- PySpark Training

- Python & R Codes for Advanced Analytics & Machine Learning

- Sameer Patil (5-3782)
- Abhishek Vaidya (5-1674)
- Xuemei Shan (5-0172)

Agenda:

- Introduction to Training
- Hadoop Architecture
- Getting familiar with terminologies
- Python v R v SASimilarities & Differences -
- Python package list
- Environment Setup
- PySpark Training
    - Data Collection & Transformation
    - Data Analysis
    - Data Modeling
    - Advanced Analytics & Machine Learning

DISCOVER

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system on Spark framework. It helps to have a PySpark interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

This tutorial does not attempt to be comprehensive and cover every single feature, or even every commonly used feature. Instead, it introduces many of PySpark's most analytical worthy features, and will give you a good idea of the language's flavor and style. After reading it, you will be able to read and write PySpark modules and programs, and you will be ready to learn more about the various Python library modules

DISCOVER

Shift to Digital creating Data Explosion: *pose need for BIG DATA framework*

Challenges while working with BIG DATA is ability to capture, store & query at speed

Challenges in data capturing:

- Large volume and high velocity.
- For example, sensors which not only sense data like temperature of a room, weather parameters in real time, but send this information directly over to cloud for storage.

Challenges with data storage:

- Cost of Data Storage
- Ability to handle structured & unstructured data

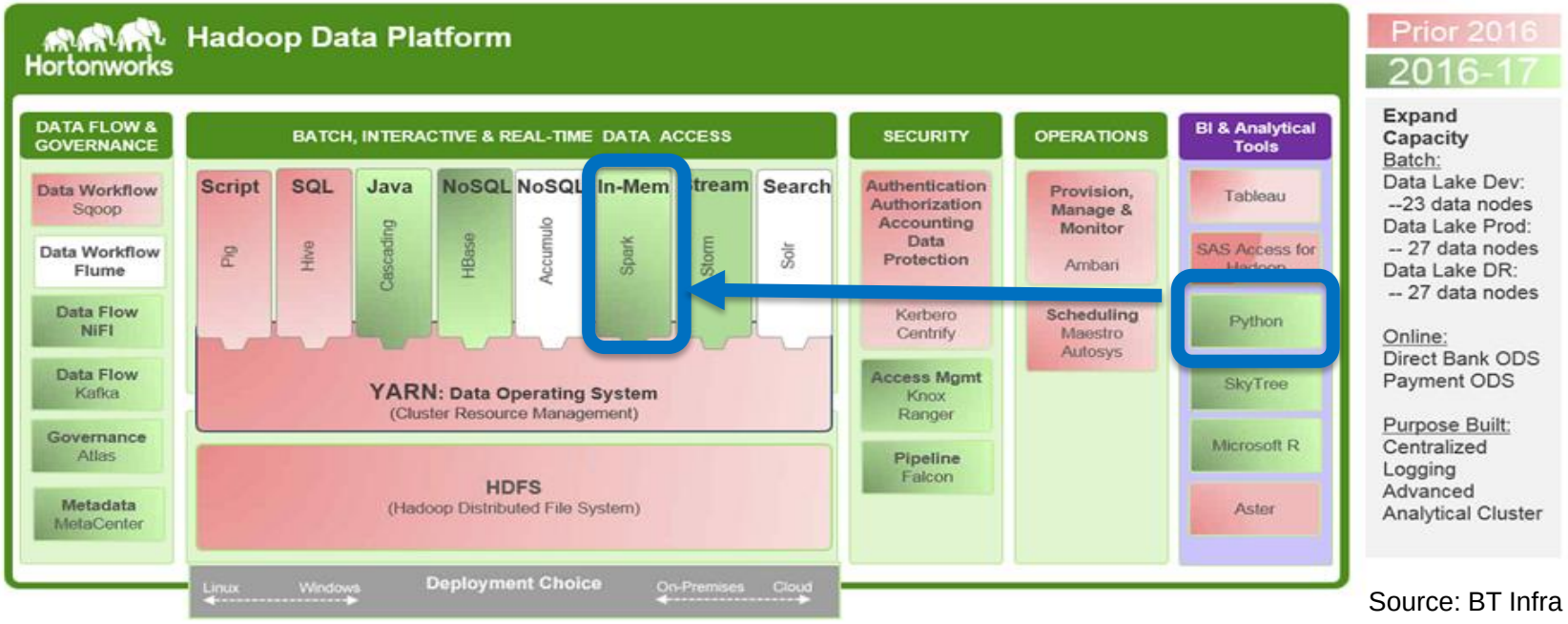
Challenges with Querying and Analysing data:

- Faster Processing
- Scalability
- Fault Tolerance

We need "Distributed Computing": Build a network of machines or nodes known as "Cluster". First break a task to sub-tasks and distribute them to different nodes. At the end, aggregate the output of each node to have final output.

