]	‘puter’	Items 3 to end
x[:5]	‘compu’	Items 0 to 4
x[-1]	‘r’	Last item
x[-3:]	‘ter’	Last 3 items
x[:-2]	‘comput’	All except last 2 items
X[1:5:2]	??	?

```
# items start through the end (but the end is not included!)
a[start:end]
 # items start through the rest of the array
a[start:]
# items from the beginning through the end (but the end is not included!) a[:end]
```

```
# start through not past end, by step a[start:end:step]
```

Accessing values stored in Lists – Slice operator [] and [:]

```
list = [ 'Ganesh', 456 , 52.32, 'Saravanan', 07.02 ]
tinylist = [123, 'Mahesh']

print list          # Prints complete list
print list[0]       # Prints first element of the list
print list[1:3]     # Prints elements starting from 2nd till 3rd
print list[2:]      # Prints elements starting from 3rd element
print tinylist * 2  # Prints list two times
print list + tinylist # Prints concatenated lists
```

```
['Ganesh', 456, 52.32, 'Saravanan', 7.02]
Ganesh
[456, 52.32]
[52.32, 'Saravanan', 7.02]
[123, 'Mahesh', 123, 'Mahesh']
['Ganesh', 456, 52.32, 'Saravanan', 7.02, 123, 'Mahesh']
```

```
prices = (3.99, 6.00, 10.00, 5.25)
print prices
```

```
(3.99, 6.0, 10.0, 5.25)
```

```
# to display all the items from a list
print prices[:]
# to display items start through the rest of the array
print  prices[1:]
# To display item from the beginning through end-1
print prices[:3]
# item start throught the rest of the array
print prices[2:]
```

```
(3.99, 6.0, 10.0, 5.25)
(6.0, 10.0, 5.25)
(3.99, 6.0, 10.0)
(10.0, 5.25)
```

Iterators

```
ListA = ['Orange', 'Yellow', 'Green', 'Brown']
ListB = [1, 2, 3, 4]
for Value in ListB[1:3]:
            print (Value)
```

```
2
3
```

```
height = [ 143.52, 129.82, 201.52, 217.32]
tmpHeight = []
tmpHeight.append( int (height[0]))
tmpHeight.append( int(height[1]))
tmpHeight.append( int(height[2]))
tmpHeight.append( int(height[3]))
print(tmpHeight)
```

```
[143, 129, 201, 217]
```

```
tmpList = []
for i in height:
      tmpList.append(int(i))
print tmpList
```

```
[143, 129, 201, 217]
```

List Comprehension

- A **list comprehension** is a syntactic construct that enables lists to be created from other lists using a compact, mathematical syntax:
- It's a very easy way to create lists
- Consists of square brackets that contain an expression and then **for** a clause
- The expression can be anything – all kinds of objects can be put in lists

```
prices = (3.99, 6.00, 10.00, 5.25)
print prices
tmplist2 = [int(i) for i in prices]
print tmplist2
```

```
(3.99, 6.0, 10.0, 5.25)
[3, 6, 10, 5]
```

```
names = ['Robert','Richard','Michelle']
last3Letters = [names[-2:] for names in names]
print last3Letters
```

```
['rt', 'rd', 'le']
```

List Comprehension

- A **list comprehension** is a syntactic construct that enables lists to be created from other lists using a compact, mathematical syntax:
- It's a very easy way to create lists
- Consists of square brackets that contain an expression and then **for** a clause
- The expression can be anything – all kinds of objects can be put in lists

```
prices = (3.99, 6.00, 10.00, 5.25)
print prices
tmplist2 = [int(i) for i in prices]
print tmplist2
```

```
(3.99, 6.0, 10.0, 5.25)
[3, 6, 10, 5]
```

```
names = ['Robert','Richard','Michelle']
last3Letters = [names[-2:] for names in names]
print last3Letters
```

```
['rt', 'rd', 'le']
```

Transforming Lists to various other data structures

Convert a list to a string & string to a list

You convert a list to a string by using ''.join(). This operation allows you to glue together all strings in your list together and return them as a string

```
# List of Strings to a String
listOfStrings = ['Welcome', 'To', 'PythonWorld !']
strOfStrings = ''.join(listOfStrings)
print(strOfStrings)
# Strings to a list
list(listOfStrings)
```

Convert a List of Integers to a String

```
listOfNumbers = [2,4, 6]
strOfNumbers = ''.join(str(n) for n in listOfNumbers)
print(strOfNumbers)
```

How to clone or copy a list ?

```
list1 = [10,20,30,50]
list2 = list1    # here the address of list1 has binded to list2 , not the data

list2 = list1[:]
```

How to concatenate two lists ?

```
list3 = list1 + list2
```

How to define and access values from tuples ?

```
a = (1,5,6,7)
print a

(1, 5, 6, 7)

print ('2nd element of the tuple a is ' , a[1])
a[1] =10

('2nd element of the tuple a is ', 5)

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-52-95c965ca135d> in <module>()
      1 print ('2nd element of the tuple a is ' , a[1])
----> 2 a[1] =10

TypeError: 'tuple' object does not support item assignment
```

Each values in an tuples can be accessed by numbers only. Using an indexing operator [] you can select a specific sequence

In this case, I can't replace the value of 5 by 10 since tuple is an immutable one

Tuples

To access the first element from tuples , the index value to be specified as 0

```
T = ("a", "b", "mpilgrim", "z", "example")
T[0]
```

```
'a'
```

Negative indices count from the end of the tuple, just as with a list.

```
T = ("a", "b", "mpilgrim", "z", "example")
T[-1]
```

```
'example'
```

The elements of a tuple have a defined order, just like a list. Tuples indices are zero-based, just like a list, so the first element of a non-empty tuple is always t[0]

Tuples

```
stuple = sorted(mytuple)
print stuple
```

```
[11, 22, 33, 44, 54]
```

Point to Note:
Sorted() returns a list

```
In [10]: mytuple = (11,22,33)
         mytuple += (44,)
         print mytuple

         (11, 22, 33, 44)

In [11]: mytuple += (54)
         print mytuple
         ---------------------------------------------------------------------------
         TypeError                                 Traceback (most recent call last)
         <ipython-input-11-5e39229d5555> in <module>()
         ----> 1 mytuple += (54)
               2 print mytuple

         TypeError: can only concatenate tuple (not "int") to tuple
```

Error cause :
Comma is required , without the comma it is treated as an integer

Lists Vs Tuples

Lists are enclosed in square brackets [] whereas tuples are enclosed in parantheses ()

Lists are mutable whereas Tuples are immutable

Dictionary

- Dictionaries are compound type different from the sequence types as compared to strings , lists and tuples.
- **Each element in a dictionary consists of key and value pair**
- **Dictionaries are enclosed in curly braces {}**
- Key and values are separated by a colon
- Pairs of entries are separated by comma
- Keys must be UNIQUE

Dictionaries are mutable

Dictionary operations

```
what_am_i = {'apples': 32, 'bananas': 47, 'pears': 17}
print('type of what_am_i',type(what_am_i))

what_am_i = {'apples', 'bananas', 'pears'}
print('type of what_am_i',type(what_am_i))
```

```
('type of what_am_i', <type 'dict'>)
('type of what_am_i', <type 'set'>)
```

```
inventory = {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}
print(inventory)
```

```
{'pears': 217, 'apples': 430, 'oranges': 525, 'bananas': 312}
```

```
del inventory['pears']
print(inventory)
```

```
{'apples': 430, 'oranges': 525, 'bananas': 312}
```

```
inventory['pears'] = 0
print(inventory)
```

```
{'pears': 0, 'apples': 430, 'oranges': 525, 'bananas': 312}
```

```
len(inventory)
```

```
4
```

```
print 'pears' in inventory
'blueberries' in inventory
```

```
True

False
```

Each values in a dictionary can be accessed by key's. Using an indexing operator [] , you can specify a selective key to access the value. To remove a specific key from a dictionary , use del command

To append a new key value pair , assign the value to the key in the dictionary

Dictionary Operations

```
inventory = {'apples': 430, 'bananas': 312, 'oranges': 525, 'pears': 217}
print(inventory)
```

```
{'pears': 217, 'apples': 430, 'oranges': 525, 'bananas': 312}
```

```
del inventory['pears']
print(inventory)
```

```
{'apples': 430, 'oranges': 525, 'bananas': 312}
```

```
inventory['pears'] = 0
print(inventory)
```

```
{'pears': 0, 'apples': 430, 'oranges': 525, 'bananas': 312}
```

```
len(inventory)
```

```
4
```

```
print 'pears' in inventory
'blueberries' in inventory
inventory.keys()
```

```
True
['pears', 'apples', 'oranges', 'bananas']
```

To display only the keys From a dictionary , specify keys()

d1={1:2,3:4}; d2={5:6,7:9}; d3={10:8,13:22}
d4 = {}
for d in (d1, d2, d3):
    d4.update(d)

Dictionary Operations

```
age = {'Baba':26,'Julia':24}
age.update({'Baba':28})
print age
```

```
{'Baba': 28, 'Julia': 24}
```

```
age.get('Baba')
```

```
28
```

```
age.pop('Julia')
print age
```

Removing an item

```
{'Baba': 28}
```

```
age.popitem()
```

```
('Baba', 28)
```

To Remove an item randomly

Sets

- A set is a python data type that holds an unordered collection of unique elements
- Identified by curly braces
- {'Raja','Ashok','Alok'}
- Can only contain unique elements
  - Duplicates are eliminated

```
myset = {22.2,33,12,2,2}
print myset
```

```
set([33, 2, 12, 22.2])
```

Sets do not support indexing

Basic operations on set

Description	Commands
Add item to set x	x.add(item)
Remove an item	x.remove(item)
get length of x	len(x)
Check membership in x	item in x item not in x
Pop random item from set x	x.pop()
Delete all items from x	x.clear()

Control Structures

if-else

```
# if-else statement
x = 5
if x > 4 or x != 9:
    print("Output #124: {}".format(x))
else:
    print("Output #124: x is not greater than 4")
```

if-elif-else

```
# if-elif-else statement
if x > 6:
    print("Output #125: x is greater than six")
elif x > 4 and x == 5:
    print("Output #125: {}".format(x*x))
else:
    print("Output #125: x is not greater than 4")
```

For loops

```
In [3]: x = [8,12,14,3]
        for items in x:
            print ("Item is ", items)

        Item is  8
        Item is  12
        Item is  14
        Item is  3
```

for loops

```
y = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', \
'Nov', 'Dec']
z = ['Annie', 'Betty', 'Claire', 'Daphne', 'Ellie', 'Franchesca', 'Greta', \
'Holly', 'Isabel', 'Jenny']

print("Output #126:")
for month in y:
     print("{!s}".format(month))

print("Output #127: (index value: name in list)")
for i in range(len(z)):
     print("{0!s}: {1:s}".format(i, z[i]))

print("Output #128: (access elements in y with z's index values)")
for j in range(len(z)):
     if y[j].startswith('J'):
         print("{!s}".format(y[j]))

print("Output #129:")
for key, value in another_dict.items():
     print("{0:s}, {1}".format(key, value))
```

While Loops

while loops

```
print("Output #134:")
x = 0
while x < 11:
    print("{!s}".format(x))
    x += 1
```

Functions

- Functions with multiple parameters
- Functions with multiple return values
- Functions with default arguments and flexible arguments
- lambda function

Functions

In [17]:
```
def changeme( mylist ):
   "This changes a passed list into this function"
   mylist = [8,4,2,3]; # This would assig new reference in mylist
   print ("Values inside the function: ", mylist)
   return

# Now you can call changeme function
mylist = [30,10,20];
changeme( mylist );
print ("Values outside the function: ", mylist)
```

Function header

Docstring

```
Values inside the function:  [8, 4, 2, 3]
Values outside the function:  [30, 10, 20]
```

In [19]:
```
# Function definition is here
def printinfo( name, age ):
   "This prints a passed info into this function"
   print ("Name: ", name)
   print ("Age ", age)
   return;

# Now you can call printinfo function
printinfo( age=50, name="miki" )
```

```
Name:  miki
Age  50
```

Assigning a function to a variable

Let’s define a simple function and see how it can be used as a variable:
Let us define a simple function.

