Data Science Made Easy – Level 1

Using python

Ramadurai Seshadri

August 2016
Ramadurai.seshadri@wipro.com

What You will Learn Today

Agenda for “Data Science Made Easy Using Python” Level 1

- Basic concepts in Python
  - variable declaration, assigning values to a variable, data types, adding comments to code
- Print statement, Conditional & Loop statements
  - print, if, for
- Reading an input from a file
  - reading a data from a text, csv, url, zip
- Functions
  - Defining & Calling a function
- Using libraries
  - Pandas, numpy, pylab
- Algorithms
  - Linear Regression

What is python™

Why now?

Python is a versatile, interpreted, general-purpose programming language (that can be compiled) which produces compact yet highly readable code.

Python Highlights:	Dynamic typing
	Portable
	Interpreted and interactive
	Easy to Learn and Use
	Object-oriented
	Truly Modular
	Automatic garbage collection
Differentiators:	Open Source
	Fastest Growing Language for Data Scientists
	Huge number of libraries for every possible use
	Need less code to write than C++ or Java

One of three "Official Languages" at Google

- Python
- Java
- C++

Who Uses Python?

- Parts of YouTube are written in Python
- Intel tests microchips using Python
- Companies such as Google, Yahoo!, Disney, Nokia, and IBM all use Python

Data Science - Demystified

What is Data Science and Why is it important now?

- Data Science is an interdisciplinary field to extract knowledge from data
- It is combination of Data Mining, Artificial Intelligence and Domain Expertise – without one or the other, it is less useful
- It has been made possible by recent advances in AI, and the explosion of Open Source toolkits such as R, Python, Hadoop and Spark which makes deriving insights from even very large data sets easier
- It is a new “Gold Rush” that due to advances in Machine Learning, Deep Learning, Probabilistic Reasoning and Natural Language Processing have made it easier to mine for gold from dirt

Data Scientists will have to know both:

1. Algorithms – this is related to predictions
2. Engineering – how to integrate disparate technologies

This a chart that is frequently used to show what Data Science is and what it is not.

Combining all three skills is Data Science

Such a person can be called a Data Scientist

R and Python for Data Science and Big Data Applications

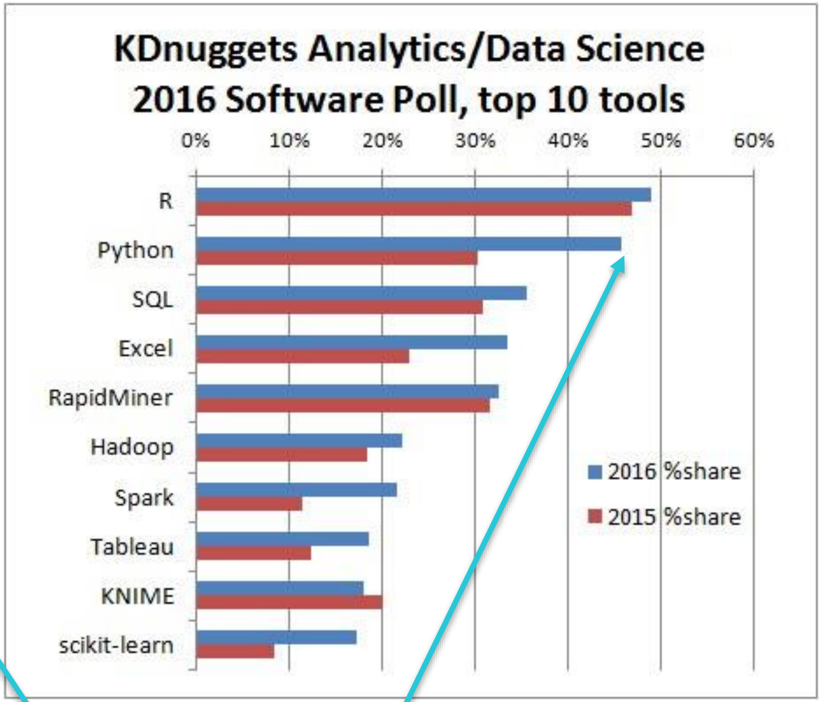
Skill	% of Big Data Jobs Mentioning This Skill Set (multiple responses allowed)	% Growth in Demand For This Skill Set Over the Previous Year
Java	6.62%	63.30%
Structured query language	5.86%	76.00%
Apache Hadoop	5.45%	49.10%
Software development	4.70%	60.30%
Linux	4.10%	76.60%
Python	3.99%	96.90%
NoSQL	2.74%	34.60%
Data warehousing	2.73%	68.80%
UNIX	2.43%	61.90%
Software as a Service	2.38%	54.10%

Source: Forbes 2014

Demand for Python Skills are the fastest growing category in Big Data skillset

What Analytics, Big Data, Data mining, Data Science software you used in the past 12 months for a real project? [2895 voters]

Legend: red: Free/Open Source tools; green: Commercial tools; Fuchsia: Hadoop/Big Data tools	% users in 2016	% users in 2015	% users in 2014
R (1419)	49.0%	46.9%	38.5%
Python (1325)	45.8%	30.3%	19.5%
SQL (1029)	35.5%	30.9%	25.3%
Excel (972)	33.6%	22.9%	25.8%
RapidMiner (944), 11.7 % alone	32.6%	31.5%	44.2%
Hadoop (641)	22.1%	18.4%	12.7%
Spark (624)	21.6%	11.3%	2.6%
Tableau (536)	18.5%	12.4%	9.1%
KNIME (521)	18.0%	20.0%	15.0%
scikit-learn (497)	17.2%	8.3%	na
Java (487)	16.8%		