Apache Hadoop & Apache Spark: *Cheap Data Storage & Faster Processing*

DFS Hadoop Ecosystem Overview – 2016 thru 2017



Source: BT Infra PMO team

- **Hadoop** is an open source, distributed computing, & Java-based programming framework that supports storage of extremely large data sets. **Hadoop** two-stage disk-based MapReduce paradigm supports batch processing
- For faster processing, **Spark's** in-memory primitives provide performance up to 100 times faster for certain applications
- **Python programming language on Spark Framework** is popular among Data Science community to perform Advanced analytics & Machine learning at speed & scale.

DISCOVER

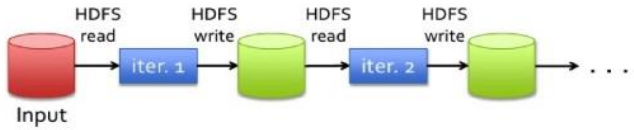
Python v SAS: *Benefits Outweighs Challenges in a BIG DATA environment*

Feature	SAS	Python	R
System	Closed Source	Open Source	
Analytical Modeling	High	Medium	High
Advanced Analytical capabilities	Not always enriched with latest statistical functions. Need to buy products	Has everything data scientist is looking for	Latest techniques gets released quickly
Hadoop BIG DATA Integration	Yes but recent	Yes	
Accessibility / Cost	Licensed & Expensive	Open Source & Free to download	
Ease of learning	Provides easy options known as procedures	Known for simplicity	Simple procedures can take longer codes
Advancements in Tool	New version roll outs	Open source gets its advanced features quickly	
Text Processing	Low	High	Medium
Graphical capabilities	Basic, customization on plots are difficult	Has native libraries matplotlib	Has most advanced graphical capabilities
Customer service & support community	Dedicated support & community	Lacks service support but supported by communities around the world	
Data Handling / Parallel Processing	All 3 have good data handling capabilities & options for parallel computations		
Customized, User defined	Limited	Exhaustive	
Governance & Control	Licensed & well tested	Open source but may not be well tested	
Companies using	Top Banks	Google	Bing, UBER

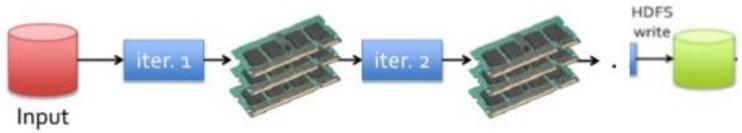
Introduction: *Python, Spark, PySpark & Dataframe*

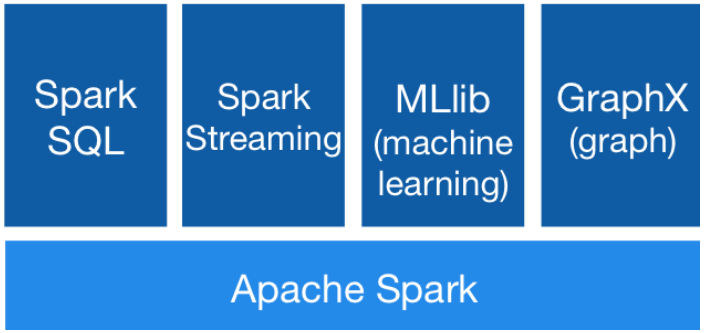
- **Python:** Python is a clear and powerful object-oriented programming language Data types supported in python.
  - ✓ **Python 2.7 –** Awesome community support, Plethora of 3rd party libraries. Many modules still don't work on 3.x
  - ✓ **Python 3.4 –** Cleaner & faster. Some features has backward compatibility with 2.7 version. It is the future!
- **Pyspark:** PySpark is the python binding for the Spark Platform and API. Simply, it is python programming on Spark framework
- **Python IDE/Editors:** ipython, Jupiter, eclipse (pydee), Emacs, PyCharm
- **Spark:** In-memory(RAM instead of disk(Hard Drive)) powerful open source processing engine built for faster processing & advanced analytics
  - Upto 100x times faster than Hadoop Map Reduce
  - Can write applications in Java, Scala, **Python, R**
  - Great for Online Transaction processing (OLTP)