In [16]:
```
def square_input(x):
  return x*x
# We will follow it by assigning that function to a variable
square_me = square_input
# And finally invoke the variable
print (square_me(5))
```

```
25
```

```
def df(x=20):
    x = x+10
    return x
```

```
def df2(df1):
    result = df1()
    return result
df2(df)
```

```
30
```

Passing a function as a parameter , Returning multiple values

```
def method1():
    return 'Welcome to Python hello world'

def method2(methodToRun):
    result = methodToRun()
    return result

method2(method1)
```

```
'Welcome to Python hello world'
```

Reading & Writing data into a file

In [14]:
```
#!/usr/bin/env python3
import pandas as pd
import sys
input_file = sys.argv[1]
output_file = sys.argv[2]
data_frame = pd.read_csv(input_file)
data_frame['Cost'] = data_frame['Cost'].str.strip('$').astype(float)
data_frame_value_meets_condition = data_frame.loc[(data_frame['Supplier Name']\
.str.contains('Z')) | (data_frame['Cost'] > 600.0), :]
data_frame_value_meets_condition.to_csv(output_file, index=False)
```

Iterators

```
ListA = ['Orange', 'Yellow', 'Green', 'Brown']
ListB = [1, 2, 3, 4]
for Value in ListB[1:3]:
            print (Value)
```

```
2
3
```

Tuples

```
# Use parentheses to create a tuple
my_tuple = ('x', 'y', 'z')
print("Output #93: {}".format(my_tuple))
print("Output #94: my_tuple has {} elements".format(len(my_tuple)))
print("Output #95: {}".format(my_tuple[1]))
longer_tuple = my_tuple + my_tuple
print("Output #96: {}".format(longer_tuple))
```

```
# Convert tuples to lists and lists to tuples
my_list = [1, 2, 3]
my_tuple = ('x', 'y', 'z')
print("Output #100: {}".format(tuple(my_list)))
print("Output #101: {}".format(list(my_tuple)))
```

User Defined Package creation

Step 1 – Create a file by the name of compSal.py and copy paste the below content

```
In [8]: print("Module Creation Example")
        def calcSal(tsal):
            hra=10000
            gross = hra + tsal
            print("Calcualted result ", gross)

Module Creation Example

In [9]: calcSal(1000)

Calcualted result  11000
```

```
In [20]: print("Om Mahaa Ganapataye Namah")
         import os
         print (os.getcwd()) #
         os.chdir('c:/Users/home') #
         import sys
         sys.path.append('c:/Users/home')

Om Mahaa Ganapataye Namah
c:\Users\home

In [22]: from compSal import calcSal

In [23]: calcSal(20000)

Calcualted result  30000
```

* represents – In-progress state

```
In [*]: import pandas as pd
unames = ['user_id', 'gender', 'age', 'occupation', 'zip']
users = pd.read_table('D:/dat2/users.dat', sep='::', header=None,names=unames,engine='python')
rnames = ['user_id', 'movie_id', 'rating', 'timestamp']
ratings = pd.read_table('D:/dat2/ratings.dat', sep='::', header=None,names=rnames,engine='python')
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('D:/dat2/movies.dat', sep='::', header=None,names=mnames,engine='python')
data = pd.merge(pd.merge(ratings, users), movies)
data.ix[0]
```

String Replacement

```
In [9]: string5 = "Let's replace the spaces in this sentence with other characters."
        string5_replace = string5.replace(" ", "!@!")
        print("Output #32 (with !@!): {0:s}".format(string5_replace))
        string5_replace = string5.replace(" ", ",")
        print("Output #33 (with commas): {0:s}".format(string5_replace))

        Output #32 (with !@!): Let's!@!replace!@!the!@!spaces!@!in!@!this!@!sentence!@!with!@!other!@!characte
        rs.
        Output #33 (with commas): Let's,replace,the,spaces,in,this,sentence,with,other,characters.

In [ ]:
```

Date

```
In [10]: from math import exp, log, sqrt
         import re
         from datetime import date, time, datetime, timedelta
         # Print today's date, as well as the year, month, and day elements
         today = date.today()
         print("Output #41: today: {0!s}".format(today))
         print("Output #42: {0!s}".format(today.year))
         print("Output #43: {0!s}".format(today.month))
         print("Output #44: {0!s}".format(today.day))
         current_datetime = datetime.today()
         print("Output #45: {0!s}".format(current_datetime))

         Output #41: today: 2016-09-05
         Output #42: 2016
         Output #43: 9
         Output #44: 5
         Output #45: 2016-09-05 22:41:41.668000
```

```
# Calculate a new date using a timedelta
one_day = timedelta(days=-1)
yesterday = today + one_day
print("Output #46: yesterday: {0!s}".format(yesterday))
eight_hours = timedelta(hours=-8)
print("Output #47: {0!s} {1!s}".format(eight_hours.days, eight_hours.seconds))
```

timedelta function is used to subtract one day from today's date.

Working with Date data types

```
# Calculate the number of days between two dates
date_diff = today - yesterday
print("Output #48: {0!s}".format(date_diff))
print("Output #49: {0!s}".format(str(date_diff).split()[0]))
```

```
# Create a string with a specific format from a date object
print("Output #50: {:s}".format(today.strftime('%m/%d/%Y')))
print("Output #51: {:s}".format(today.strftime('%b %d, %Y')))
print("Output #52: {:s}".format(today.strftime('%Y-%m-%d')))
print("Output #53: {:s}".format(today.strftime('%B %d, %Y')))
```

```
# Create a datetime object with a specific format
# from a string representing a date
date1 = today.strftime('%m/%d/%Y')
date2 = today.strftime('%b %d, %Y')
date3 = today.strftime('%Y-%m-%d')
date4 = today.strftime('%B %d, %Y')

# Two datetime objects and two date objects
# based on the four strings that have different date formats
print("Output #54: {!s}".format(datetime.strptime(date1, '%m/%d/%Y')))
print("Output #55: {!s}".format(datetime.strptime(date2, '%b %d, %Y')))

# Show the date portion only
print("Output #56: {!s}".format(datetime.date(datetime.strptime\
(date3, '%Y-%m-%d'))))
print("Output #57: {!s}".format(datetime.date(datetime.strptime\
(date4, '%B %d, %Y'))))
```

Debugging in python

```
python –m pdb computeSal.py
<Pdb> s                --→ step through (execute one statement at a  time)
# type the variable to view the values that are assigned
<Pdb>  print sal,hra
<Pdb> c                --→ continue
<Pdb> list 3
<Pdb> break            --→ To see all break points
<Pdb> disable  1           --→ disable breakpoint 1
```

Debugging in Python

```
D:\py>python -m pdb computeSal.py
> d:\py\computesal.py(1)<module>()
-> print "Om Mahaa Ganapataye Namah"
(Pdb) list
  1  -> print "Om Mahaa Ganapataye Namah"
  2     def salary(sal,hra,leave):
  3         """
  4         compute the salary
  5         """
  6         netsal = (sal + hra -leave)
  7         if netsal > 100000:
  8             netsal = netsal - (netsal * .30)
  9             print netsal
 10             return netsal
 11         else:
(Pdb) break 6
Breakpoint 1 at d:\py\computesal.py:6
(Pdb) break 8
Breakpoint 2 at d:\py\computesal.py:8
(Pdb) _
```

```
 15     salary(10000,1000,5000)
[EOF]
(Pdb) s
Om Mahaa Ganapataye Namah
> d:\py\computesal.py(2)<module>()
-> def salary(sal,hra,leave):
(Pdb) list
  1     print "Om Mahaa Ganapataye Namah"
  2  -> def salary(sal,hra,leave):
  3         """
  4         compute the salary
  5         """
  6 B       netsal = (sal + hra -leave)
  7         if netsal > 100000:
  8 B           netsal = netsal - (netsal * .30)
  9             print netsal
 10             return netsal
 11         else:
(Pdb) _
```

Debugging in Python

```
  1  -> print "Om Mahaa Ganapataye Namah"
  2     def salary(sal,hra,leave):
  3         """
  4         compute the salary
  5         """
  6 B       netsal = (sal + hra -leave)
  7         if netsal > 100000:
  8 B           netsal = netsal - (netsal * .30)
  9             print netsal
 10             return netsal
 11         else:
(Pdb) break 8
Breakpoint 8 at d:\py\computesal.py:8
(Pdb) c
Om Mahaa Ganapataye Namah
> d:\py\computesal.py(6)salary()
-> netsal = (sal + hra -leave)
(Pdb) n
> d:\py\computesal.py(7)salary()
-> if netsal > 100000:
(Pdb) netsal
5000
(Pdb) whatis netsal
<type 'int'>
(Pdb)
```

Numpy Introduction

- Numpy stands for Numerical Python
- It performs calculations over entire array
- It provides ndarray, a fast and space-efficient multidimensional array providing vectorized arithmetic operations and sophisticated *broadcasting capabilities*
- Standard mathematical functions for fast operations on entire arrays of data without having to write loops

In order to use numpy first we must import the numpy as follows:

```
import numpy as np
```

Arrays

Working with **Lists** and **Arrays** , compare the Difference between Lists and Arrays

```
array11 = np.array(mydataset1)
print type(array11)
print array11
```

```
<type 'numpy.ndarray'>
[ 2  7  8 10]
```

```
mydataset2 = ([11,10,15,23])
print mydataset2
```

```
[11, 10, 15, 23]
```

```
print type(mydataset1)
print type(mydataset2)
print (mydataset1 + mydataset2)
# print (mydataset1 * mydataset2)
# the above one throws an error
print (mydataset1[0] * mydataset2[0])
```

```
<type 'list'>
<type 'list'>
[2, 7, 8, 10, 11, 10, 15, 23]
```

```
print ('value of row 1 ',multiarray[0])
print ('value of row 2 ',multiarray[1])
print ('value of row 3 ',multiarray[2])
```

```
('value of row 1 ', array([2, 5, 3]))
('value of row 2 ', array([5, 7, 3]))
('value of row 3 ', array([1, 5, 3]))
```

Arrays

```
In [5]: import numpy as np
        np_arr = np.array([[2.37,1.86,1.98,1.97],[52.73,11.68,21.89,31.79]])
        np_arr

Out[5]: array([[  2.37,   1.86,   1.98,   1.97],
               [ 52.73,  11.68,  21.89,  31.79]])

In [6]: np_arr.shape

Out[6]: (2, 4)
```

Slicing and Indexing Arrays

```
multiarray = np.array(([2,5,3],[5,7,3],[1,5,3]))
print multiarray
```

```
[[2 5 3]
 [5 7 3]
 [1 5 3]]
```

```
print multiarray[0]
print multiarray[2]
```

```
[2 5 3]
[1 5 3]
```

```
print ('everything before  1 ',multiarray[:1])
print ('Display all starting from  1 ',multiarray[1:])
```

```
('everything before  1 ', array([[2, 5, 3]]))
('Display all starting from  1 ', array([[5, 7, 3],
       [1, 5, 3]]))
```

Problem 1 : arr = array([0, 1, 2, 3, 4, 64, 64, 64, 8, 9])
What is the output of arr[1:9:2] ?

