Data Science Made Easy – Level 2

Using python™

Ramadurai Seshadri
August 2016
Ramadurai.seshadri@wipro.com

What You will Learn Today

Agenda for “Data Science Made Easy Using Python – Level 2”

1. What is Data Science and why is it important now?
2. What are the tools used by a Data Scientist for perform predictive analytics
3. What are the steps involved in building a predictive model
    - ➢Hypothesis Generation
    - ➢Data Ingestion
    - ➢Data Preparation
    - ➢Data Visualization
    - ➢Feature Engineering
    - ➢Model Preprocessing
4. How does a Data Scientist use Python
    - ✓ Using libraries - Pandas, Numpy, Matplotlib, Scipy, Statsmodels, Sci-Kit Learn
    - ✓ Machine Learning Algorithms

© 2016 WIPRO LTD | WWW.WIPRO.COM | CONFIDENTIAL

Data Science – Why Now?

What is Data Science and Why is it important now?

Data Science is an interdisciplinary field to extract knowledge from data

Google's Self Driving Cars and Robots get a lot of press, but Google's real future is in Machine Learning: the technology that enables computers to get smarter and more personal.
- Eric Schmidt (Google Chairman)

My personal feeling can be summed up as follows:

We are probably living in the most defining exciting period in the Computer Revolution. What makes this period exciting is the democratization of tools and techniques that enable us to use "smart" machines and make them "smarter".

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

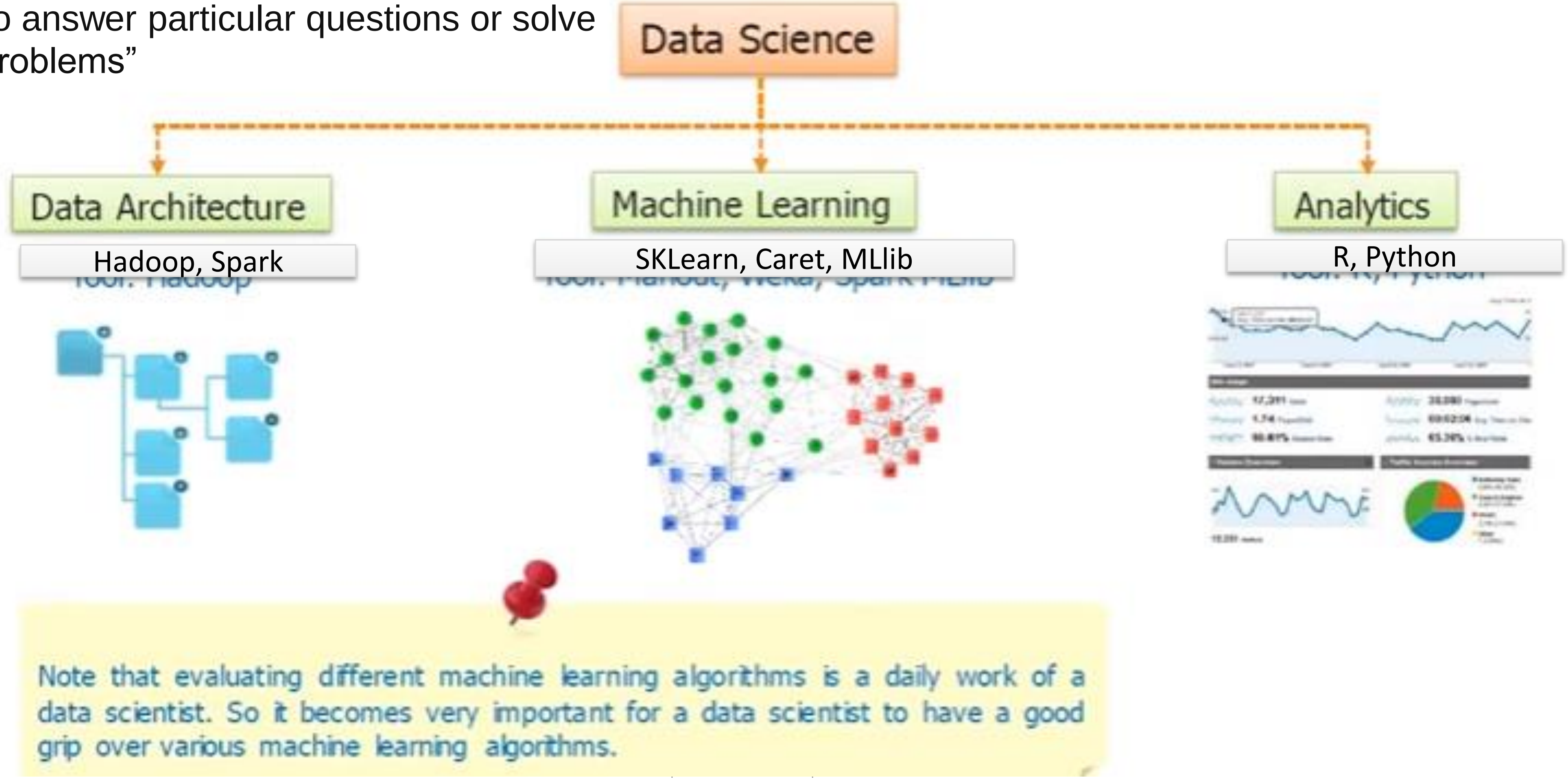
A person with such all-round skills can be called a Data Scientist

© 20 ... | CONFIDENTIAL

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"



Problem Definition and Hypotheses Generation

What is the business problem I am trying to solve?