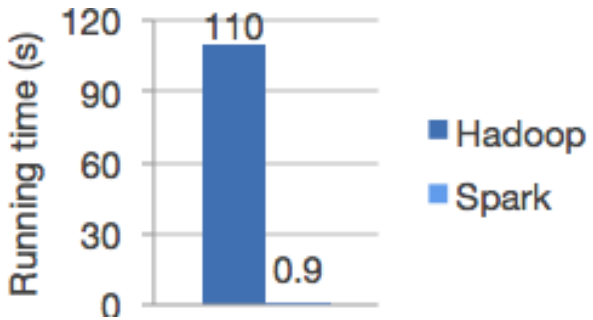
Map-Reduce : Read/Write at each iteration



Spark : In memory computation







Python Basics: *Data Structures*

- **Lists** – Lists are one of the most versatile data structure in Python. A list can simply be defined by writing a list of comma separated values in square brackets.

```
Squares_list = [ 0, 1, 4, 9, 16, 25]
```

- **Strings** – Strings can simply be defined by use of single (‘), double (”) or triple (”’) inverted commas. Strings enclosed in tripe quotes (”’) can span over multiple lines and are used frequently in docstrings (Python’s way of documenting functions). \ is used as an escape character.

```
Greeting = ‘ Hello World ’
```

- **Tuples** – A tuple is represented by a number of values separated by commas.

```
Tuple_example = 0, 1, 4, 9, 16, 25
```

- **Dictionary –** Dictionary is an unordered set of *key: value* pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}

```
extensions = (‘Kunal’ : 9073, ‘Pete’ : 9128, ‘Xian’ : 9330)
```

Python Basics: *Modules v Packages v Libraries in Python*

- **Module:** a module in python is a .py file that defines one or more function/classes which you intend to reuse in different codes of your program. It can define functions, classes & variables. A module is a single file of python code that is meant to be imported. **Think of it as specific Procedure or function in SAS**

- **Package:** A Python package refers to a directory of Python module(s). This feature comes in handy for organizing modules of one type at one place. Think of it as alias given to **group of Procedures in SAS**

- **All packages are modules, but not all modules are packages.**

- **Libraries:** When used in Python, a library is used loosely to describe a collection of the core modules. **Libraries & Packages are interchangeable**

- **Functions:** A function is a name container for block of organized, reusable code that is used to perform a single, related action. Python gives you many built-in functions like print(), etc. but you can also create your own functions. These functions are called user-defined functions. **Think of it as Macros or SAS functions**

In Python, blocks of code can be named. Such named blocks of code can be run from elsewhere in your program, and are known as functions.
A module is a text file that contains a number of functions and other items, and a package is a group of related modules.

DISCOVER

Python Basics: *Libraries for Data Analysis & Machine Learning*

- **SymPy** for symbolic computation. It has wide-ranging capabilities from basic symbolic arithmetic to calculus, algebra, discrete mathematics and quantum physics.
- **NumPy** stands for Numerical Python. The most powerful feature of NumPy is n-dimensional array. This library also contains basic linear algebra functions,
- **SciPy** stands for Scientific Python. SciPy is built on NumPy. It is one of the most useful library for variety of high level science and engineering modules
- **Scikit Learn** for machine learning. Built on NumPy, SciPy and matplotlib, this library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.
- **Statsmodels** for statistical modeling. Statsmodels is a Python module that allows users to explore data, estimate statistical models, and perform statistical tests. An extensive list of descriptive statistics, statistical tests, plotting functions, and result statistics.
- **Matplotlib** for plotting vast variety of graphs, starting from histograms to line plots to heat plots.. **Pandas** for structured data operations and manipulations.
- **Seaborn** for statistical data visualization. Seaborn is a library for making attractive and informative statistical graphics in Python. It is based on matplotlib.
- **Pandas** is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series

Python Basics: *Additional Libraries for visualization, web crawling & math*

- **Bokeh** for creating interactive plots, dashboards and data applications on modern web browsers Moreover, it has the capability of high-performance interactivity over very large or streaming datasets.