Problem 2 : arr2 = array([[2, 5, 3], [5, 7, 3], [1, 5, 3]])
What is the output of arr2[:2,1:] ?

Operations between Arrays and scalars

```
arr
```

```
array([[ 1.,  2.,  3.],
       [ 4.,  5.,  6.]])
```

```
arr * arr
```

```
array([[  1.,   4.,   9.],
       [ 16.,  25.,  36.]])
```

```
1/ arr
```

```
array([[ 1.        ,  0.5       ,  0.33333333],
       [ 0.25      ,  0.2       ,  0.16666667]])
```

```
arr + 1
```

```
array([[ 2.,  3.,  4.],
       [ 5.,  6.,  7.]])
```

```
arr ** 0.5
```

```
array([[ 1.        ,  1.41421356,  1.73205081],
       [ 2.        ,  2.23606798,  2.44948974]])
```

1/1 ¼ 1/9
1/16 1/25 1/36

What is Pandas ? Why Pandas ?

- Pandas was created by Wed McKinney in 2008
- conda install pandas or pip install pandas
- It is built on top of NumPy and provides features not available in it

- It provides fast, easy-to-understand data structures and helps fill the gap between Python and a language such as R.

- Easy data alignment , Date/time handling is pretty easier

- Fast data access (SQL databases, flat files etc.,)

- Group by, joining/merging etc.,
- Time series modeling

Pandas Data structures

- Series - A Series is a one-dimensional array-like object containing an array of data (of any NumPy data type) and an associated array of data labels, called its *index*
- Data Frame

Series

It is a one-dimensional array-like object which consists an array of data (of any Numpy data type) and an associated array of data labels, called its index

```
import pandas as pd

pandas_Series = pd.Series([5,7,8.2,0])
pandas_Series
```

```
0    5.0
1    7.0
2    8.2
3    0.0
dtype: float64
```

values

Index

Series

```
import numpy as np
import pandas as pd
from  pandas import Series, DataFrame
```

```
###how to define series
s = Series [data,index=index or data labeles]
```

```
commission = Series ([7500,8200,1200], index=['Ashok','Mythri','Selva'])
print commission
```

```
Ashok     7500
Mythri    8200
Selva     1200
dtype: int64
```

```
commission['Mythri']
```

```
8200
```

```
sal = commission
print sal[sal < 8200]
print sal[sal > 1200]
```

```
Ashok    7500
Selva    1200
dtype: int64
Ashok     7500
Mythri    8200
dtype: int64
```

DataFrame

- DataFrame is a 2-dimensional labeled array
- Column can be heterogeneous types : float, int, bool, and so on
- Columns are made up of pandas Series objects
- Analogous to a table or spreadsheet of data
- Columns can be inserted or deleted
- Rich indexing operations: SQL-like join/merges, etc.,

Index

- Indexes are used to identify the row and to select the subset of the data

Index and also it is called as row label

```
warranty.tail()
```

	Row_ID	Order_ID	Order_Date	Ship_Date	Ship_Mode	Customer_ID	Customer_Name	Segment	City	State
19	45674	TZ-2014-9260	5/8/2014	5/13/2014	Standard Class	DH-3675	Duane Huffman	Home Office	Moshi	Kilimanjaro
20	41745	BU-2013-9680	9/9/2013	9/14/2013	Standard Class	DH-3675	Duane Huffman	Home Office	Sofia	Sofiya-Grad
21	42851	TU-2013-3950	4/5/2013	4/6/2013	First Class	DH-3675	Duane Huffman	Home Office	Istanbul	Istanbul
22	43579	RS-2011-8340	4/13/2011	4/13/2011	Same Day	DH-3675	Duane Huffman	Home Office	Ufa	Bashkortostan
23	46793	IZ-2014-3850	12/11/2014	12/15/2014	Standard Class	DH-3675	Duane Huffman	Home Office	Baghdad	Baghdad

Label, integer, and mixed indexing

To make the job of indexing easier and more convenient , many operators are available

- The .loc operator : It allows label-oriented indexing
- The .iloc operator : It allows integer-based indexing
- The .ix operator : It allows mixed label and integer-based indexing

Label, integer, and mixed indexing

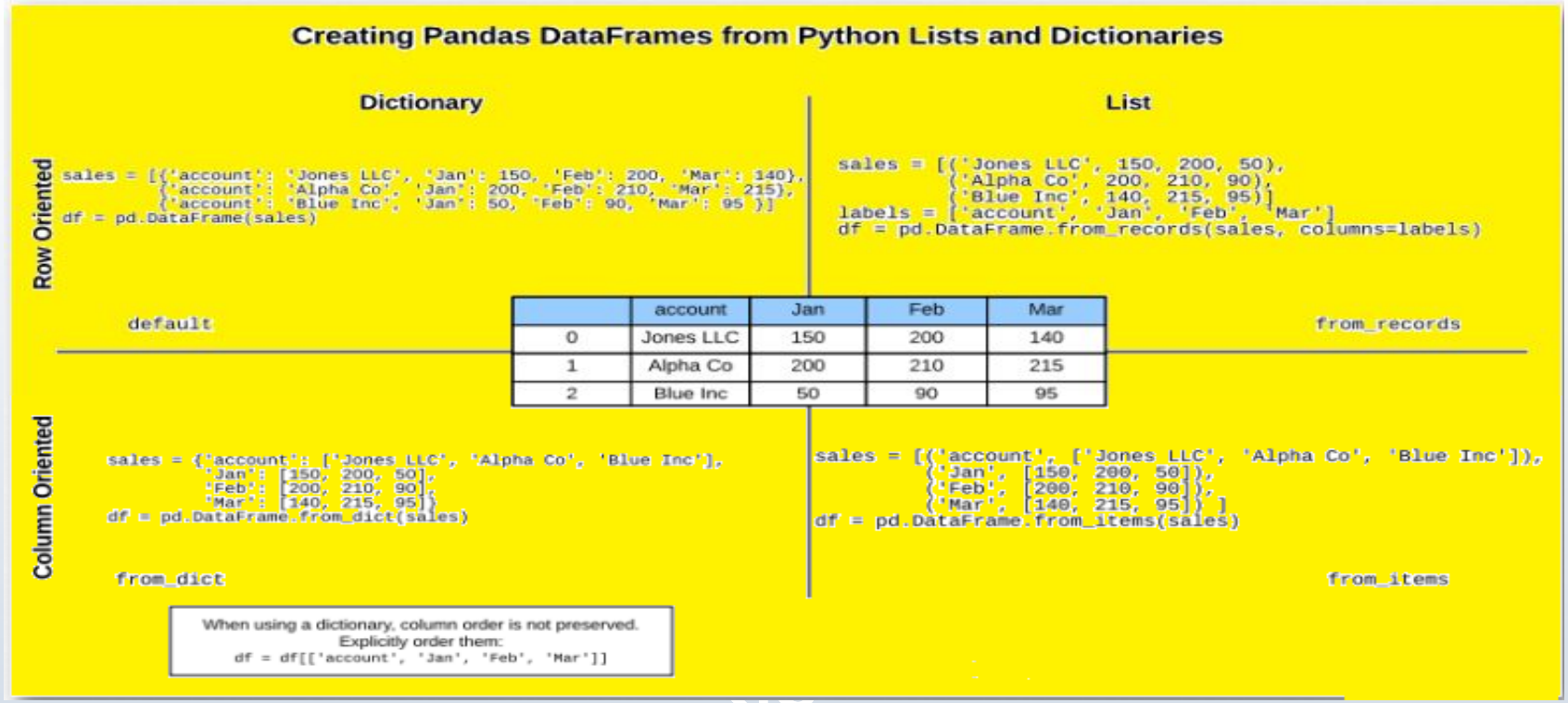
A. Integer based index

	A	B
0	10	no
1	20	yes
2	40	yes
3	50	no

B. Label index

	A	B
a	10	no
b	20	yes
c	40	yes
d	50	no

Creating data frame using list and Dictionary



Selecting Columns & Rows

To select specific columns

```
warranty[['State','Quantity']].head(3)
```

	State	Quantity
0	Analamanga	6.0
1	Kinshasa	1.0
2	Al Buhayrah	4.0

To select specific Rows

```
warranty.loc[0]
warranty.loc[10:12]
```

	Row ID	Order_ID	Order_Date	Ship_Date	Ship_Mode	Customer_ID	Customer Name	Segment	City	State	...	Sub-Category
10	42849	TU-2013-3950	4/5/2013	4/6/2013	First Class	DH-3675	Duane Huffman	Home Office	Istanbul	Istanbul	...	Machines
11	43581	RS-2011-8340	4/13/2011	4/13/2011	Same Day	DH-3675	Duane Huffman	Home Office	Ufa	Bashkortostan	...	Art
12	41521	MA-2013-1850	10/14/2013	10/15/2013	First Class	DH-3675	Duane Huffman	Home Office	Antananarivo	Analamanga	...	Binders

3 rows × 26 columns

Selecting columns in Pandas

```
# select the 'Product Name' Series using bracket notation
warranty['Product Name']

# or equivalently, use dot notation if there is no space in your column
warranty.City
```

```
0       Antananarivo
1            Kinshasa
2            Damanhur
3              Vienna
4            Istanbul
5              Bejaia
6                 Ufa
7            Kinshasa
8                 Ufa
9      Port Harcourt
10           Istanbul
11                Ufa
12       Antananarivo
13              Sofia
14                Ufa
15            Baghdad
16             Bejaia
17           Brampton
18            Baghdad
19              Moshi
20              Sofia
21           Istanbul
22                Ufa
23            Baghdad
Name: City, dtype: object
```

Most of the time **Bracket notation** willwork, whereas **dot notation** has limitations:

- Dot notation doesn't work if there are **spaces** in the Series name
- Dot notation doesn't work if the Series has the same name as a **DataFrame method or attribute** (like 'head' or 'shape')
- Dot notation can't be used to define the name of a **new Series**

Sorting a pandas data frame

6. How to sort a pandas DataFrame or a Series?

```
# sort the 'title' Series in ascending order (returns a Series)
warranty.Ship_Mode.sort_values().head()
```

```
12    First Class
4     First Class
21    First Class
10    First Class
6        Same Day
Name: Ship_Mode, dtype: object
```

```
# sort in descending order instead
warranty.Ship_Mode.sort_values(ascending=False).head()
```

```
23    Standard Class
16    Standard Class
5     Standard Class
9     Standard Class
13    Standard Class
Name: Ship_Mode, dtype: object
```

```
# sort the DataFrame first by 'content_rating', then by 'duration'
warranty.sort_values(['Ship_Mode', 'Shipping_Cost']).head()
```

Renaming column names

4. How to rename columns in a pandas DataFrame?

```
: # examine the column names
  warranty.columns
```

```
: Index([u'Row ID', u'Order ID', u'Order Date', u'Ship Date', u'Ship Mode',
         u'Customer ID', u'Customer Name', u'Segment', u'City', u'State',
         u'Country', u'Postal Code', u'Market', u'Region', u'Product ID',
         u'Category', u'Sub-Category', u'Product Name', u'Sales', u'Quantity',
         u'Discount', u'Profit', u'Shipping Cost', u'Order Priority', u'Return',
         u'Returned'],
        dtype='object')
```

```
: # rename two of the columns by using the 'rename' method
  warranty.rename(columns={'Customer Name':'Customer_Name', 'Product Name':'Product_Name'}, inplace=True)
  warranty.columns
```

```
: Index([u'Row ID', u'Order ID', u'Order Date', u'Ship Date', u'Ship Mode',
         u'Customer ID', u'Customer_Name', u'Segment', u'City', u'State',
         u'Country', u'Postal Code', u'Market', u'Region', u'Product ID',
         u'Category', u'Sub-Category', u'Product_Name', u'Sales', u'Quantity',
         u'Discount', u'Profit', u'Shipping Cost', u'Order Priority', u'Return',
         u'Returned'],
        dtype='object')
```

```
: # replace all spaces with underscores in the column names by using the 'str.replace' method
  warranty.columns = warranty.columns.str.replace(' ', '_')
  warranty.columns
```

```
: Index([u'Row_ID', u'Order_ID', u'Order_Date', u'Ship_Date', u'Ship_Mode',
         u'Customer_ID', u'Customer_Name', u'Segment', u'City', u'State',
         u'Country', u'Postal_Code', u'Market', u'Region', u'Product_ID',
         u'Category', u'Sub-Category', u'Product_Name', u'Sales', u'Quantity',
         u'Discount', u'Profit', u'Shipping_Cost', u'Order_Priority', u'Return',
```

Removing columns

5. How do I remove columns from a pandas DataFrame?

```
# remove a single column (axis=1 refers to columns)
warranty.drop('Discount', axis=1, inplace=True)
warranty.head()
```

	Row_ID	Order_ID	Order_Date	Ship_Date	Ship_Mode	Customer_ID	Customer_Name	Segment	City	State	...	Category
0	41520	MA-2013-1850	10/14/2013	10/15/2013	First Class	DH-3675	Duane Huffman	Home Office	Antananarivo	Analamanga	...	Technology
1	47596	CG-2012-2700	12/22/2012	12/24/2012	Second Class	DH-3675	Duane Huffman	Home Office	Kinshasa	Kinshasa	...	Furniture
2	45892	EG-2014-710	9/3/2014	9/9/2014	Standard Class	DH-3675	Duane Huffman	Home Office	Damanhur	Al Buhayrah	...	Furniture
3	48904	AU-2014-6170	2/17/2014	2/20/2014	Second Class	DH-3675	Duane Huffman	Home Office	Vienna	Vienna	...	Technology
4	42850	TU-2013-3950	4/5/2013	4/6/2013	First Class	DH-3675	Duane Huffman	Home Office	Istanbul	Istanbul	...	Office Supplies

5 rows × 25 columns

Filtering with Boolean indexing

Instead of selecting rows by labels, we can also select rows satisfying specific properties.

```
warranty.loc[warranty.City == 'Istanbul']
```

	Row ID	Order_ID	Order_Date	Ship_Date	Ship_Mode	Customer_ID	Customer Name	Segment	City	State	...	Sub-Category	Product Name
4	42850	TU-2013-3950	4/5/2013	4/6/2013	First Class	DH-3675	Duane Huffman	Home Office	Istanbul	Istanbul	...	Appliances	Cuisinart Microwave, Red
10	42849	TU-2013-3950	4/5/2013	4/6/2013	First Class	DH-3675	Duane Huffman	Home Office	Istanbul	Istanbul	...	Machines	Epson Phone, White
21	42851	TU-2013-3950	4/5/2013	4/6/2013	First Class	DH-3675	Duane Huffman	Home Office	Istanbul	Istanbul	...	Supplies	Kleencut Ruler, Easy Grip

3 rows × 26 columns

Group by

7 How to perform "groupby" in pandas?

In [37]:
```
# calculate the mean beer servings just for countries in Africa
warranty[warranty.Ship_Mode=='First Class'].Shipping_Cost.mean()
warranty[warranty.Ship_Mode=='First Class'].Shipping_Cost.sum()
warranty[warranty.Ship_Mode=='First Class'].Shipping_Cost.min()
warranty[warranty.Ship_Mode=='First Class'].Shipping_Cost.max()
```

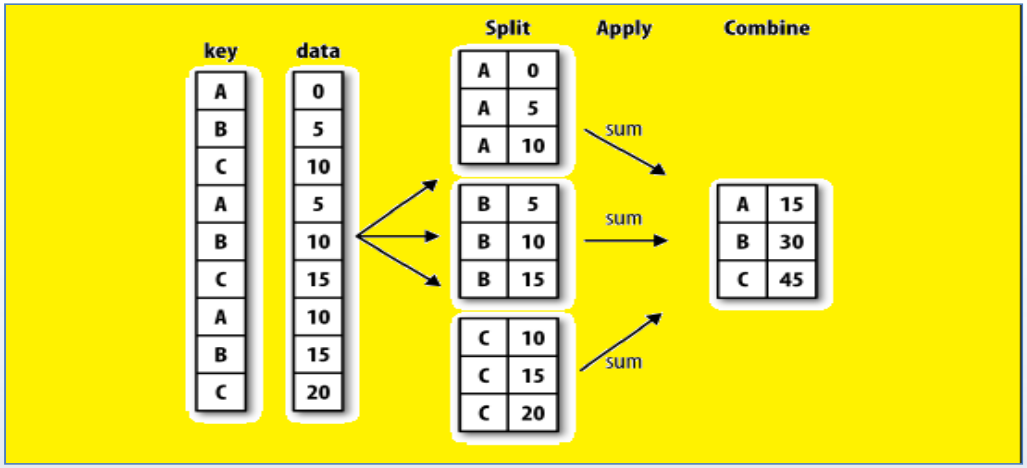
Out[37]: 1.2

In [38]:
```
# Can I apply multiple aggregation functions  simultaneously
warranty.groupby('Ship_Mode').Shipping_Cost.agg(['count', 'mean', 'min', 'max'])
```

Out[38]:

Ship_Mode	count	mean	min	max
First Class	4	12.322500	1.20	36.82
Same Day	5	9.936000	1.20	28.68
Second Class	2	29.020000	19.04	39.00
Standard Class	11	9.108182	0.93	41.78

Mechanics of Group by - Split-apply-combine



- **Step 1** – split the data into groups based on one or more keys

  Splitting is performed on a particular axis, a data frame can be grouped on its **rows (axis =0) or its columns(axis=1)**

- **Step 2-** A function is applied to each group , producing a new value

- **Step 3 –** The results of all those function applications are combined into a result object

Agenda – Dealing with missing values

- Introduction
- Data Imputation strategies
- Exploring Missing Data
- Performing Data Imputations

Data Imputation

- Some machine learning model's don't do well with missing values, so replacing the missing values could prove useful
- Missing values can be replaced by mean, median or mode

- Imputing the missing values can give better results than discarding the samples containing any missing value
- Imputing does not always improve the predictions
- In some cases, dropping rows or using marker values is more effective

Approaches of treating the Missing Values

Mean Imputation

- Replace the missing values with the mean of that variable

Conditional mean imputation

- Replace missing values with value of the variable mean for a relevant subgroup

Deletion method

- If the number of missing values are very minimal in the data set and removing them doesn't have any impact then remove those observations
- List wise and pair wise deletion
- A proportion of missing data is less than 15% but the disadvantage is it may remove a considerable fraction of data.
- Listwise deletion (complete-case analysis) removes all data for a case that has one or more missing values
- Pairwise deletion (available-case analysis) attempts to minimize the loss that occurs in listwise deletion.
- **Pairwise deletion** occurs when the statistical procedure uses cases that contain some missing data.
- The procedure cannot include a particular variable when it has a missing value, but it can still use the case when analyzing other variables with non-missing values. A case may contain 3 variables: VAR1, VAR2, and VAR3. A case may have a missing value for VAR1, but this does not prevent some statistical procedures from using the same case to analyze variables VAR2 and VAR3

Dealing with missing data

- In Pandas, missing data is represented by NaN (Not a Number) or None.
- Pandas offers several Series and DataFrame methods to handle missing data
  - isnull() indicates whether values are null or not
  - notnull() indicates the opposite
  - dropna() removes missing data
  - fillna(some_default_value) replaces missing data with a default value

Dealing with missing data

```
In [21]: from numpy import nan as NA
         data = Series([NA,5.2,NA,3.2,4.5])
         print data
```

```
0    NaN
1    5.2
2    NaN
3    3.2
4    4.5
dtype: float64
```

Filtering out Missing Data You have a number of options for filtering out missing data. While doing it by hand is always an option, dropna can be very helpful. On a Series, it returns the Series with only the non-null data and index values:

```
In [22]: data.dropna()
```

```
Out[22]: 1    5.2
         3    3.2
         4    4.5
         dtype: float64
```

Passing how='all' will only drop rows that are all NA:

```
In [ ]: data.dropna(how='all')
```

```
In [37]: df = DataFrame(np.random.randn(7, 3))
         print df
```

```
          0         1         2
0 -0.577207  0.916780  0.213985
1 -0.315045 -1.295961  0.337726
2  1.859421  2.083067 -0.545897
3  0.868045  0.175680  1.131219
4  0.013106 -1.014764 -1.664611
5 -2.014061 -2.303613 -0.239133
6 -0.256915 -0.820610 -0.821980
```

```
In [38]: df.ix[:4, 1] = NA; df.ix[:2, 2] = NA
         print df
```

```
          0         1         2
0 -0.577207       NaN       NaN
1 -0.315045       NaN       NaN
2  1.859421       NaN       NaN
3  0.868045       NaN  1.131219
4  0.013106       NaN -1.664611
5 -2.014061 -2.303613 -0.239133
6 -0.256915 -0.820610 -0.821980
```