1. Problem Statement example

   Predicting Revenues for a Chain Store. A Chain Store has revenue data for 2013 across 10 different stores and 1559 categories of Items. Their problem is they want to forecast revenues by store and by item for the next year. Can we design a prediction algorithm to help the Store?

2. What is the Outcome I am trying to achieve or predict?

   - Outcome variable example

   Your task is to predict the Item Outlet Sales for the next year across Outlets and Items.

3. What kind of data do I have?

   - Store and Item Sales data*

Data

We have train (8523) and test (5681) data set, train data set has both input and output variable(s). You need to predict the sales for test data set.

Variable	Description
Item_Identifier	Unique product ID
Item_Weight	Weight of product
Item_Fat_Content	Whether the product is low fat or not
Item_Visibility	The % of total display area of all products in a store allocated to the particular product
Item_Type	The category to which the product belongs
Item_MRP	Maximum Retail Price (list price) of the product
Outlet_Identifier	Unique store ID
Outlet_Establishment_Year	The year in which store was established
Outlet_Size	The size of the store in terms of ground area covered
Outlet_Location_Type	The type of city in which the store is located
Outlet_Type	Whether the outlet is just a grocery store or some sort of supermarket
Item_Outlet_Sales	Sales of the product in the particulat store. This is the outcome variable to be predicted.

* source: AnayticsVidhya.com

Before you begin your Data Science project

You must understand a few things:

1. What is the business problem I am trying to solve?
   - Problem Statement
2. What is the Outcome I am trying to achieve or predict?

   - Outcome variable
3. What kind of data do I have? Does it have all the information that I want captured to do a thorough analysis? If not, gather more data or wait.

   - Data Gathering
4. Once I have the data, what do I know about the business problem that:
   a) I can generate possible hypotheses that can predict the outcome

      - Hypothesis Generation

   b) I can explore possible features that will impact the outcome

      - Feature Engineering

   c) I can list one or more methods can I use to solve the problem

      - Modeling

   d) I can evaluate whether the methods are appropriate and working

      - Validation

© 2016 WIPRO LTD | WWW.WIPRO.COM | CONFIDENTIAL

Problem Definition and Hypotheses Generation

Once I have the data, what do I know about the business problem that:

a) I can generate possible hypotheses that can predict the outcome
   - Hypothesis Generation example

   MRP of an item is an important factor when customers purchase an item. Similarly Location and Age of a store determines its Sales.

b) I can explore possible features that will impact the outcome
   - Feature Engineering example

   Age of a Store and Average MRP may be factors in purchasing Items. However, Item ID and Outlet ID is not.

c) I can list one or more methods can I use to solve the problem
   - Modeling

   Since the prediction is of the Numeric type, we can use a Random Forest Regressor

d) I can evaluate whether the methods are appropriate and working

   Use RMSE Score

Data Load & Preparation

Process to prepare data to be ready for Visualization using Pandas

Load Data from CSV format into a Python Pandas DataFrame

We have train (100) and test (50) data sets. Hence we must load both into separate Pandas Data Frames.

```
train = pd.read_csv('train2.csv',sep=',')
train.shape
test = pd.read_csv('test2.csv',sep=',')
test.shape
train['source']='train'
test['source']='test'
df = pd.concat([train,test])
df.shape
```

Pandas provides DataFrames and more

In [7]:
```
import pandas as pd
```

In [8]:
```
train = pd.read_csv('train2.csv',sep=',')
train.shape
```

Out[8]: (100, 12)

In [9]:
```
test = pd.read_csv('test2.csv',sep=',')
test.shape
```

Out[9]: (50, 11)

In [11]:
```
train['source']='train'
test['source']='test'
df = pd.concat([train,test])
df.shape
```

Out[11]: (150, 13)

In [41]:
```
df.head()
```

Out[41]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Type	Item_Visibility	Item_Weight	Outlet
0	Low Fat	FDA15	249.8092	3735.1380	Dairy	0.016047	9.30	OUT04
1	Regular	DRC01	48.2692	443.4228	Soft Drinks	0.019278	5.92	OUT01
2	Low Fat	FDN15	141.6180	2097.2700	Meat	0.016760	17.50	OUT04
3	Regular	FDX07	182.0950	732.3800	Fruits and Vegetables	0.000000	19.20	OUT01
4	Low Fat	NCD19	53.8614	994.7052	Household	0.000000	8.93	OUT01