- **Blaze** for extending the capability of Numpy and Pandas to distributed and streaming datasets. It can be used to access data from a multitude of sources including Apache Spark, PyTables, etc. Together with Bokeh, Blaze can be very powerful tool for creating visualizations and dashboards on huge data.

- **Scrapy** for web crawling. It is a very useful framework for getting specific patterns of data. It has the capability to start at a website home url and then dig through web-pages within the website to gather data

Requests for accessing the web. It works similar to the the standard python library urllib2 but is much easier to code. Requests might be more convenient.

Additional libraries, you might need:

- **os** for Operating system and file operations
- **networkx** and **igraph** for graph based data manipulations
- **regular expressions** for finding patterns in text data
- **BeautifulSoup** for scrapping web. It is inferior to Scrapy as it will extract information from just a single webpage in a run.
- **math** provides access to mathematical functions
- **Cmath** provides access to complex mathematical functions

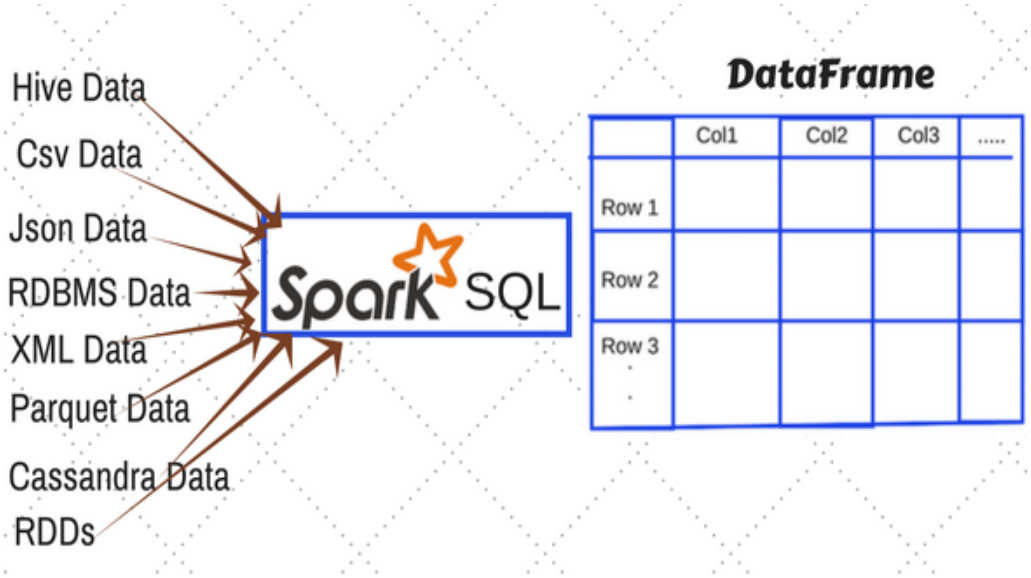
DISCOVER

Introduction to DataFrame: *Analogous to dataset in SAS*

In Apache Spark, DataFrame is a distributed collection of rows under named columns

Why DataFrames are useful?

- Brings structure to a file.
- Has the ability to handle Petabytes of data
- Can be creating using different data formats. For example: Hive, Csv, Json etc
- Helps Spark to understand schema & optimize execution plan of queries