Merging Data sets

- DataFrame.merge is used to mix the two or more data frames based on the columns provided and join mentioned
- If joining columns on columns, the data frame indexes will be ignored.
- If joining indexes on indexes or columns on columns , the indexes will be passed on

Join method	Sql Join	Description
Left	LEFT OUTER JOIN	Use keys from left frame only
Right	RIGHT OUTER JOIN	Use keys from right frame only
Outer	FULL OUTER JOIN	Use union of keys from both frames
Inner	INNER JOIN	Use intersection of keys from both frames

Combining (Joining) Data sets

- Join function combines columns with other data frame either on index or on a key column
- Use DataFrame.join instead of merge operation if you are joining on index only
- DataFrame.join(objs, on=None,how='left', lsuffix = '',rsuffix='', sort=False)
- Two data frames can be joined together over a unique id (identified common in both) and have all types of join operations performed on them

Difference between merge() , concat() and join in pandas

merge()

- merge() is used to combine two (or more) data frames on the basis of values of common columns(indexes can also be used, use left_index=True and/or right index=True)
- **Merge on columns**

  df_new = pd.merge(left=df1, right=df2, how='outer', left_index = True, right_index = True)
- **Merge on columns**

  df_new = pd.merge(left=df1, right=df2, how='left', left_on='col1', right_on='col2')

Join() (Another way of merging the data frames)

- join() is used to merge 2 data frames on the basis of the index; instead of using merge() with the option left_index=True we can use join().

  df_new = df1.join(other=df2, on='col1', how='outer')

  df_new = df1.join(other=df2,on=['a','b'], how='outer') Note: DataFrame.join() joins on indexes by default

- concat() is used to append one (or more) data frames one below the other (or sideways, depending on whether the axis option is set to 0 or 1)

Important tip

Note: DataFrame.join() joins on indexes by default.

DataFrame.merge() joins on common columns by default.

Data sets

Student Data Frame

	Class Id	Students	
0	Id1	Trevor	
1	Id2	Chris	
2	Id4	John	
3	Id6	David	

Subject Data Frame

	SectionId	Semester	Subjects
0	Id1	First	Maths
1	Id2	Second	Science
2	Id3	Frist	Computer
3	Id4	Third	History

cars1 Data Frame

	Color	Name	Year
0	Red	Bmw	2012
1	White	Lexus	2014
2	Black	Ferrari	2011
3	Blue	Mercedez	2016

cars2 Data Frame

	Color	Name	Year
4	Grey	Audi	2011
5	Brown	Hyndai	2014
6	Yellow	Maruti	2012
7	White	Ford	2015

cars Details

	Country	Units	Variant
0	germany	2500	Diesel
1	France	4700	Petrol
2	US	3250	Petrol
3	UK	6470	diesel

Assignments

1. Create a list for the cars 1 data set
2. How to get help for the list arguments
3. Print the type
4. Save the data frame as a .csv file and read the same as df3
5. Select row 2 through row 3
6. Display the car models before 2014
7. Change the brand name of all rows with an year greater than 2014 to "bmw“ and assign it to a new data frame df5
8. Create an index on Brands and locate the bmw brand from df3 data frame
9. Create a dictionary for the cars 2 data set
10. Print the type
11. Convert both cars 1 list & cars 2 dictionary as DataFrame
12. What is axis 0 and axis 1 ?
13. Concat both car 1 & car 2 data set
14. Concat both car 1
15. Merge the data frames car 1 & car2 , car2 & car1
16. Join the data frames car 1 & car2 , car2 & car1
17. Explain the difference between concat, merge , and Join

1.Concat the cars1 and cars 2 data with axis = 0 (row wise)
2.Concat the cars1 and cars 2 data with axis = 1 (column wise)

Pandas Quiz

What is a DataFrame object ?

The pandas DataFrame is a two dimensional table of data with column and row indexes. The columns are made up of pandas Series objects.

What is a Series object ?

Series object: an ordered, one-dimensional array of data with an index. All the data in a Series is of the same data type. Series arithmetic is vectorised after first aligning the Series index for each of the operands

What is an index object ?

The pandas Index provides the axis labels for the Series and Data Frame objects.
It can only contain hashable objects.

A pandas Series has one Index; and a DataFrame has two Indexes.

Reshaping the data

dataframe.pivot (index='index', columns= 'columns', values = 'value')

Returns either a data frame or panel , depending on whether you request single value column(data frame) or all columns (panel))

Region	Year	Revenue
North	2015	11000
North	2014	12500
South	2015	13800
West	2015	14500
East	2014	13509
North	2014	10250
South	2014	13204
West	2014	11223

Assignments

Versions of required libraries :

- Pandas - 0.18.1
- Numpy - 1.11.1
- Matplotilb - 1.5.1
- re - 2.2.1

Note: Required libraries pandas, numpy are imported as pd and np. Additional details are provided in the questions itself.

Loan_ID	Gender	Married	Dependents	Education	ApplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
LP001002	Male	No	0	Graduate	5849	NaN	360.0	1.0	Urban	Y
LP001003	Male	Yes	1	Graduate	4583	128.0	360.0	1.0	Rural	N
LP001005	Male	Yes	0	Graduate	3000	66.0	360.0	1.0	Urban	Y
LP001006	Male	Yes	0	Not Graduate	2583	120.0	360.0	1.0	Urban	Y
LP001008	Male	No	0	Graduate	6000	141.0	360.0	1.0	Urban	Y

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **How can you find the number of missing values in the column "LoanAmount"?**
- train.count().maximum() - train.LoanAmount.count()
- (train.LoanAmount == NaN).sum()
- (train.isnull().sum()).LoanAmount
- All of these

Assignments

Versions of required libraries :

- Pandas - 0.18.1
- Numpy - 1.11.1
- Matplotilb - 1.5.1
- re - 2.2.1

Note: Required libraries pandas, numpy are imported as pd and np. Additional details are provided in the questions itself.

Loan_ID	Gender	Married	Dependents	Education	ApplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
LP001002	Male	No	0	Graduate	5849	NaN	360.0	1.0	Urban	Y
LP001003	Male	Yes	1	Graduate	4583	128.0	360.0	1.0	Rural	N
LP001005	Male	Yes	0	Graduate	3000	66.0	360.0	1.0	Urban	Y
LP001006	Male	Yes	0	Not Graduate	2583	120.0	360.0	1.0	Urban	Y
LP001008	Male	No	0	Graduate	6000	141.0	360.0	1.0	Urban	Y

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **You need to create a new DataFrame named "new_dataframe", which contains rows which have a non-missing value for variable"Credit_History" in our DataFrame "train". Which of the following commands would do this?**
- new_dataframe = train[~train.Credit_History.isnull()]
- new_dataframe = train[train.Credit_History.isna()]
- new_dataframe = train[train.Credit_History.is_na()]
- None of these

Assignments

Versions of required libraries :

- Pandas - 0.18.1
- Numpy - 1.11.1
- Matplotilb - 1.5.1
- re - 2.2.1

Note: Required libraries pandas, numpy are imported as pd and np. Additional details are provided in the questions itself.

Loan_ID	Gender	Married	Dependents	Education	ApplicantIncome	LoanAmount	Loan_Amount_Term	Credit_History	Property_Area	Loan_Status
LP001002	Male	No	0	Graduate	5849	NaN	360.0	1.0	Urban	Y
LP001003	Male	Yes	1	Graduate	4583	128.0	360.0	1.0	Rural	N
LP001005	Male	Yes	0	Graduate	3000	66.0	360.0	1.0	Urban	Y
LP001006	Male	Yes	0	Not Graduate	2583	120.0	360.0	1.0	Urban	Y
LP001008	Male	No	0	Graduate	6000	141.0	360.0	1.0	Urban	Y

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **In the dataset above, you can see row with Loan_id = LP001005 has very little information (i.e. most of the variables are missing). It is recommended to filter out these rows as they could create problems / noise in your model.**

  If a row contains more than 5 missing values, you decide to drop them and store remaining data in DataFrame "temp". Which of the following commands will achieve that?

- temp = train.dropna(axis=0, how='any', thresh=5)
- temp = train.dropna(axis=0, how='all', thresh=5)
- temp = train.dropna(axis=0, how='any', thresh=train.shape[1] - 5)
- None of these

Assignments

- **6) Now, it is time to slice and dice data. The first logical step is to make data ready for your machine learning algorithm. In the dataset, you notice that number of rows having "Property_Area" equal to "Semiurban" is very low. After thinking and talking to your business stakeholders, you decide to combine "Semiurban" and "Urban" in a new category "City" . You also decide to rename "Rural" to "Village"**

  Which of the following commands will make these changes in the column 'Property_Area' ?

- >>> turn_dict = ['Urban': 'City', 'Semiurban': 'City', 'Rural': 'Village']
  >>> train.loc[:, 'Property_Area'] = train.Property_Area.replace(turn_dict)
- >>> turn_dict = {'Urban': 'City', 'Semiurban': 'City', 'Rural': 'Village'}
  >>> train.loc[:, 'Property_Area'] = train.Property_Area.replace(turn_dict)
- >>> turn_dict = {'Urban, Semiurban': 'City', 'Rural': 'Village'}
  >>> train.iloc[:, 'Property_Area'] = train.Property_Area.update(turn_dict)
- None of these

Assignments

- **Q 7) While you were progressing in direction of building your first machine learning model, you notice something interesting. On a quick overview of the first few rows, you see that percentage of people who are "Male" and are married (Married = "Yes") seems high.**

  To check this hypothesis, how will you find the percentage of married males in the data?

- (train.loc[(train.Gender == 'male') && (train.Married == 'yes')].shape[1] / float(train.shape[0]))*100
- (train.loc[(train.Gender == 'Male') & (train.Married == 'Yes')].shape[1] / float(train.shape[0]))*100
- (train.loc[(train.Gender == 'male') and (train.Married == 'yes')].shape[0] / float(train.shape[0]))*100
- None of these

Assignments

train dataset :

User_ID	Product_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Purchase
1000001	P00069042	F	0-17	10	A	0	3	8370
1000001	P00248942	F	0-17	10	A	0	1	15200
1000001	P00087842	F	0-17	10	A	0	12	1422
1000001	P00085442	F	0-17	10	A	0	12	1057
1000002	P00285442	M	55+	16	C	0	8	7969

test dataset :

Unnamed: 0	User_ID	Product_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Comb
0	1000004	P00128942	M	46-50	7	B	0	1	1000004P00128942
1	1000009	P00113442	M	26-35	17	C	0	3	1000009P00113442
2	1000010	P00288442	F	36-45	1	B	0	5	1000010P00288442
3	1000010	P00145342	F	36-45	1	B	0	4	1000010P00145342
4	1000011	P00053842	F	26-35	1	C	0	4	1000011P00053842

- **Take a brief look at train and test datasets mentioned above. You might have noticed that the columns in these datasets do not match, i.e. some columns in train are not present in test and vice versa.**

  How to find which cols are present in test but not in train? Assume data has already been read in DataFrames "train" & "test" respectively.