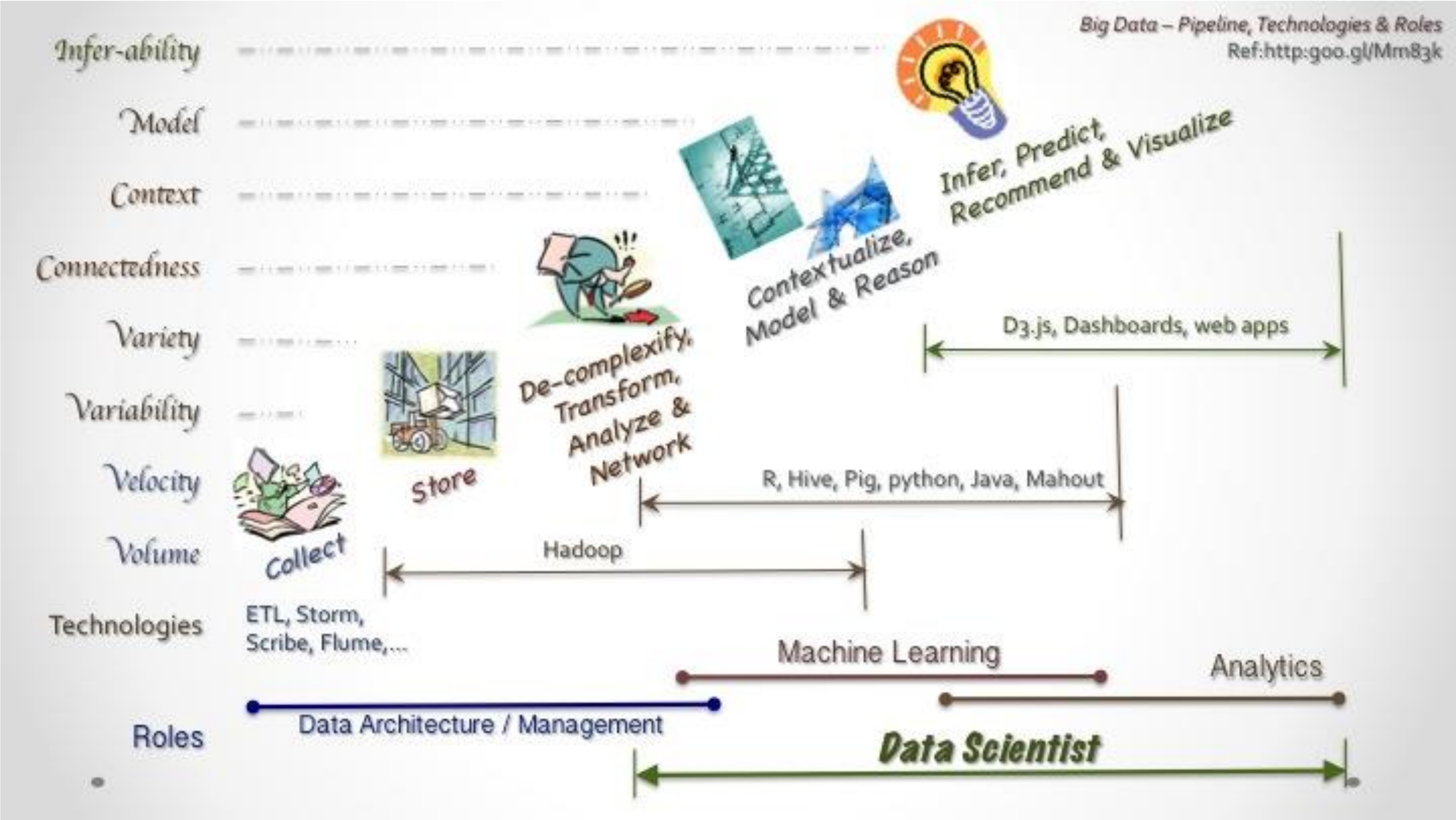


KDnuggets Analytics/Data Science 2016 Software Poll

Python has almost caught up with R in Market Share and yet it is growing 10X faster than R

Data Scientist: Technologies and Roles

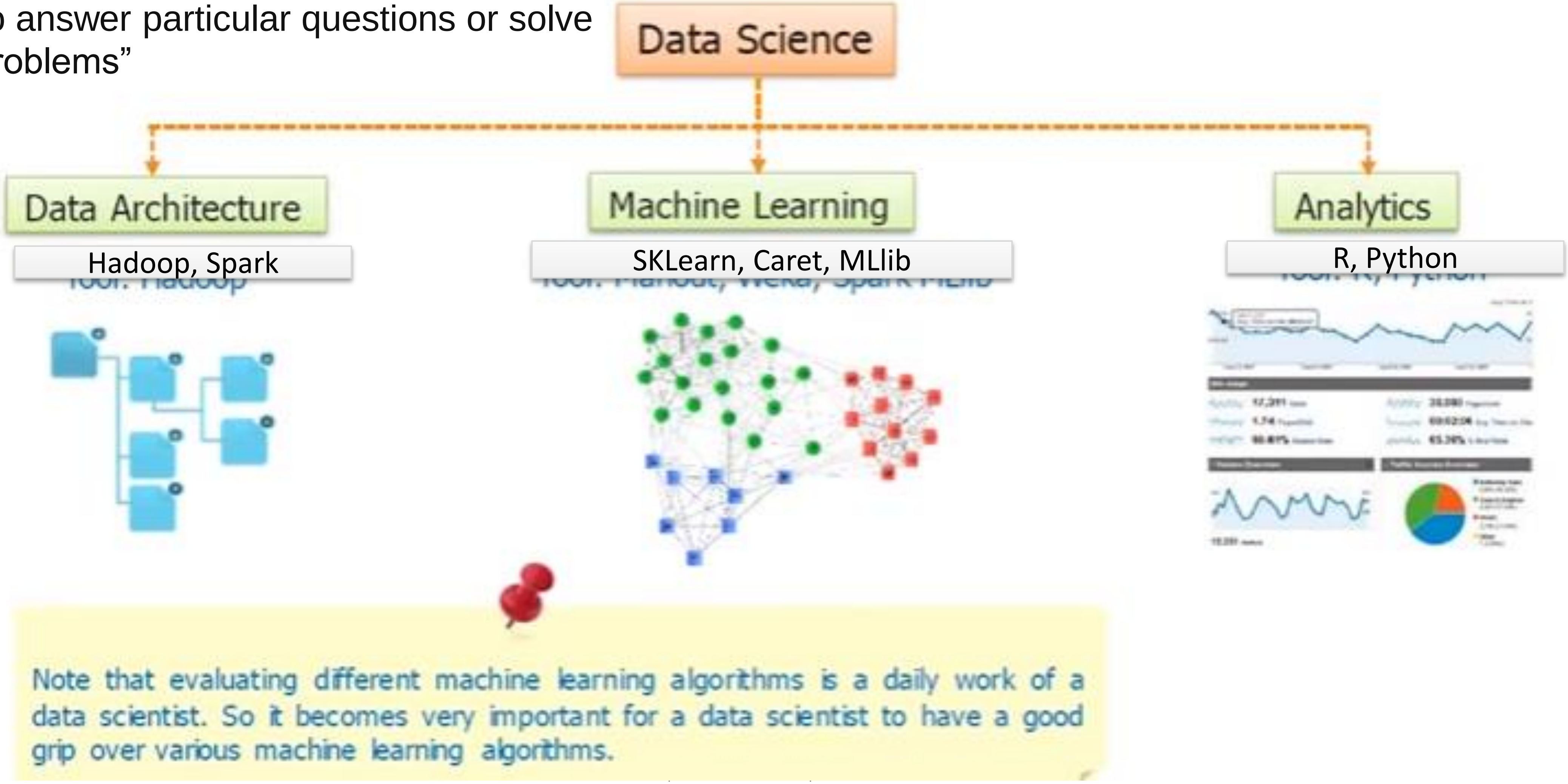
How to prepare for a career in data science



Why Python or R for Data Science?