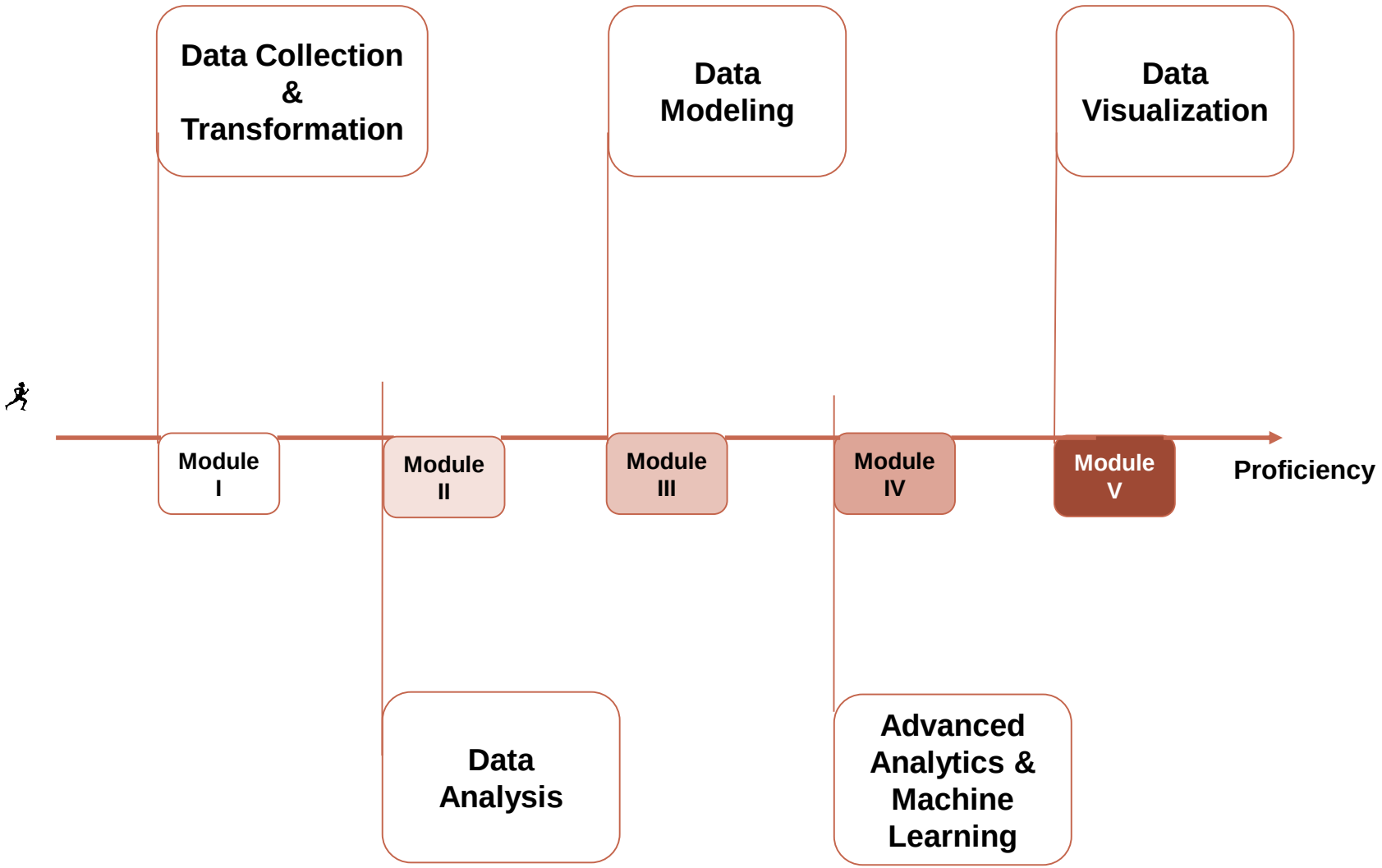


Key Difference between Pandas and PySpark DataFrame:

- Operation on Pyspark DataFrame run parallel on different nodes in cluster but, not possible in Pandas.

Training Modules based on PySpark dataframe for its ability to process data faster

PySpark Training Modules: *Window to PySpark analytics worthy concepts & features*



DISCOVER

PySpark Training Modules: *Designed for Data Analysts & Scientists*

Analysis Process	Description	Python Packages	Modules
Data Collection, Transformation & Quality Check	Load and transform data. It's process of converting data or information from one format to another, which includes: • Loading, Selection • Filtering • Sorting • Joining • Grouping • Missing Values	NA	• Filter Rows, • Keep or Drop Attributes, • Merge & Join Tables
Data Analysis	Feel the Data • Sampling • Distribution Analysis • Measure of Central Tendency & Dispersions • Initial Variable Screening	numpy	• Mean, • Min, • Max, • Stdv
Data Modeling	Understand how to apply basic regression models to data set using spark framework and pyspark packages	Statmod, Scipy	• Regression, • DecisionTree
Advanced Analytics & Machine Learning	Pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data.	Mllib, scikitlearn	• GBT, SVM, RF • K-Means, K-NN • Collaborative
Data Visualization	Establish patterns, correlations & trends • Bar Chart • Histogram • Scatter Plot	Matplot	• Plot, • Axis, • Annotate, • line