1. Describe Data using Summary Stats

In [40]:
```
df.dtypes
```
Out[40]:
```
Item_Fat_Content          object
Item_Identifier           object
Item_MRP                 float64
Item_Outlet_Sales        float64
Item_Type                 object
Item_Visibility          float64
Item_Weight              float64
Outlet_Identifier         object
Outlet_Location_Type      object
Outlet_Size               object
Outlet_Type               object
source                    object
Age of Outlet              int64
dtype: object
```

Data Analysis using Pandas DataFrames

In [12]:
```
df.describe().T
```
Out[12]:

	count	mean	std	min	25%	50%	75%	m
Item_MRP	150.0	140.605500	63.952782	32.0900	95.668600	143.62990	187.847050	2
Item_Outlet_Sales	100.0	2123.395992	1581.167955	125.8362	792.801350	1837.60800	3099.299000	6
Item_Visibility	150.0	0.068658	0.057833	0.0000	0.025743	0.05783	0.101872	0
Item_Weight	121.0	12.818306	4.725583	4.7850	8.750000	12.50000	17.500000	2
Outlet_Establishment_Year	150.0	1997.446667	8.299379	1985.0000	1987.000000	1998.00000	2004.000000	2

Basic Measures:

- Cardinality
- Unique Count
- No. Of Missing values
- % of missing values
- # of Non-Missing values
- % of Non Missing values
- Min
- Max
- Sum

Central Tendency

Dispersion

Quantiles

1. Assess Data Quality by Calculating Missing Values

Data Analysis using Pandas DataFrames

In [7]:
```
df.describe().T
```

Item Visibility Zero – does it make sense?

Out[7]:

	count	mean	std	min	25%	50%	75%
Item_MRP	150.0	140.605500	63.952782	32.0900	95.668600	143.62990	187.847050
Item_Outlet_Sales	100.0	2123.395992	1581.167955	125.8362	792.801350	1837.60800	3099.299000
Item_Visibility	150.0	0.068658	0.057833	0.0000	0.025743	0.05783	0.101872
Item_Weight	121.0	12.818306	4.725583	4.7850	8.750000	12.50000	17.500000
Outlet_Establishment_Year	150.0	1997.446667	8.299379	1985.0000	1987.000000	1998.00000	2004.000000

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Item_Fat_Content              0
Item_Identifier               0
Item_MRP                      0
Item_Outlet_Sales            50
Item_Type                     0
Item_Visibility               0
Item_Weight                  29
Outlet_Establishment_Year     0
Outlet_Identifier             0
Outlet_Location_Type          0
Outlet_Size                  41
Outlet_Type                   0
source                        0
dtype: int64
```

Missing Data

2. Variable Reduction – get rid of “useless” variables!

Exclude Variables that don’t meet your Data Quality standard

Extreme Observations

Tests for Normality

Tests for Location

Confidence Intervals

```
In [8]:  df.isnull().sum()

Out[8]:  Item_Fat_Content              0
         Item_Identifier               0
         Item_MRP                      0
         Item_Outlet_Sales            50
         Item_Type                     0
         Item_Visibility               0
         Item_Weight                  29
         Outlet_Establishment_Year     0
         Outlet_Identifier             0
         Outlet_Location_Type          0
         Outlet_Size                  41
         Outlet_Type                   0
         source                        0
         dtype: int64
```

Number of Null Values

If % of Missing values exceeds 90% **Reject**

2. Fill or Impute “Missing” Values first before Visualization

Python provides a few simple ways to fill or impute “missing data”

In the first case we use the mean of the column since it is a Continuous variable

In the next case, we use the Mode of the column since it is a Categorical variable

Fill null values first and then perform Data Visualization

```
In [14]: df['Item_Weight'].isnull().sum()

Out[14]: 29

In [15]: mean = df['Item_Weight'].mean()
         mean

Out[15]: 12.818305785123965

In [16]: df['Item_Weight'].fillna(mean, inplace=True)
         df['Item_Weight'].isnull().sum()

Out[16]: 0

In [17]: df['Outlet_Size'].isnull().sum()

Out[17]: 41

In [19]: #Python 3 code for mode:
         from statistics import mode
         outlet_size_mode=mode(df['Outlet_Size'])
         outlet_size_mode= outlet_size_mode
         outlet_size_mode

Out[19]: 'Medium'

In [20]: # Python 2 Code for mode
         #from scipy.stats import mode

         #outlet_size_mode=mode(df['Outlet_Size'])
         #outlet_size_mode= outlet_size_mode[0][0]
         #outlet_size_mode