Algorithms + Data + Insights = Data Science

Data Science is “about the extraction of knowledge from data to answer particular questions or solve particular problems”



Python Basics

Please Open Your iPython Notebooks to Follow Along with this Demo

Type Examples

None None :
- Singleton null object

Boolean:
- True, False

Integer: -1, 0, 1
- Int32 and int64: numpy length ints

Float: 3.14159265
- inf, float('inf'): infinity
- -inf: negative infinity
- nan, float('nan'):Not a number

Complex:
- 2+3j (note use of j)

String:
- 'I am a string', "me too"
- '''multi-line string''', """+1"""
- r'raw string', b'ASCII string'
- u'unicode string'

Continued

Tuple:
- empty = (), (dog,)

Immutable list:
- (1, True, 'dog')

List:
- empty = [], [‘dog’,]
- Mutable list: [1, True, 'dog']

Set:
- empty set=()
- Mutable set: (1, True, 'a')

Dictionary (mutable object)
- empty = {}
- Dictionary: {'a': 'dog', 7: 'seven’, True: 1}

File object:
- f = open('filename', 'rb')

Python Basics

Please Open Your iPython Notebooks to Follow Along with this Demo

Operator Functionality

+ Addition
- (also string, tuple, list concatenation)

-Subtraction
- (also set difference)

* Multiplication (also string, tuple, list replication)

/ Division

% Modulus (also a string format function, but use deprecated)

// Integer division rounded towards minus infinity

Continued

** Exponentiation

Assignment
- =, -=, +=, /=,
- *=, %=, //=,
- **=

Equality
- ==, !=, <, <=,
- >=, >

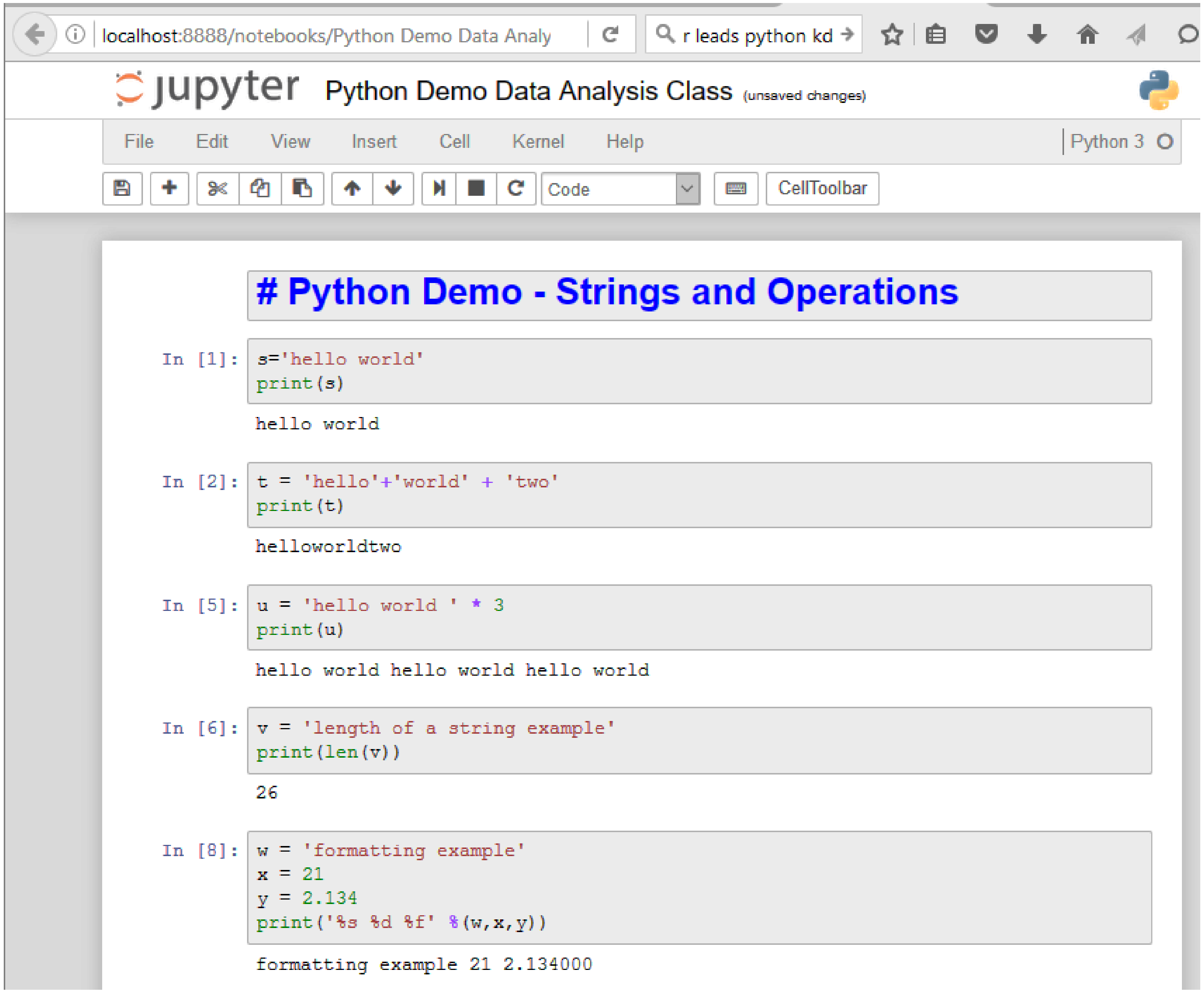
Boolean comparisons
- and, or, not Boolean operators
- in, not in Membership test operators