Training Setup: *Environment, Data & Legends*

- **Environment Set up Guide**: Please follow steps in attached guide to set up environment. It's one time activity.
- After log-in (do not enter beeline or exit out), At command prompt, type *pwd* → */dsap/home/<userid>*
- Create 2 input files */dsap/home/<avaidy1* **Input File1**: *superstore.csv* **Input File2**: *returns.csv*
  - *vi superstore.csv* (to create file)
  - *Esc i shift+insert(or right click)* to paste data copied from input file1
  - *Esc :wq* to save & exit
- **Input File 1:** This is our sample input file one. It has customer order history data, It has card member demographics & key metrics like, sales profit and discounts at order level
- **Input File 2:** This is our another sample input file. It has customer order history data and indicator if that order has been returned or not. Primarily we will use this file to join with another input file based on order number.
- **Legends: (Color Codes)**
  - **blue:** PySpark parameters & functions
  - **green** : comments
  - Dataframe - prefix with "df"


Environment Setup Guide


Input File 1


Input File 2

Module I: **Data Collection & Transformation**

Welcome! In this module, you'll learn how to load and transform data. It’s process of converting data or information from one format to another, usually from the format of a source system into the required format of a new destination system.

- **DATA & PYSPARK SET-UP**
- **DATA LOADING**
- **DATA DESCRIPTION**
- **DATA PRINT**
- **DATA DUPLICATION & DATE FUNCTION**
- **DATA FILTERING OR SUBSETTING**
- **DATA SORT**
- **DATA MERGING OR DATA JOINS**
- **DATA GROUPING**

- ✓ **Commands are case-sensitive Ex: –copyFromLocal**
- ✓ **Windows Quotation Marks don’t work**
- ✓ **Copy & Paste carefully. Check for spaces & syntax**

DISCOVER

Data & PySpark Setup: *Copy Files & invoke spark framework*

Comments

(1)

- Create directory called train using –mkdir in HDFS;
  - prefix hdfs dfs to command. It helps to execute it on Hadoop node
  - train directory is created in HDFS under your user id

(2)

- To check if train directory is created, use
  - hdfs dfs -ls

(3)

- Copy input file superstore.csv from local directory to hdfs using
  - –copyFromLocal hdfs command
  - Using hdfs dfs –cat and head -10, print 10 observations in HDFS

(4)

- Spark Python API: PySpark & exposes the Spark programming model to Python.

Code:

1. Create directory 'train' in HDFS

➢ hdfs dfs -mkdir /user/<user-id>/train

➢ Ex: hdfs dfs -mkdir /user/avaidy1/train

2. To check if directory is created in HDFS

➢ hdfs dfs -ls /user/<user-id>/

➢ Ex: hdfs dfs -ls /user/avaidy1/

3. Copy sample input files from local to HDFS

➢ hdfs dfs –copyFromLocal */dsap/home/<user-id>*/superstore.csv
/user/*<userid>*/train/superstore.csv

Example:

➢ hdfs dfs -copyFromLocal
/dsap/home/avaidy1/superstore.csv
/user/avaidy1/train/superstore.csv

To check file is populated, print 10 obs

➢ hdfs dfs -cat /user/avaidy1/train/superstore.csv | head -10

4. Invoke python on spark framework

➢ pyspark

Data & PySpark Setup: *Import Libraries*

Comments

(1)

- Import command brings installed libraries to the current session
  - For example: Pyspark.sql is a library/package
  - Row is a module from that package

Code:

1.Import libraries

```
from pyspark.sql import Row,column
from pyspark.sql import *
from datetime import datetime
from pyspark.sql.types import
 DateType,StringType
from pyspark.sql.functions  import *
```

DISCOVER

Data Loading: *Load file & convert in table format in Spark*

SAS equivalent: *PROC IMPORT*

Comments

(1)
Load input csv file from HDFS onto spark framework and assign file name customer.
- SC is SparkContext, it establishes a connection to a Spark execution environment

(2)
map transformation & lambda functions used so that each word split by ‘,’ are put into a column.
http://spark.apache.org/docs/2.0.2/programming-guide.html#transformations

(3)
- Assigning a column name to each column
  - Use in-built ROW function

(4)
- To create dataframe(analogous to dataset in SAS)
  - Use createDataFrame function from sqlContext to create and define dataframe named as df_schemaTable.