In [21]: df['Outlet_Size'].fillna(outlet_size_mode,inplace=True)
         df['Outlet_Size'].isnull().sum()

Out[21]: 0
```

© 2016 WI

2. Fill or Impute "Missing" Values (continued)

Python provides a few simple ways to fill or impute "missing data"

Another method is to impute values from the same column or a different column

In [7]:
```
df.head(4)
```
Out[7]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	NaN

In [8]:
```
df['Outlet_Size'].isnull().sum()
```
Out[8]: 2410

Before the operation

In [9]:
```
# Python 2 Code for mode
from scipy.stats import mode
```

In [ ]:
```
outlet_size_mode=mode(df['Outlet_Size'])
outlet_size_mode= outlet_size_mode[0][0]
```

In this case we use the mode of the column since it is a categorical variable (mode is preferred)

In [11]:
```
df['Outlet_Size'].fillna(outlet_size_mode,inplace=True)
df.head()
```
Out[11]:

	Item_Identifier	Item_Weight	Item_Fat_Content	Item_Visibility	Item_Type	Item_MRP	Outlet_Identifier	Outlet_Establishment_Year	Outlet_Size
0	FDA15	9.30	Low Fat	0.016047	Dairy	249.8092	OUT049	1999	Medium
1	DRC01	5.92	Regular	0.019278	Soft Drinks	48.2692	OUT018	2009	Medium
2	FDN15	17.50	Low Fat	0.016760	Meat	141.6180	OUT049	1999	Medium
3	FDX07	19.20	Regular	0.000000	Fruits and Vegetables	182.0950	OUT010	1998	Medium

In [12]:
```
df['Outlet_Size'].isnull().sum()
```
Out[12]: 0

After the operation

© 2016 WIPRO LTD | WWW.WIPRO.COM | CONFIDENTIAL

Data Visualization

Powerful Graphing and Visualizing Capabilities of Matplotlib, Seaborn

3. Visualize Data Relationships

Powerful Python Libraries: Matplotlib and Seaborn

Python has powerful libraries for Visualization.
We are going to explore a couple:

- Matplotlib and
- Seaborn

Matplotlib:

- Provides powerful publication quality 2D plots
- New tools added to augment Matplotlib:
    - Matplot3D, Basemap, Canopy

Seaborn:

- Developed at Stanford
- Statistical visualization built on Matplotlib
- High level interface provided to plot statistical measures
- https://web.stanford.edu/~mwaskom/software/seaborn/

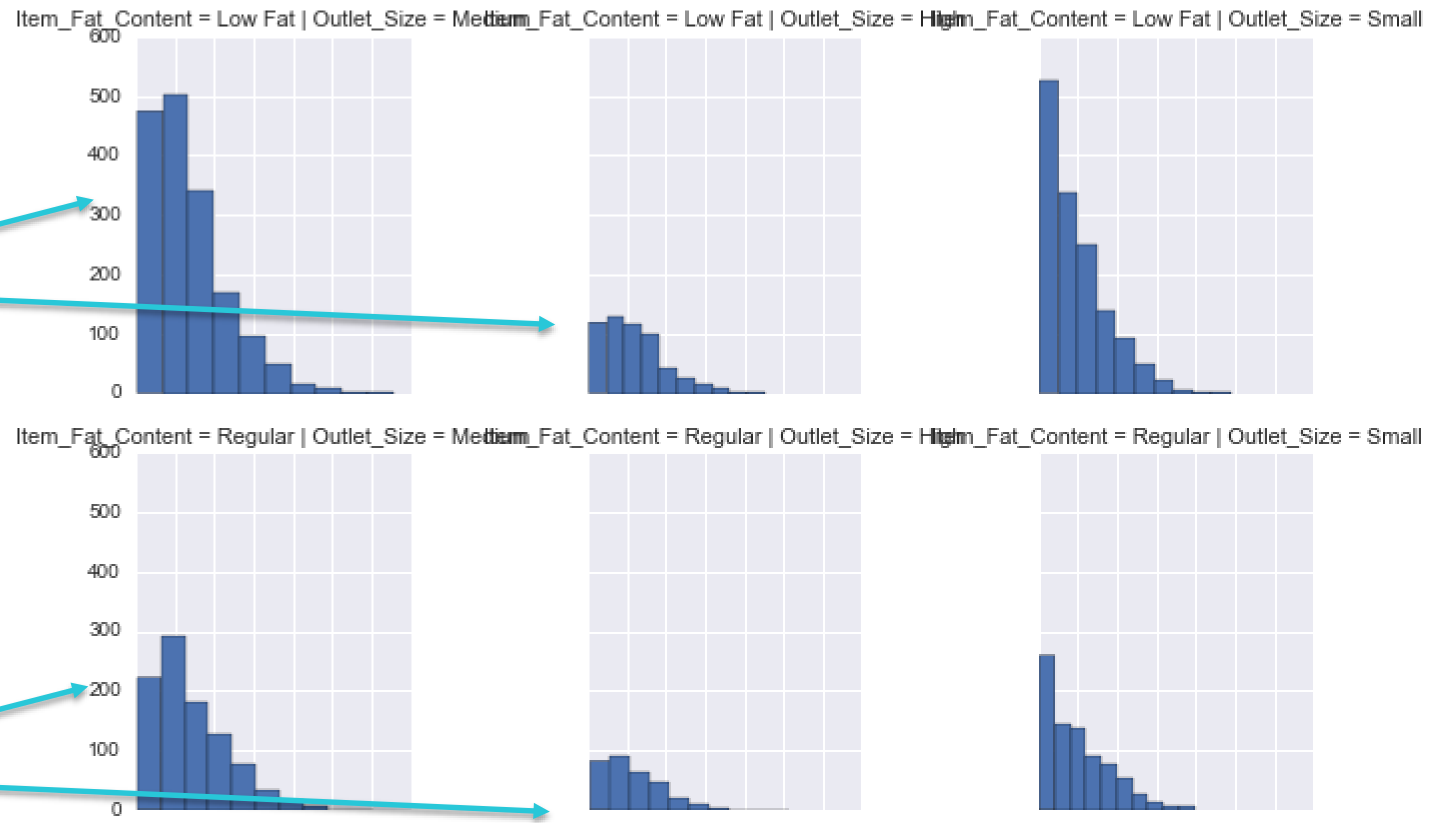
© 2016 WIPRO LTD | WWW.WIPRO.COM | CONFIDENTIAL

3. Visualize Changing Patterns

Visualize the distribution of independent variables under each category of the segmentation variable

The idea is to ask whether the frequency distributions of a given variable shows a change in pattern across the various types of the segmentation variable. This may imply a trend that is of interest in your analysis.