Boolean continued
- is, is not Object identity operators
- |, ^, &, ~ Bitwise: or, xor, and, compliment
- <<, >> Left and right bit shift

; Inline statement separator
- # inline statements discouraged

Strings

Strings and String Operations



Lists

Lists and List Operations

Python Demo: Lists and List Operations

In [9]:
```
lst = ['this', 'list', 'is', 'interesting',12,24,36]
print(lst)
```
```
['this', 'list', 'is', 'interesting', 12, 24, 36]
```

In [34]:
```
lst2=list(range(5))
lst2
```
Out[34]:
```
[0, 1, 2, 3, 4]
```

In [20]:
```
lst[1]
```
Out[20]:
```
'list'
```

In [21]:
```
lst[1]='bist'
print(lst)
```
```
['this', 'bist', 'is', 'interesting', 12, 24, 36]
```

In [28]:
```
lst[-1]
```
Out[28]:
```
36
```

In [31]:
```
lst[:-1]
```
Out[31]:
```
['this', 'bist', 'is', 'interesting', 12, 24]
```

In [32]:
```
lst[::2]
```
Out[32]:
```
['this', 'is', 12, 36]
```

In [35]:
```
lst = lst+lst2
lst
```
Out[35]:
```
['this', 'bist', 'is', 'interesting', 12, 24, 36, 0, 1, 2, 3, 4]
```

In [37]:
```
lst[1:5]=['modified','list','is','interesting']
lst
```
Out[37]:
```
['this', 'modified', 'list', 'is', 'interesting', 24, 36, 0, 1, 2, 3, 4]
```

In [40]:
```
del lst[5:]
lst
```
Out[40]:
```
['this', 'modified', 'list', 'is', 'interesting']
```

List Methods

Methods of Lists – this is where OOP comes in

Methods on Lists

```
In [41]: lst.append('again') #Add the element x to the end of the list
         lst

Out[41]: ['this', 'modified', 'list', 'is', 'interesting', 'again']

In [43]: n = lst.count('is') #Count the number of times something occurs in the list
         n

Out[43]: 1

In [44]: m = lst.index('is') #Return the index of the first occurrence of x in the list.
         m

Out[44]: 3

In [45]: lst.remove('is') #Delete the first occurrence of x from the list.
         lst

Out[45]: ['this', 'modified', 'list', 'interesting', 'again']

In [46]: lst.reverse() #Reverse the order of elements in the list
         lst

Out[46]: ['again', 'interesting', 'list', 'modified', 'this']

In [47]: lst.sort() # By default, sort the elements in ascending order.
         lst

Out[47]: ['again', 'interesting', 'list', 'modified', 'this']
```

Dictionaries

Dictionaries and Methods

Dictionaries and Methods on Dictionaries

```
In [48]: dict1 = {'first': 'James', 'middle':'Clerk','Age':21}
         dict1

Out[48]: {'Age': 21, 'first': 'James', 'middle': 'Clerk'}

In [50]: dict1.update({'first': 'James', 'born': 1995, 'last': 'Maxwell'})
         dict1

Out[50]: {'Age': 21,
          'born': 1995,
          'first': 'James',
          'last': 'Maxwell',
          'middle': 'Clerk'}

In [51]: dict1.keys( ) #Return a list of all the keys in the dictionary.

Out[51]: dict_keys(['middle', 'Age', 'born', 'first', 'last'])

In [52]: dict1.values( ) #Return a list of all the values in the dictionary.

Out[52]: dict_values(['Clerk', 21, 1995, 'James', 'Maxwell'])

In [53]: dict1.items( ) #Return a list of all the key/value pairs in the dictionary.

Out[53]: dict_items([('middle', 'Clerk'), ('Age', 21), ('born', 1995), ('first', 'Jam
         es'), ('last', 'Maxwell')])

In [55]: 'Age' in dict1 #Test whether the dictionary contains the key 'Age'.

Out[55]: True

In [58]: 'James' in dict1.values() ##Test whether the dictionary contains value 'James'

Out[58]: True
```

Conditions and Loops

Conditional, For and While Loops

Conditional Statements

if :

elif :

else:

In [8]:
```
word = 'Age'
if word in dict1:
    print('value is: ', dict1[word])
else:
    print('that key does not exist')
```

```
value is:  21
```

For Loops

In [9]:
```
for key,value in dict1.items():
    print(key,value)
```

```
Age 21
middle Clerk
last Maxwell
first James
born 1995
```

While Loops

In [11]:
```
lst = ['this','is','the','while','loop']
while lst:
    print(lst)
    lst = lst[1:]
```

```
['this', 'is', 'the', 'while', 'loop']
['is', 'the', 'while', 'loop']
['the', 'while', 'loop']
['while', 'loop']
['loop']
```

Functions

Functions: Map, Reduce, Filter, Lambda

Functions: Map, Reduce, Filter, Lambda

In [2]:
```
def square(a):
    return a*a
def add(a,b):
    return a+b
print(square(12),add(3,2))
```
```
144 5
```

In [3]:
```
s = map(square,[3,6,9])
for i in s:
    print(i)
```
```
9
36
81
```

In [4]:
```
from functools import reduce
t = reduce(add,[3,5,7,8])
t
```
Out[4]:
```
23
```

In [1]:
```
def f(x): return x % 2 !=0
y = filter(f, [5,4,7,9,8,10])
for i in y:
    print(i)
```
```
5
7
9
```

In [6]:
```
m = map(lambda x: x**2, range(5))
for i in m:
    print(i)
```
```
0
1
4
9
16
```

Project

Wordcount Example

Wordcount Example

In [40]:
```python
def wordcount(textfile):
    results = []
    with open(textfile,'r') as f:
    # read lines and discard header
        lines = f.readlines()
        print(lines)
        line = lines[0].split()
        print(line)
        for w in line:
            results.append([w,(line.count(w))])
#         for u in res:
#             print(u)
    f.close()
    return results
```

In [41]:
```python
wc = wordcount('wordcount.txt')
wc
```

```
['R remains the leading tool, with leading share, but Python grows faster an
d almost catches up to R. RapidMiner remains the most popular general Data S
cience platform. Big Data tools used by almost a majority, and Deep Learning
 usage doubles. ']
['R', 'remains', 'the', 'leading', 'tool,', 'with', 'leading', 'share,', 'bu
t', 'Python', 'grows', 'faster', 'and', 'almost', 'catches', 'up', 'to', 'R.
', 'RapidMiner', 'remains', 'the', 'most', 'popular', 'general', 'Data', 'Sc
ience', 'platform.', 'Big', 'Data', 'tools', 'used', 'by', 'almost', 'a', 'm
ajority,', 'and', 'Deep', 'Learning', 'usage', 'doubles.']
```

Out[41]:
```
[['R', 1],
 ['remains', 2],
 ['the', 2],
 ['leading', 2],
 ['tool,', 1],
 ['with', 1],
 ['leading', 2],
 ['share,', 1],
 ['but', 1],
 ['Python', 1],
 ['grows', 1],
 ['faster', 1],
 ['and', 2],
 ['almost', 2],
 ['catches', 1],
 ['up', 1],
```

Pitfalls in Python

Sending Lists as Arguments

What happened to a?