- set(test.columns).difference(set(train.columns))
- set(test.columns.tolist()) - set(train.columns.tolist())
- set(train.columns.tolist()).difference(set(test.columns.tolist()))
- Both A and B

Assignments

User_ID	Product_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Purchase
1000001	P00069042	F	0-17	10	A	0	3	8370
1000001	P00248942	F	0-17	10	A	0	1	15200
1000001	P00087842	F	0-17	10	A	0	12	1422
1000001	P00085442	F	0-17	10	A	0	12	1057
1000002	P00285442	M	55+	16	C	0	8	7969

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **As you might be aware, most of the machine learning libraries in Python and their corresponding algorithms require data to be in numeric array format.**

  Hence, we need to convert categorical "Gender" values to numerical values (i.e. change M to 1 and F to 0). Which of the commands would do that?

- train.ix[:, 'Gender'] = train.Gender.applymap({'M':1,'F':0}).astype(int)
- train.ix[:, 'Gender'] = train.Gender.map({'M':1,'F':0}).astype(int)
- train.ix[:, 'Gender'] = train.Gender.apply({'M':1,'F':0}).astype(int)
- None of these

Assignments

train dataset:

User_ID	Product_ID
1000001	P00069042
1000001	P00248942
1000001	P00087842
1000001	P00085442
1000002	P00285442

test dataset:

User_ID	Product_ID
1000004	P00069042
1000009	P00069042
1000010	P00085442
1000010	P00087842
1000011	P00085442

NOTE: The datasets have been loaded in python using 'pandas' library in the form of DataFrame object named "train" and "test".

- **In the datasets above, "Product_ID" column contains a unique identification of the products being sold. There might be a situation when there are a few products present in the test data but not in train data. This could be troublesome for your model, as it has no "historical" knowledge for the new product.**

  How would you check if all values of "Product_ID" in test DataFrame are available in train DataFrame dataset?
- train.Product_ID.unique().contains(test.Product_ID.unique())
- set(test.Product_ID.unique()).issubset(set(train.Product_ID.unique()))
- train.Product_ID.unique() = test.Product_ID.unique()
- None of these

Assignments

User_ID	Product_ID	Gender	Age	Occupation	City_Category	Marital_Status	Product_Category_1	Purchase
1000001	P00069042	F	0-17	10	A	0	3	8370
1000001	P00248942	F	0-17	10	A	0	1	15200
1000001	P00087842	F	0-17	10	A	0	12	1422
1000001	P00085442	F	0-17	10	A	0	12	1057
1000002	P00285442	M	17-25	16	C	0	8	7969

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **If you look at the data above, "Age" is currently a categorical variable. Converting it to a numerical field might help us extract more meaningful insight.**

  You decide to replace the Categorical column 'Age' by a numeric column by replacing the range with its average (Example: 0-17 and 17-25 should be replaced by their averages 8.5 and 21 respectively)

- train['Age'] = train.Age.apply(lambda x: (np.array(x.split('-'), dtype=int).sum()) / x.shape)
- train['Age'] = train.Age.apply(lambda x: np.array(x.split('-'), dtype=int).mean())
- Both of these
- None of these

Assignments

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **The other scenario in which numerical value could be "hiding in plain sight" is when it is plagued with characters. We would have to clean these values before moving on to model building.**

  For example, in "Ticket", the values are represented as one or two blocks separated with spaces. Each block has numerical values in it, but only the first block has characters combined with numbers. (eg. "ABC0 3000").

  Which of the following code return only the last block of numeric values? (You can assume that numeric values are always present in the last block of this column)

- train.Ticket.str.split(' ').str[0]
- train.Ticket.str.split(' ').str[-1]
- train.Ticket.str.split(' ')
- None of these

Assignments

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	NaN	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **As you might have noticed (or if you haven't, do it now!), the above dataset is the famous Titanic dataset. (PS: its a bit unclean than usual, but you get the gist, right?)**

  Coming back to the point; the data has missing values present in it. It is time to tackle them! The simplest way is to fill them with "known" values.

  You decide to fill missing "Age" values by mean of all other passengers of the same gender. Which of the following code will fill missing values for all passengers by the above logic?

- train = train.groupby('Sex').transform(lambda x: x.fillna(x.sum()))
- train['Age'] = train.groupby('Sex').transform(lambda x: x.fillna(x.mean())).Age
- train['Age'] = train.groupby('Sex').replace(lambda x: x.fillna(x.mean())).Age
- None of these

Assignments

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **You can see that column "Cabin" has 3 missing values out 5 sample records.**

  If a particular column has a high percentage of missing values, we may want to drop the column entirely. However, this might also lead to loss of information.

  Another method to deal with this type of variable, without losing all information, is to create a new column with flag of missing value as 1 otherwise 0.

  Which of the following code will create a new column "Missing_Cabin" and put the right values in it (i.e. if "cabin_missing" then 1 else 0)?

- train['Missing_Cabin'] = train.Cabin.apply(lambda x: x == '')
- train['Missing_Cabin'] = train.Cabin.isnull() == False
- train['Missing_Cabin'] = train.Cabin.isnull().astype(int)
- None of these

Assignments

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **You can see that column "Cabin" has 3 missing values out 5 sample records.**

  If a particular column has a high percentage of missing values, we may want to drop the column entirely. However, this might also lead to loss of information.

  Another method to deal with this type of variable, without losing all information, is to create a new column with flag of missing value as 1 otherwise 0.

  Which of the following code will create a new column "Missing_Cabin" and put the right values in it (i.e. if "cabin_missing" then 1 else 0)?

- train['Missing_Cabin'] = train.Cabin.apply(lambda x: x == '')
- train['Missing_Cabin'] = train.Cabin.isnull() == False
- train['Missing_Cabin'] = train.Cabin.isnull().astype(int)
- None of these

Assignments

train.csv

FDF22	6.865	Low Fat	0.056783
FDS36	8.380	Regular	0.046982
NCJ29	10.600	Low Fat	0.035186
FDN46	7.210	Regular	0.145221
DRG01	14.800	Low Fat	0.044878

Column Names:

Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility

NOTE: Pandas library has been imported by the name "pd".

- **Let us take a look at another dataset. The data represents sales of an outlet along with product attributes.**

  The problem is, the dataset does not contain headers. Inspite of this, you know what are the appropriate column names. How would you read the the dataframe by specifying the column names?

- pd.read_csv("train.csv", header=None, columns=['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility'])
- pd.read_csv("train.csv", header=None, usecols=['Item_Identifier', 'Item_Weight', 'Item_Fat_Content', 'Item_Visibility'])
- pd.read_csv("train.csv", header=None, names=['Item_Identifier' ,'Item_Weight' ,'Item_Fat_Content', 'Item_Visibility'])
- None of these

Assignments

Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Size	Outlet_Type	Item_Outlet_Sales
FDA15	9.30	low fat	0.016047	Dairy	249.8092	OUT049	Medium	Supermarket Type1	3735.1380
DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	Medium	Supermarket Type2	443.4228
FDN15	17.50	LF	0.016760	Meat	141.6180	OUT049	Medium	Supermarket Type1	2097.2700
FDX07	19.20	reg	0.000000	Fruits and Vegetables	182.0950	OUT010	NaN	Grocery Store	732.3800
NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	High	Supermarket Type1	994.7052

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

- **Sometimes while reading the data in pandas, the datatypes of columns are not parsed correctly. To deal with this problem, you can either explicilty specify datatypes while reading the data, or change the datatypes in the dataframe itself.**
- **Which of the following code will change the datatype of "Item_Fat_Content" column from "object" to "category"?**
- train['Item_Fat_Content'] = train['Item_Fat_Content'].asdtype('categorical')
- train['Item_Fat_Content'] = train['Item_Fat_Content'].astype('category')
- train['Item_Fat_Content'] = train['Item_Fat_Content'].asdtype('category')
- None of these

Assignments

Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Size	Outlet_Type	Item_Outlet_Sales
FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	Medium	Supermarket Type1	3735.1380
DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	Medium	Supermarket Type2	443.4228
FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	Medium	Supermarket Type1	2097.2700
FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	NaN	Grocery Store	732.3800
NCD19	8.93	Low Fat	0.000000	Household	53.8614	OUT013	High	Supermarket Type1	994.7052

NOTE: The dataset has been loaded in python using 'pandas' library in the form of DataFrame object named "train".

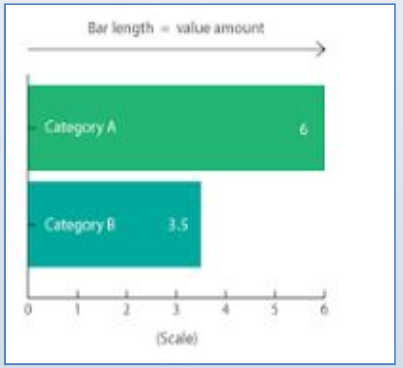
- **In above data, notice that the "Item_Identifier" column has some relation with the column "Item_Type". As the first letter of "Item_Identifier" changes, the "Item_Type" changes too. For example, notice that if the value in "Item_Identifier" starts with "F", then all the corresponding values in "Item_Type" are eatables, whereas those with "D" are drinks.**
- **To check this hypothesis, find all values in "Item_Identifier" that starts with "F".**
- train.Item_Identifier.str.starts_with('F')
- train.Item_Identifier.str.startswith('F')
- train.Item_Identifier.str.is_start('F')
- None of these

Agenda – Data Visualization

- Data Visualization Introduction
- Introduction to matplotlib
- Plotting bar, pie, Lines , Scatter plots , subplots using matplotlib
- Introduction to Seaborn
- Performing Univariate Analysis
- Performing Bi variate Analysis
- Faceting
- Grammar of graphics ggplot
- DV Quiz

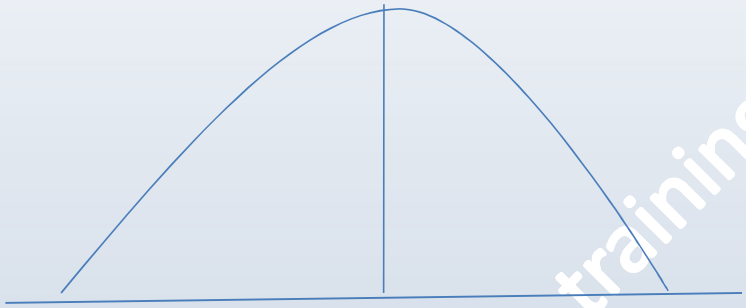
What is Data Visualization ?