```
In [9]: 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()
```

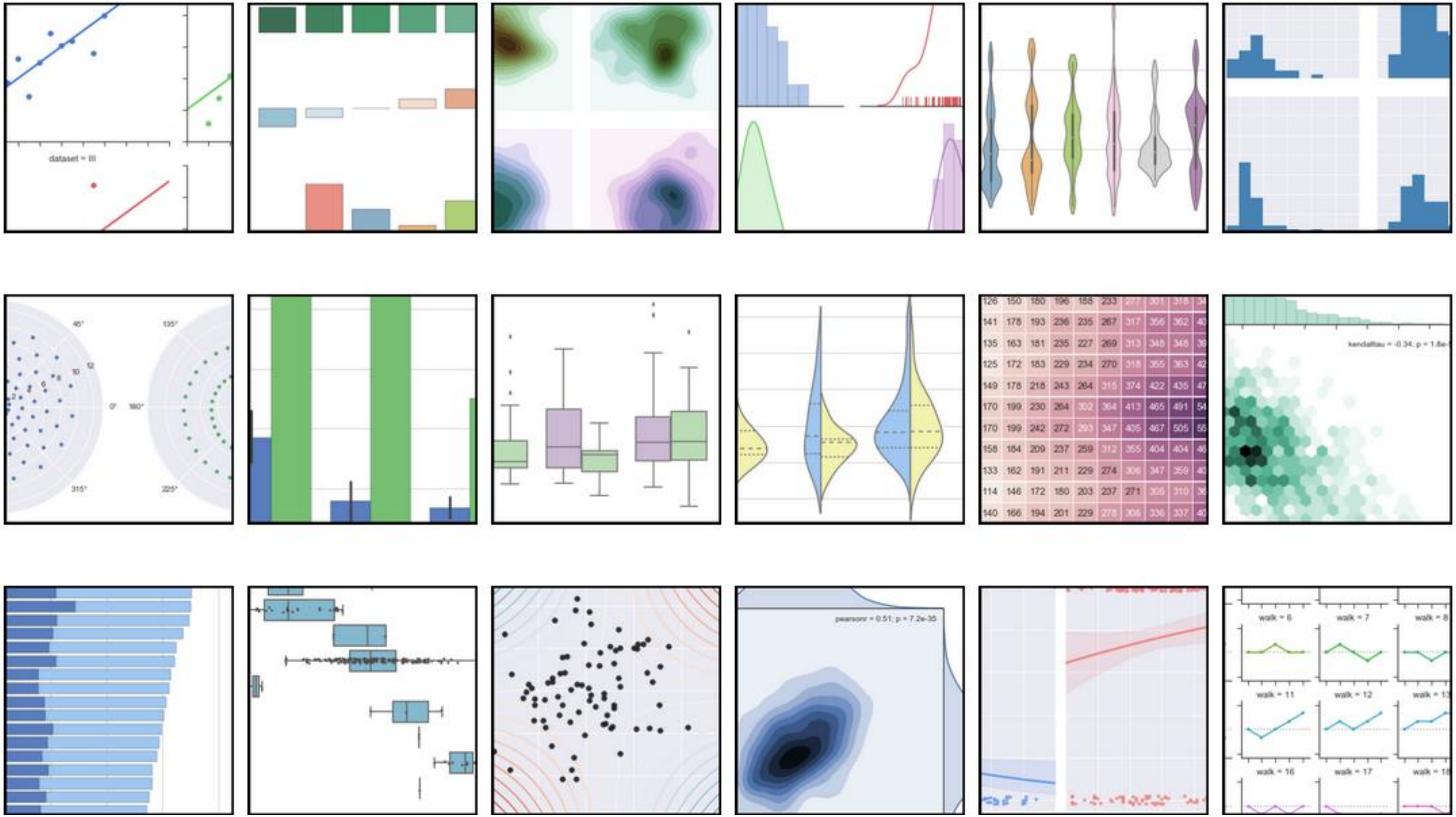


Notice how sales of "low fat" items fall off when the Outlet Size varies from Medium to High. Same with "Regular" items as well. This means that Sales tend to be smaller for Big Outlets.

© 2016 WIPRO LTD | WWW.WIPRO.COM | CONFIDENTIAL

Introduction to Seaborn

Powerful Statistical Visualization Made Easy. Provides numerous charting capabilities.



Source: Seaborn

3.1 Univariate Analysis

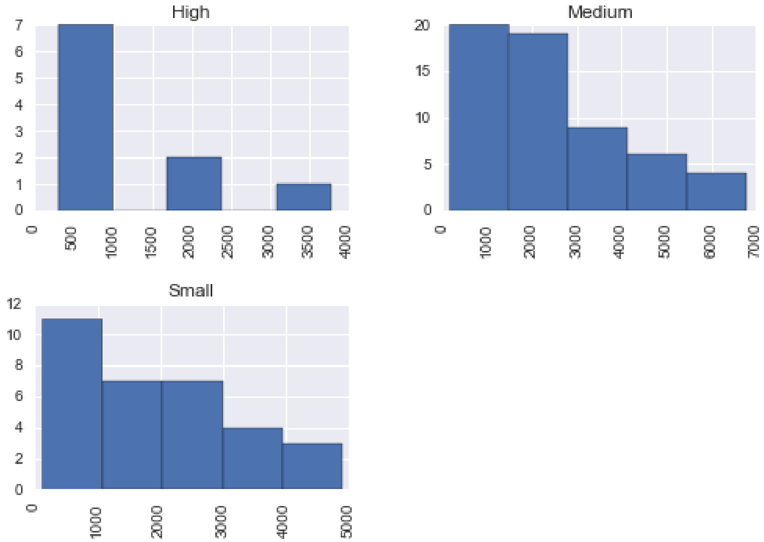
Univariate Analysis focuses on the behavior of a single variable

Analysis done to understand the distribution of the variable
Pyplot is great for simple charts such as:

- Boxplots: Powerful method to visualize Numeric Variables
- Time Series plots – allows you to look for seasonality in date/time variables
- Bar Charts – allows you to look at levels and freq for categorical variables
- Histogram – look at distribution of a continuous variable
- Density plots – probability distribution of a continuous variable