Pitfalls of Sending Function Arguments

```
In [32]: def sum(lst):
             tot=0
             for i in range(0,len(lst)):
                 lst[i]+=1
                 tot += lst[i]
             return tot
         a=list(range(1,4))
         a

Out[32]: [1, 2, 3]

In [30]: sum(a)

Out[30]: 9

In [31]: a

Out[31]: [2, 3, 4]

In [35]: a_copy=a[:]
         a_copy

Out[35]: [1, 2, 3]

In [36]: sum(a_copy)

Out[36]: 9

In [34]: a

Out[34]: [1, 2, 3]
```

Overcoming Pitfalls

Always do deepcopy when sending lists as arguments to a function

 # Always perform deepcopy when copying Lists

```
In [37]: a=[1,2,3,[4,5]]
         b=a[:]
         a[0]=2
         b
Out[37]: [1, 2, 3, [4, 5]]

In [38]: a[3][0]=0
         b
Out[38]: [1, 2, 3, [0, 5]]

In [39]: # Can be fixed by
         import copy
         a=[1,2,3,[4,5]]
         b = copy.deepcopy(a)
         a[3][0]=0
         b
Out[39]: [1, 2, 3, [4, 5]]

In [ ]:
```

What happened to b?

Data Analysis using Python

Numpy and Pandas are the two libraries most used for Data Analysis in Python

numpy Arrays and Operations

```
In [42]: import numpy as np

In [51]: arra = np.array([1, 2, 3,4])
         arra

Out[51]: array([1, 2, 3, 4])

In [52]: arrb = np.array([[1, 2, 3], [4, 5, 6]])
         arrb

Out[52]: array([[1, 2, 3],
                [4, 5, 6]])

In [53]: arrc=np.arange(3, 10, 2)
         arrc

Out[53]: array([3, 5, 7, 9])

In [54]: arra+arrc

Out[54]: array([ 4,  7, 10, 13])

In [55]: arra[2]

Out[55]: 3
```

Pandas

Pandas builds on top of numpy to provide a powerful feature: DataFrames

```
# Pandas provides DataFrames and more

In [56]: import pandas as pd

In [59]: sera = pd.Series([1, 2, 3], index=['a', 'b', 'c'])
         sera

Out[59]: a    1
         b    2
         c    3
         dtype: int64

In [60]: serb = pd.Series({'a': 1, 'b': 2, 'd': 4}, index=['a', 'b', 'c'])
         serb

Out[60]: a    1.0
         b    2.0
         c    NaN
         dtype: float64

In [63]: dfa = pd.DataFrame({'a': [1, 2], 'b': [3, 4]}, columns=['a','b','c'],
                            index=['top', 'bottom'])
         dfa

Out[63]:
```

	a	b	c
top	1	3	NaN
bottom	2	4	NaN

```
In [65]: df = pd.read_csv('train.csv',sep='\t')
         df.head()

Out[65]:
```

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950

Data Analysis using Python

DataFrames represent the fastest means to store, analyze and transform data

Data Analysis using Pandas DataFrames

In [66]:
```
df.describe().T
```

Out[66]:

	count	mean	std	min	25%	50%	75%	max
Item_Weight	19.0	13.770789	4.215969	5.9200	9.847500	13.650000	17.55000	19.20000
Item_Visibility	22.0	0.046427	0.043475	0.0000	0.013568	0.034819	0.06917	0.13819
Item_MRP	22.0	126.992082	67.075309	45.5402	56.685750	117.513700	172.94090	250.87240
Outlet_Establishment_Year	22.0	1996.681818	8.328873	1985.0000	1987.000000	1998.500000	2001.25000	2009.00000
Item_Outlet_Sales	22.0	2041.221745	1303.020888	343.5528	1015.178550	1799.657400	2637.23380	4710.53500

In [67]:
```
df.isnull().sum()
```

Out[67]:
```
Item_Identifier              0
Item_Weight                  3
Item_Fat_Content             0
Item_Visibility              0
Item_Type                    0
Item_MRP                     0
Outlet_Identifier            0
Outlet_Establishment_Year    0
Outlet_Size                  3
Outlet_Location_Type         0
Outlet_Type                  0
Item_Outlet_Sales            0
dtype: int64
```

In [68]:
```
df['Item_Type'].unique()
```

Out[68]:
```
array(['Dairy', 'Soft Drinks', 'Meat', 'Fruits and Vegetables',
       'Household', 'Baking Goods', 'Snack Foods', 'Frozen Foods',
       'Breakfast', 'Health and Hygiene', 'Hard Drinks'], dtype=object)
```

In [70]:
```
df.corr()
```

Out[70]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Weight	1.000000	0.333284	0.549445	-0.229564	0.432069
Item_Visibility	0.333284	1.000000	0.448929	-0.229607	0.579775
Item_MRP	0.549445	0.448929	1.000000	-0.091522	0.659095
Outlet_Establishment_Year	-0.229564	-0.229607	-0.091522	1.000000	-0.254797
Item_Outlet_Sales	0.432069	0.579775	0.659095	-0.254797	1.000000

Data Analysis using Python

You can use DataFrames like RDBMS Tables or Excel Sheets – they are versatile and powerful!