- Representation of data in a pictorial or graphical format
- It's a process of converting raw data into easily understood pictures of information that enable fast and effective decisions



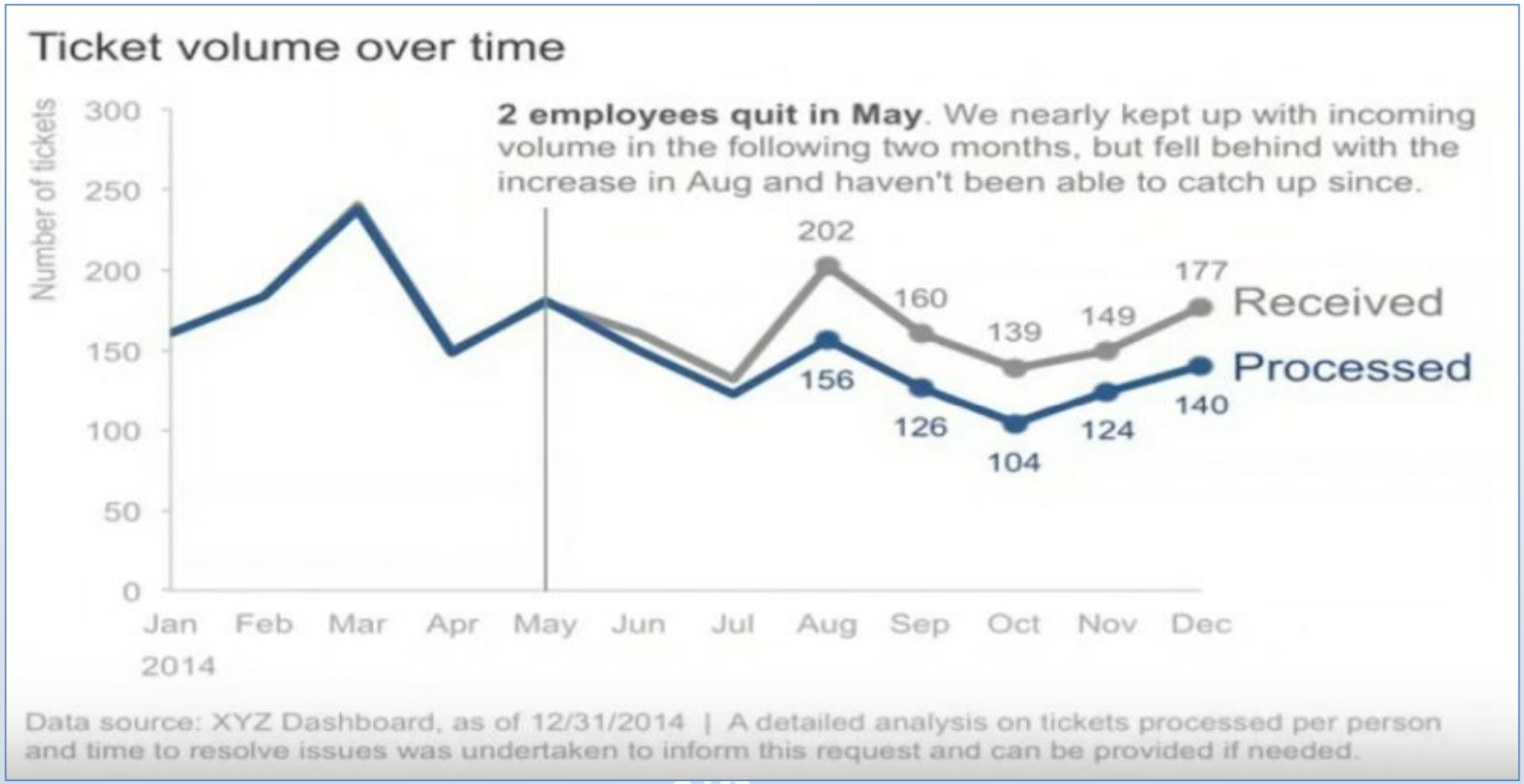
Why Data Visualization is so important ?

- Identify areas that need attention or improvement
- Clarify which factors influence customer behavior
- Helps you predict sales volumes
- Data visualization can help translate data patterns into insights, making it a highly effective decision-making tool.

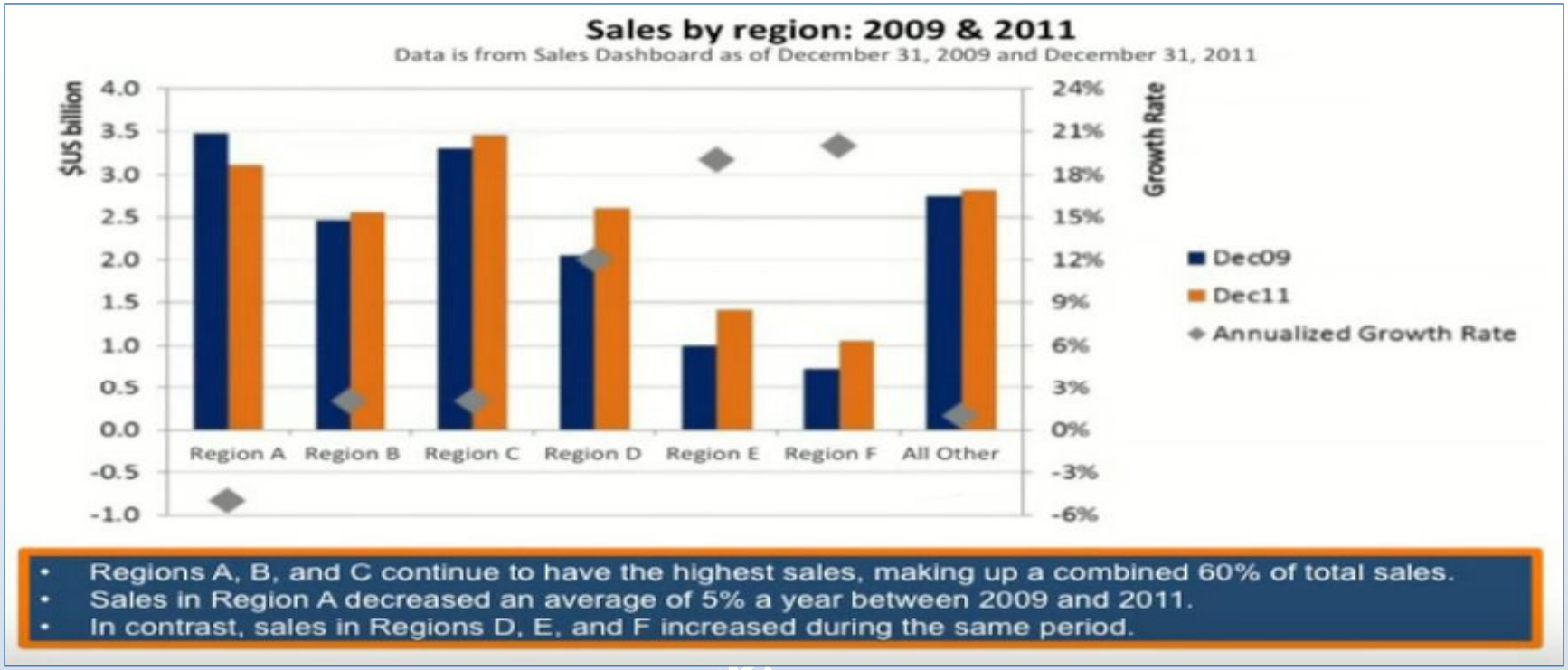


Data Visualization - Example

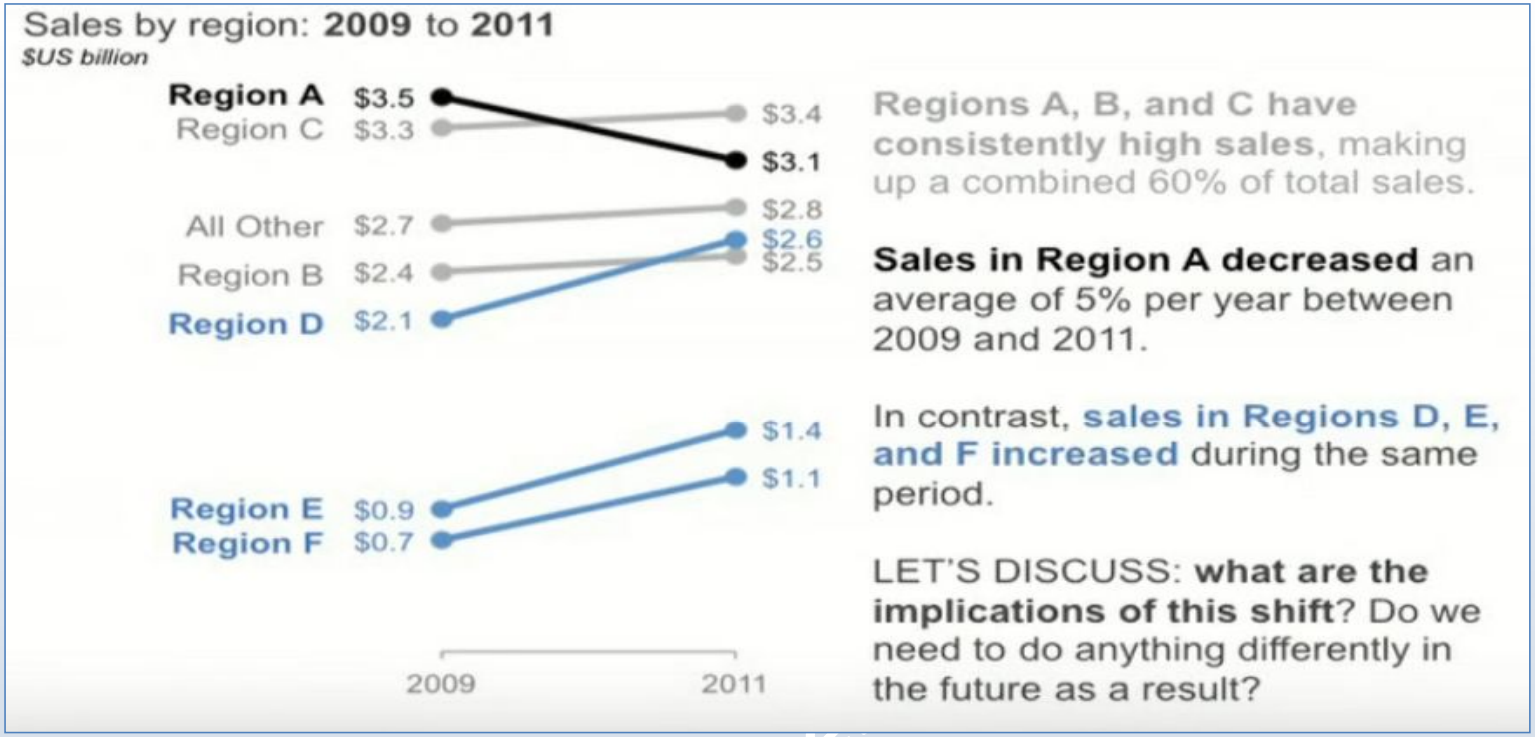
Data Visualization - Example



Data Visualization - Example



Data Visualization - Example



Matplotlib

- **conda install matplotlib**
- **import matplotlib.pyplot as plt**
- It was developed by John Hunter
- Explore the data is to communicate the outcomes using effective representation of data in a form of pictorial or graphical format
- Python offers a rich set of visualization libraries such as Matplotlib which emerged as the main source.
- Some other libraries like Seaborn primarily built on top of matplotlib
- Vispy library for interactive scientific visualization
- Bokeh which is a python interactive visualization library , it helps constructing novel graphics like **D3.js** language

Matplotlib

- To plot charts or graphs in matplotlib library , pyplot module is used which provides a procedural interface to matplotlib object-oriented plotting library.