```
In [24]: df.hist(column='Item_Outlet_Sales',by='Outlet_Size',bins=5)

Out[24]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002C13B3BC278>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000002C13B425550>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000002C13B46BEB8>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000002C13B4A5DA0>]], dtype=object)
```

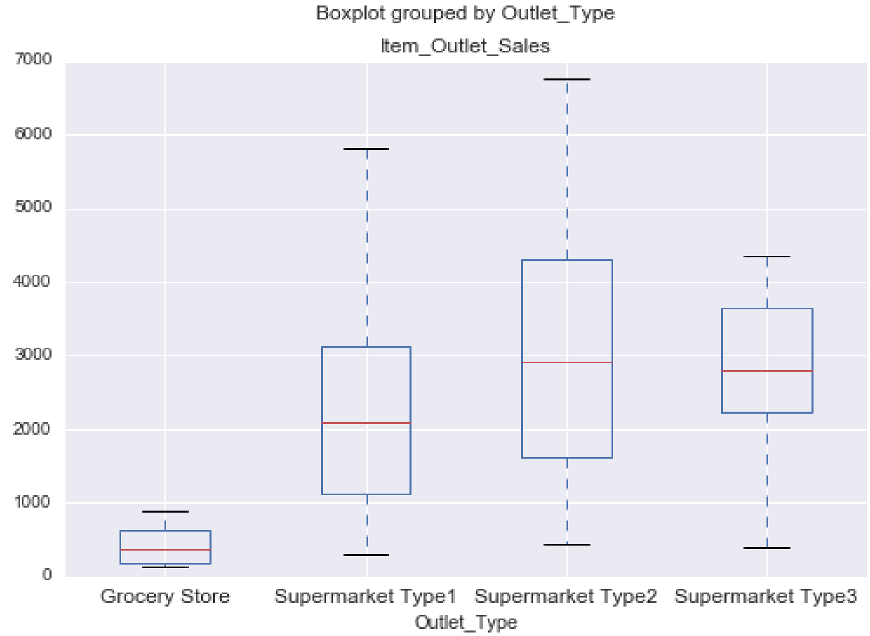


Python has Powerful Visualization Libraries