DataFrames can be used like Excel Tables or SQL Queries

In [72]:
```
df.groupby('Item_Fat_Content').sum()
```

Out[72]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Fat_Content					
Low Fat	89.030	0.373627	1254.2014	17954	21017.3086
Regular	172.615	0.647766	1539.6244	25973	23889.5698

In [74]:
```
pd.pivot_table(df,values='Item_Outlet_Sales',index='Item_Fat_Content',columns='Outlet_Location_Type')
```

Out[74]:

Outlet_Location_Type	Tier 1	Tier 2	Tier 3
Item_Fat_Content			
Low Fat	2449.47820	2748.4224	2184.09032
Regular	1637.46852	2893.5668	1652.51560

In [75]:
```
pd.pivot_table(df,values='Item_Outlet_Sales',index='Item_Fat_Content',columns='Outlet_Location_Type',aggfunc=len)
```

Out[75]:

Outlet_Location_Type	Tier 1	Tier 2	Tier 3
Item_Fat_Content			
Low Fat	3.0	1.0	5.0
Regular	5.0	2.0	6.0

In [76]:
```
pd.crosstab(df.Item_Fat_Content,df.Outlet_Location_Type)
```

Out[76]:

Outlet_Location_Type	Tier 1	Tier 2	Tier 3
Item_Fat_Content			
Low Fat	3	1	5
Regular	5	2	6

Data Analysis using Python

You can add, drop and transform new columns on the fly

You can drop columns, fill null values, add columns on the fly

In [78]:
```
df['Age of Outlet']=2016-df['Outlet_Establishment_Year']
```

In [81]:
```
df.head()
```

Out[81]:

em_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Age of Outlet
airy	249.8092	OUT049	1999	Medium	Tier 1	Supermarket Type1	3735.1380	17
oft Drinks	48.2692	OUT018	2009	Medium	Tier 3	Supermarket Type2	443.4228	7
leat	141.6180	OUT049	1999	Medium	Tier 1	Supermarket Type1	2097.2700	17
ruits and egetables	182.0950	OUT010	1998	NaN	Tier 3	Grocery Store	732.3800	18
ousehold	53.8614	OUT013	1987	High	Tier 3	Supermarket Type1	994.7052	29

In [82]:
```
df.drop(['Outlet_Establishment_Year'],axis=1,inplace=True)
```

In [83]:
```
df.head()
```

Out[83]:

at_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Age of Outlet
t	0.016047	Dairy	249.8092	OUT049	Medium	Tier 1	Supermarket Type1	3735.1380	17
-	0.019278	Soft Drinks	48.2692	OUT018	Medium	Tier 3	Supermarket Type2	443.4228	7
t	0.016760	Meat	141.6180	OUT049	Medium	Tier 1	Supermarket Type1	2097.2700	17
-	0.000000	Fruits and Vegetables	182.0950	OUT010	NaN	Tier 3	Grocery Store	732.3800	18
t	0.000000	Household	53.8614	OUT013	High	Tier 3	Supermarket Type1	994.7052	29

Pandas Operations

You can select, drop and fill columns

Fill Null Values and Get Ready for Data Visualization

In [99]:
```
from statistics import mode
outlet_size_mode=df['Outlet_Size'].mode()
outlet_size_mode[0]
```

Out[99]: 'Medium'

In [100]:
```
df['Outlet_Size'].fillna(outlet_size_mode[0],inplace=True)
df.head()
```

Out[100]:

at_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Age of Outlet
	0.016047	Dairy	249.8092	OUT049	Medium	Tier 1	Supermarket Type1	3735.1380	17
	0.019278	Soft Drinks	48.2692	OUT018	Medium	Tier 3	Supermarket Type2	443.4228	7
	0.016760	Meat	141.6180	OUT049	Medium	Tier 1	Supermarket Type1	2097.2700	17
	0.000000	Fruits and Vegetables	182.0950	OUT010	Medium	Tier 3	Grocery Store	732.3800	18
	0.000000	Household	53.8614	OUT013	High	Tier 3	Supermarket Type1	994.7052	29

In [102]:
```
df[df['Outlet_Size'].map(lambda Outlet_Size: 'High' in Outlet_Size)]
```

Out[102]:

at_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Size	Outlet_Location_Type	Outlet_Type	Item_Outlet_Sales	Age of Outlet
	0.000000	Household	53.8614	OUT013	High	Tier 3	Supermarket Type1	994.7052	29
	0.012741	Snack Foods	57.6588	OUT013	High	Tier 3	Supermarket Type1	343.5528	29
	0.068024	Fruits and Vegetables	196.4426	OUT013	High	Tier 3	Supermarket Type1	1977.4260	29
	0.138190	Snack Foods	250.8724	OUT013	High	Tier 3	Supermarket Type1	3775.0860	29

Visualization in Python

Python has powerful visualization libraries.