- Majority of the commands, along with the arguments , in Pyplot are analogous to that of matplotlib

- To import the Pyplot procedural interface, simply use the import command

  from matplotlib import pyplot as plt

  Its an alias

Seaborn

- Seaborn has been developed on top of matplotlib, to overcome the potential drawbacks of matplotlib such as a data frame cannot be directly visualized ,
- It's a statistical plotting library

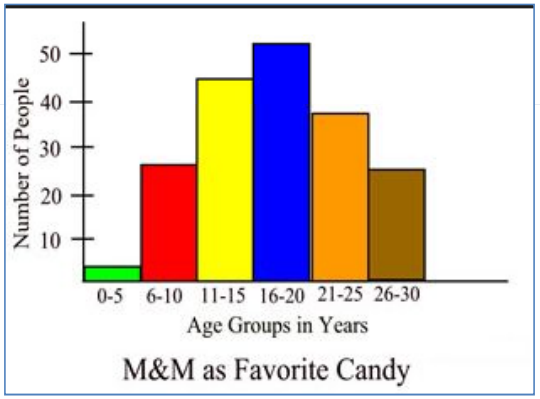
- pip install seaborn
- Uses simple functions for common statistical visualizations

- It provisions of functions to visualize univariate and **bi-variate** distributions

- Improved default matplotlib aesthetics by using various **built-in themes**

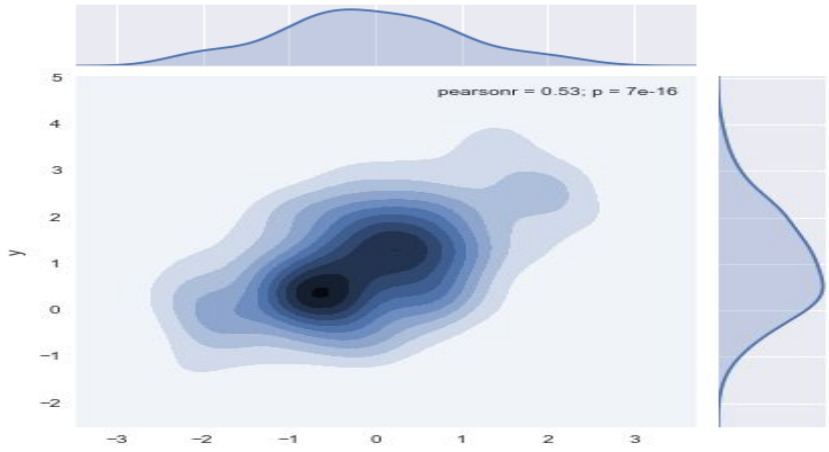
histogram

- Histogram display the distribution of a continuous variable
- Dividing up the range of scores into bins on the x-axis and displaying the frequency of scores in each bin on the y-axis



Kernel Density plot

- Histogram may not be the efficient way to view the distribution always
- Kernel density plots are usually a much more effective way to view the distribution of a variable
- It is also posible to use the kernel density estimation procedure described above to visualize a bivariate distribution. In seaborn, this kind of plot is shown with a contour plot and is available as a style in jointplot():

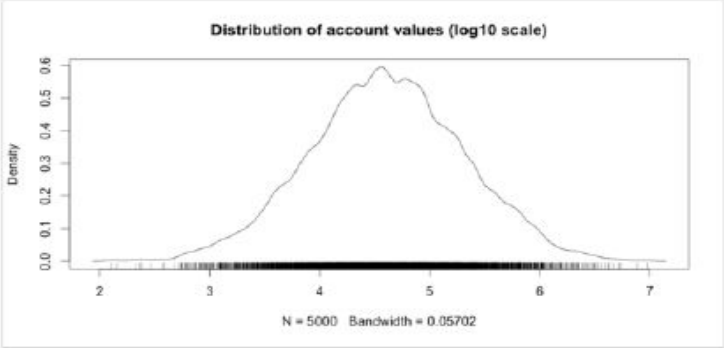


Kde vs Histogram

- Histograms an often be a poor method for determining the shape of a distribution because they are strongly affected by the number of bins used.
- For example, visualizing the same data with only four bins can make the same observations appear normally distributed.

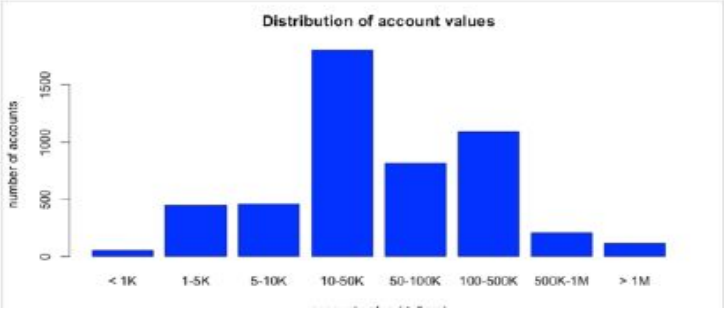
- Kde are a more effective way to illustrate the distribution of a variable. This is now surprisingly easy to do. To form a *KDP, a kernel - that is, a smooth, strongly peaked function - is placed at the position of each data point.
- The contributions from all kernels are added to obtain a smooth curve, which can be evaluated at any point along the x-axis.

Density Plot Vs Bar plot



Data Exploration:

This tells you what you need to know.



Presentation:

This tells the stakeholders what they need to know.

Heatmap

- A heatmap is a two-dimensional representation of data in which values are represented by colors.

To

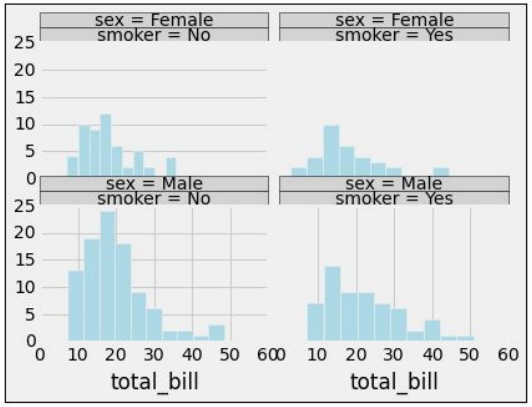
- A simple heatmap provides an immediate visual summary of information.

Trellis Plot

- A Trellis plot is a layout of smaller charts in a grid with consistent scales. Each smaller chart represents an item in a category, named conditions. The data displayed on each smaller chart is conditional for the items in the category.

To

- Trellis plots are useful for finding structures and patterns in complex data. The grid layout looks similar to a garden trellis, hence the name Trellis plots.



Hexagon bin plot

- A hexagon bin plot can be created using the DataFrame.plot() function and kind = 'hexbin'. This kind of plot is really useful if your scatter plot is too dense to interpret. It helps in binning the spatial area of the chart and the intensity of the color that a hexagon can be interpreted as points being more concentrated in this area.
- The following code helps in plotting the hexagon bin plot, and the structure of the code is similar to the previously discussed plots:

This kind of plot is really useful if your scatter plot is too dense to interpret

Faceting

Helps to visualize the distribution of a variable or to find out relationship amongst multiple variables with subsets of the data

FacetGrid class (AxisGrid API) is used to initialize a FacetGrid object. This interacts with the Pandas data frame to connect to the matplotlib

Faceting specification describes which variables should be used to split up the data, and how they should be arranged

Grammar of Graphics - ggplot

- ggplot is an implementation of grammar of graphics.
- It's a framework , for graphics, by providing structure , to combine graphical elements, into figures that display data, in a meaningful way
- This framework was developed by Wilkinson, Anand, and Grossman
- Splitting the graphic into various layers
- conda install ggplot (or) pip install –U ggplot

with ggplot we can create professional looking plots rapidly with lesser code

Element	Description
Data	The dataset being plotted
Aesthetics	The scales onto which we map our data
Geometrics (geoms for short)	The visual elements used for our data. It controls the type of plot

Grammar of Graphics

Data	Variables of interest
Aesthetics	x-axis color size alpha line width y-axis fill labels shape line type
Geometrics	Point line histogram bar boxplot
Facets	columns rows
Statistics	Binning smoothing descriptive inferential
Coordinates	Cartesian fixed polar limits
Themes	non-data ink

Components of ggplot2

- **Data** : the raw data that needs to be plotted
- **Aesthetics:** including mapping e.g., which variable is on the x-axis ? The y-axis ? Should the color/size/position of the plotted data that be mapped to some variable ?
- **Geometrics:** the geometric shapes that represent the data
- **Statistics** : Statistical transformations that are used to summarize the data

Python's strftime directives

Code	Meaning	Example
%a	Weekday as locale’s abbreviated name.	Mon
%A	Weekday as locale’s full name.	Monday
%w	Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.	1
%d	Day of the month as a zero-padded decimal number.	30
%-d	Day of the month as a decimal number. (Platform specific)	30
%b	Month as locale’s abbreviated name.	Sep
%B	Month as locale’s full name.	September
%m	Month as a zero-padded decimal number.	09
%-m	Month as a decimal number. (Platform specific)	9
%y	Year without century as a zero-padded decimal number.	13
%Y	Year with century as a decimal number.	2013
%H	Hour (24-hour clock) as a zero-padded decimal number.	07
%-H	Hour (24-hour clock) as a decimal number. (Platform specific)	7
%I	Hour (12-hour clock) as a zero-padded decimal number.	07
%-I	Hour (12-hour clock) as a decimal number. (Platform specific)	7
%p	Locale’s equivalent of either AM or PM.	AM
%M	Minute as a zero-padded decimal number.	06
%-M	Minute as a decimal number. (Platform specific)	6
%S	Second as a zero-padded decimal number.	05
%-S	Second as a decimal number. (Platform specific)	5
%f	Microsecond as a decimal number, zero-padded on the left.	000000
%z	UTC offset in the form +HHMM or -HHMM (empty string if the the object is naive).	
%Z	Time zone name (empty string if the object is naive).	
%j	Day of the year as a zero-padded decimal number.	273
%-j	Day of the year as a decimal number. (Platform specific)	273
%U	Week number of the year (Sunday as the first day of the week) as a zero padded decimal number. All days in a new year preceding the first Sunday are considered to be in week 0.	39
%W	Week number of the year (Monday as the first day of the week) as a decimal number. All days in a new year preceding the first Monday are considered to be in week 0.	39
%c	Locale’s appropriate date and time representation.	Mon Sep 30 07:06:05 2013
%x	Locale’s appropriate date representation.	09/30/13
%X	Locale’s appropriate time representation.	07:06:05
%%	A literal '%' character.	%