```
In [22]: import matplotlib.pyplot as plt
         %matplotlib inline
         import seaborn as sns

In [23]: #### Box Plots are a powerful method to visualize Numeric Variables
         df.boxplot(column='Item_Outlet_Sales',by='Outlet_Type')

ut[23]: <matplotlib.axes._subplots.AxesSubplot at 0x2c13b1a9748>
```



3.2 Bivariate and Multivariate Analysis

Analyze the behavior of two or more variables in tandem

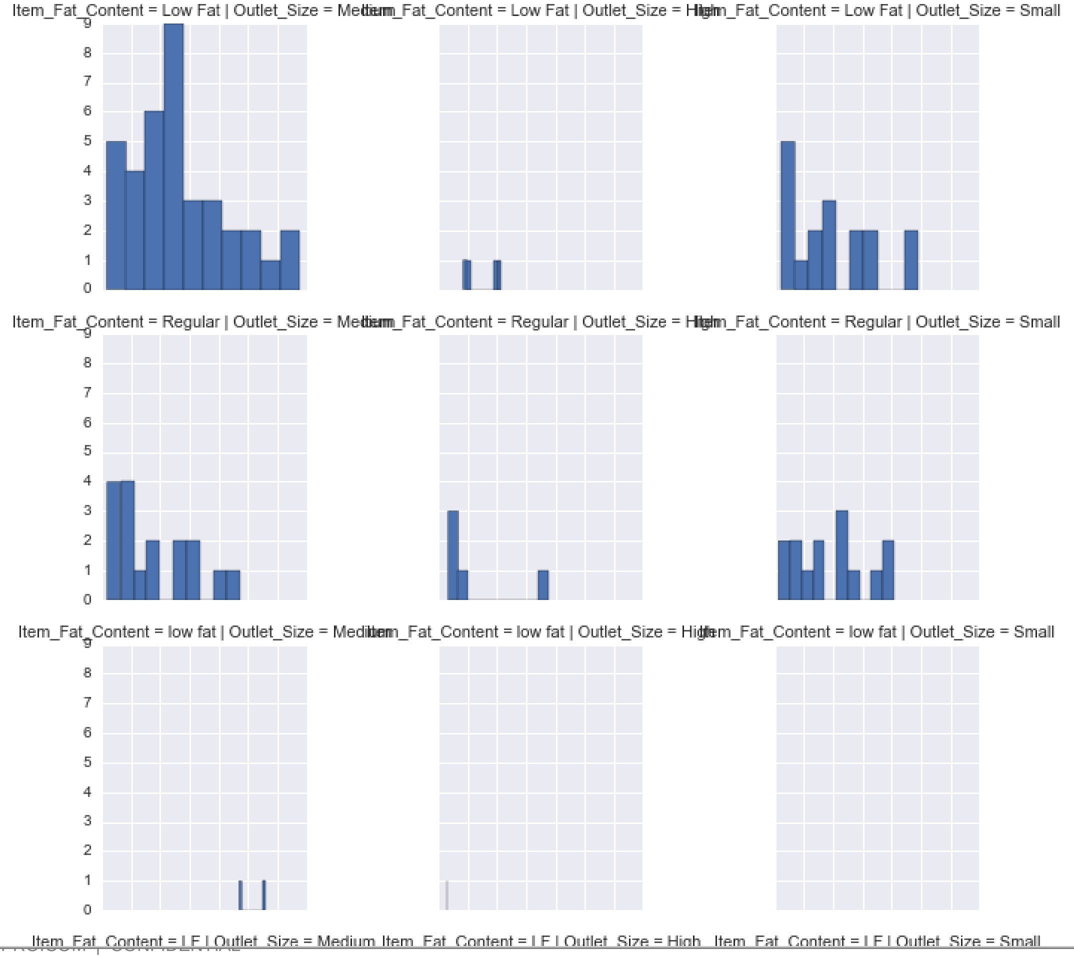
Seaborn is perfect for Bivariate and Multi-Variate Charts and Graphs, including:

- Factorplot
- FacetGrid
- PairGrid

And more.

Seaborn can create Complex Multivariate Plots