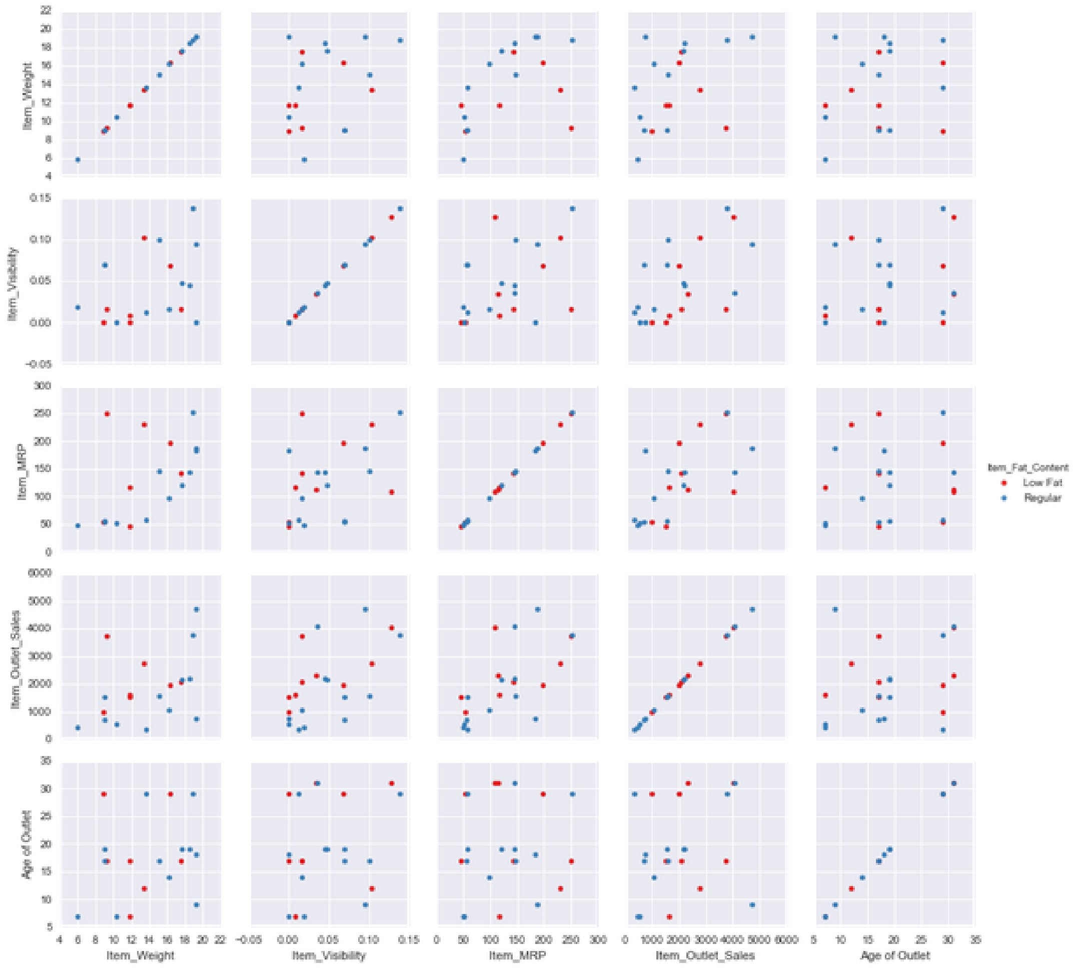
One of them is Seaborn

A PairGrid takes only one variable 'hue' and plots every other variable in the dataframe against it

Python has Powerful Visualization Libraries

In [108]:
```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
```

In [111]:
```
g = sns.PairGrid(df,hue='Item_Fat_Content',palette='Set1')
g=g.map(plt.scatter)
g = g.add_legend()
```



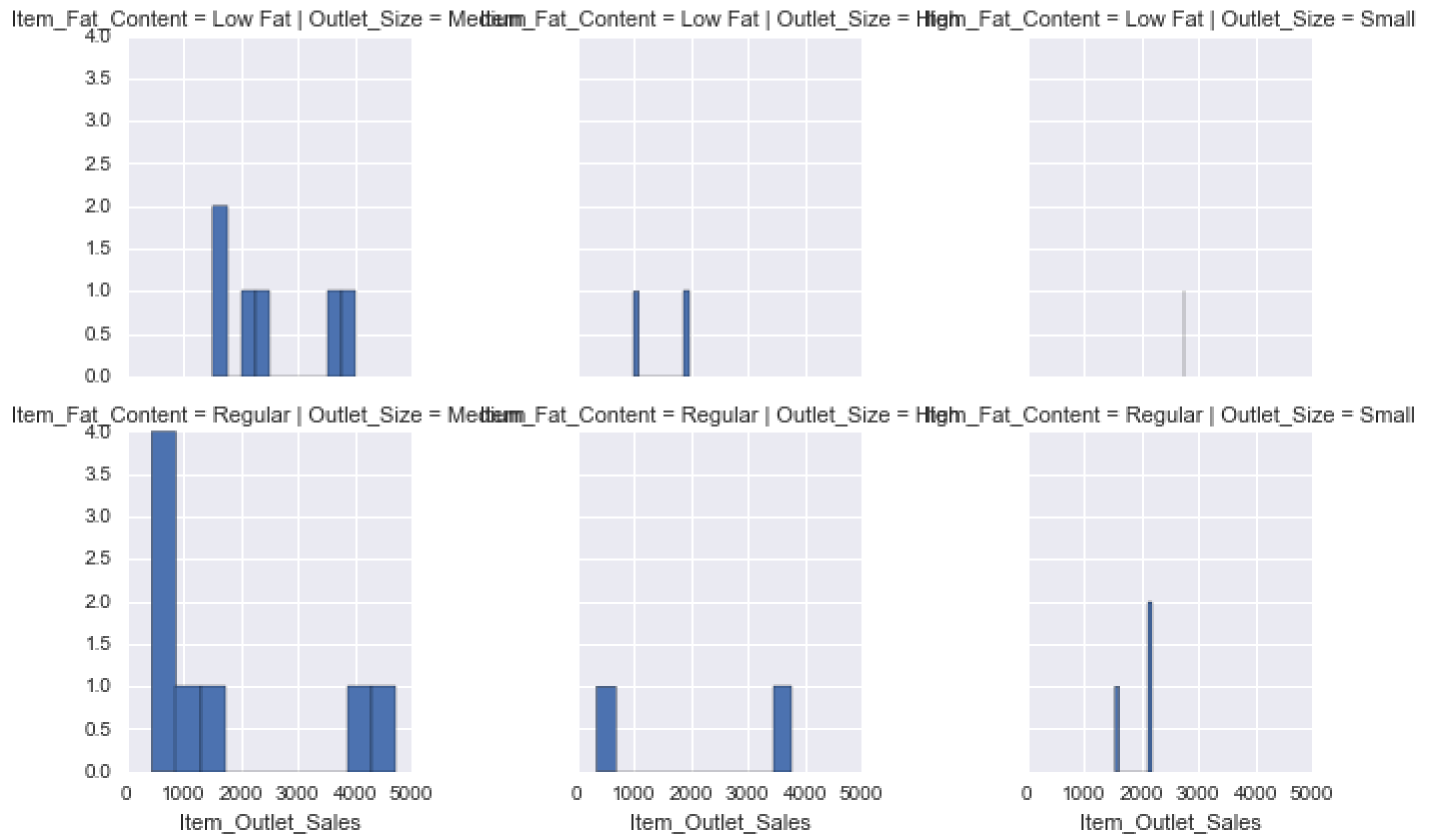
Visualization

Python has powerful visualization libraries.

A FacetGrid plots a continuous var against two facets (categorical vars)

Seaborn and PyPlot create numerous graphs

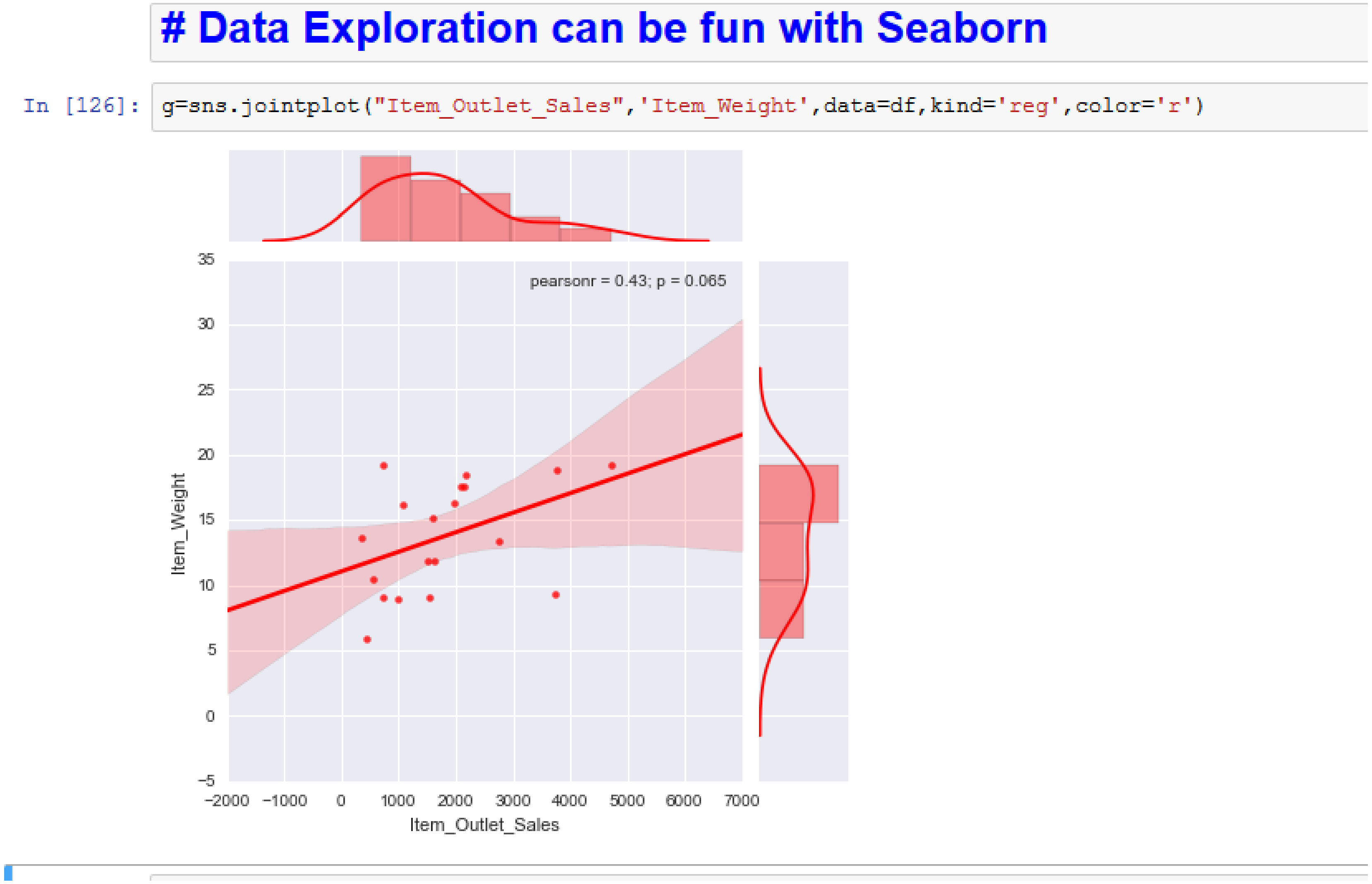
In [120]:
```
g = sns.FacetGrid(df,row='Item_Fat_Content',col='Outlet_Size',
                  palette='Set1')
g = g.map(plt.hist,'Item_Outlet_Sales')
g = g.add_legend()
```



Visualization

Python has powerful visualization libraries.

A jointplot plots two continuous vars against each other



Statistical Modeling

Python has powerful Modeling libraries

Statsmodels is one such library for Regression and other Modeling techniques

Python has lots of Statistical Modeling Libraries

In [127]:
```
import pandas
import statsmodels.api as sm
import numpy as np
```

In [128]:
```
url = "http://www.ats.ucla.edu/stat/mult_pkg/faq/general/lgtrans.csv"
data = pandas.read_csv(url)
print(data.female.unique())
```

```
['male' 'female']
```

In [129]:
```
data['female'] = data.female.replace(dict(male=0, female=1))
data[['math', 'read']] = np.log(data[['math', 'read']])
data['const'] = 1 # sm.add_constant(data)
y_name = 'write'
x_name = ['const', 'female', 'math', 'read']
test_scores = sm.OLS(data[y_name], data[x_name]).fit()
print(test_scores.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  write   R-squared:                       0.530
Model:                            OLS   Adj. R-squared:                  0.523
Method:                 Least Squares   F-statistic:                     73.70
Date:                Tue, 28 Jun 2016   Prob (F-statistic):           5.92e-32
Time:                        22:34:57   Log-Likelihood:                -657.58
No. Observations:                 200   AIC:                             1323.
Df Residuals:                     196   BIC:                             1336.
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
const        -99.1640     10.804     -9.178      0.000      -120.471   -77.857
female         5.3888      0.931      5.789      0.000         3.553     7.224
math          20.9410      3.431      6.104      0.000        14.175    27.707
read          16.8522      3.063      5.501      0.000        10.811    22.894
==============================================================================
Omnibus:                        2.259   Durbin-Watson:                   1.999
Prob(Omnibus):                  0.323   Jarque-Bera (JB):                2.286
Skew:                          -0.253   Prob(JB):                        0.319
Kurtosis:                       2.866   Cond. No.                         135.
==============================================================================
```

Predictive Analytics

Python has powerful Modeling libraries

Predictive Modeling is easy with such libraries

SKLearn is another powerful library focused on Machine Learning

```
# Predictive Analytics is easy with Python

In [130]: X = [1, 1, np.log(65), np.log(70)]
          print(test_scores.predict(X))

          [ 65.2368995]

In [ ]:
```

Closure

What we have learned so far

Python:
1. Basics and Types
2. Strings and Operations
3. Lists and Operations
4. Dictionaries and Methods
5. Conditionals and Loops
6. Functions, Map and Reduce
7. Libraries
8. Pitfalls

Data Analysis:
1. Numpy Arrays
2. Pandas DataFrames
3. Columns and Values
4. Data Access Methods
5. Data Transformations
6. Data Visualizations
7. Model Building
8. Predictions

Thank You and please fill out your feedback forms

Ramadurai.Seshadri@wipro.com