```
In [25]: 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()
```



Feature Engineering

Also known as Feature Extraction: it is the science of deriving “Signals” from “Noise”

4. Merge Old Variables or Derive New Variables

After Visualization

Python makes it easy to "derive" new columns from old columns

The values of the Outlet_Establishment_Year mean nothing unless it shows perhaps the "age" of the store.

Hence "Age of Outlet" is a natural "derived" variable to choose for predictive models, since we'd like to build models predicting based on "Age".

You can Drop columns, Fill null values, and Add columns

In [26]:
```
df['Age of Outlet']=2016-df['Outlet_Establishment_Year']

df.head(2)
```

Out[26]:

Weight	Outlet_Establishment_Year	Outlet_Identifier	Outlet_Location_Type	Outlet_Size	Outlet_Type	source	Age of Outl
	1999	OUT049	Tier 1	Medium	Supermarket Type1	train	17
	2009	OUT018	Tier 3	Medium	Supermarket Type2	train	7

In [27]:
```
df.drop(['Outlet_Establishment_Year'],axis=1,inplace=True)

df.head(2)
```

Out[27]:

ype	Item_Visibility	Item_Weight	Outlet_Identifier	Outlet_Location_Type	Outlet_Size	Outlet_Type	source	Age of Outlet
	0.016047	9.30	OUT049	Tier 1	Medium	Supermarket Type1	train	17
inks	0.019278	5.92	OUT018	Tier 3	Medium	Supermarket Type2	train	7

5. Changing Bucketing of Variables: “Binning”

After Visualization

```
In [12]: df.groupby('Item_Fat_Content').sum()
```

Out[12]:

	Item_Weight	Item_Visibility	Item_MRP	Outlet_Establishment_Year	Item_Outlet_Sales
Item_Fat_Content					
LF	3328.835	21.054330	43857.1062	631348	6.552424e+05
Low Fat	54687.900	326.541080	717390.8394	10167044	1.101503e+07
Regular	30341.545	200.970026	409413.1214	5771667	6.457454e+06
low fat	1055.375	7.313655	15071.7328	223539	2.338270e+05
reg	1361.320	7.764190	15948.6810	233923	2.295765e+05

Merge

LF, low fat and Low Fat look suspiciously similar
Same with Regular and reg

Change "Bucketing" of Variables: "Binning"

```
In [29]: df['Item_Fat_Content'].value_counts()

Out[29]: Low Fat    94
         Regular    45
         LF          6
         low fat     3
         reg         2
         Name: Item_Fat_Content, dtype: int64

In [31]: df['Item_Fat_Content'].replace({'low fat': 'Low Fat','LF':'Low Fat','reg':'Regular'},
                                        regex=True,inplace=True)
         df['Item_Fat_Content'].value_counts()

Out[31]: Low Fat    103
         Regular     47
         Name: Item_Fat_Content, dtype: int64
```

Python allows you to accomplish this with just one line of code!

Model Pre-Processing

The process of getting all data into numeric form so that a Model can be built

. Convert all Categorical Variables into Numeric Variables

Using SKLEARN which is one of Python's most famous Machine Learning Libraries

Python gives you a very easy way to "encode" any type of categorical variable into a numeric variable

Convert all Categorical Variables into Numeric Variables

```
In [ ]: from sklearn import preprocessing  ## sklern is ML library-- preprocessing
        encoding=preprocessing.LabelEncoder() ## create label encoding object
```

```
In [ ]: train['Item_Fat_Content'] = encoding.fit_transform(train['Item_Fat_Content'])
```

```
In [ ]: train['Item_Type'] = encoding.fit_transform(train['Item_Type'])
```

```
In [ ]: train['Outlet_Identifier'] = encoding.fit_transform(train['Outlet_Identifier'])
```

```
In [ ]: train['Outlet_Size'] = encoding.fit_transform(train['Outlet_Size'])
```

```
In [ ]: train['Outlet_Location_Type'] = encoding.fit_transform(train['Outlet_Location_Type'])
```

```
In [ ]: train['Outlet_Type'] = encoding.fit_transform(train['Outlet_Type'])
```

```
In [36]: df.head()
```

Out[36]:

	Item_Fat_Content	Item_Identifier	Item_MRP	Item_Outlet_Sales	Item_Type	Item_Visibility	Item_Weight	Outlet_Identifier	Outlet_Location_
0	0	FDA15	249.8092	3735.1380	4	0.016047	9.30	9	0
1	1	DRC01	48.2692	443.4228	14	0.019278	5.92	3	2
2	0	FDN15	141.6180	2097.2700	10	0.016760	17.50	9	0
3	1	FDX07	182.0950	732.3800	6	0.000000	19.20	0	2
4	0	NCD19	53.8614	994.7052	9	0.000000	8.93	1	2

Now almost all variables all Numeric except a few which is what we wanted

7. Preprocess Data for Modeling

Get Data Ready for Modeling

Once preprocessing is done, You are now ready for Modeling!

© 2016 WIPRO LTD | WWW.WIPRO.COM | CONFIDENTIAL