Code:

1. Load input file
```
customer=sc.textFile("///user/<user_input>/train/superstore.csv")
```
Example:
```
customer=sc.textFile("///user/avaidy1/train/superstore.csv")
```

2. Split input file
```
cust_col= customer.map(lambda word: word.split(','))
```

3. Getting ready for creating DataFrame
```
cust_table= cust_col.map(lambda p: Row(ROW_ID=p[0], ORDER_ID=p[1], ORDER_DATE=p[2], SHIP_DATE=p[3], SHIP_MODE=p[4], CUSTOMER_ID=p[5], CUSTOMER_NAME=p[06]))
```

4. Create Dataframe
```
df_schematable = sqlContext.createDataFrame(cust_table)
```

DISCOVER

Data Structure: *Check column names, data types & variable lengths*

SAS equivalent: *PROC CONTENTS*

Comments

(1)
- PrintSchema will help schema structure including column name and its datatype
- Syntax: <dataframe>.PrintSchema()

(2)
- To get feel of the data, checking the variable length can be done by
- Syntax: <dataframe>.select(max(length('<variable>'))).show()

(3)
- While conducting data QC, leading & trailing white spaces can be removed by

Syntax:
<new data frame name> = <dataframename>.withColumn('<newvar>',trim(<dataframe>.<variable>))

(4)
- To elimintate spaces between words in a column
- Syntax: build user define function, use like (3)
- spaceDeleteUDF = udf(lambda s: s.replace(" ", ""), StringType())

Code:

1. Print your table schema

```
df_schematable.printSchema()
```

```
root
 |-- CUSOTMER_ID: string (nullable = true)
 |-- CUSTOMER_NAME: string (nullable = true)
 |-- ORDER_DATE: string (nullable = true)
 |-- ORDER_ID: string (nullable = true)
 |-- ROW_ID: string (nullable = true)
 |-- SHIP_DATE: string (nullable = true)
 |-- SHIP_MODE: string (nullable = true)
```

2. Calculating length of the variable

```
df_schematable.select(max(length('ORDER_ID'))).show()
```

3. Trim white space of a variable

```
df_trim=df_schematable.withColumn('ROW_ID',trim(df_schematable.ROW_ID))
```

- ltrim for left trim
- rtrim for right trim

4. Remove blank space from column name

```
spaceDeleteUDF = udf(lambda s: s.replace(" ", ""), StringType())
df_check=df_schematable.withColumn("CNAME",spaceDeleteUDF("CUSTOMER_NAME"))
df_check.show()
```

DISCOVER

Data Print: *to hawk-eye the data*

SAS equivalent: *PROC PRINT*

Comments

(1)
- Prints all column in a dataframe with default 20 records
  - Syntax: <dataframe>.select('*'). show()

- User can change the default valye
  - Syntax: <dataframe>.select('*'). show(<n>)

- To print specific column
  - Syntax: <dataframe>.select('<var>'). show()

(2)
- Sample the output with 50 records
  - Syntax: <dataframe_new> = <dataframe>.limit(<n>)

(3)
- Count number of rows in dataframe
  - Syntax: <dataframe>.count()

(4)
- Columns in dataframe
  - Syntax: len(<dataframe>.columns),<dataframe>.columns

Code:

```
1. will print all columns with default 20 records
➢ df_schematable.select('*'). show()

will print all columns with 10 records
➢ df_schematable.select('*'). show(10)

Example1:
#Specific column
➢ df_schematable.select('ROW_ID').show()

Example 2:
#Specific multiple columns
➢ df_schematable.select('ROW_ID','CUSTOMER_NAME').
  show()

2. Limit your output
➢ df_limit50=df_schematable.limit(50)
➢ df_limit50.show()

3. Count numbers of rows in dataframe
➢ df_schematable.count()

4. Columns in dataframe
➢ len(df_schematable.columns),df_schematable.colum
  ns
```

DISCOVER

Data Quality Checks: *Count observations, check for missing, eliminate duplicates*

SAS equivalent: *COUNT(), NODUPKEY, MISSING*

Comments

(1)
- To count distinct values or check duplicates
  - Syntax: <dataframe>.select(countDistinct('<var>')).show()
- To count approximate distinct values
  - Synax: <dataframe>.select(approxCountDistinct('R<var>')).show()
  - Please note, being a distributed data and the way it works at backend it provides you approximate distinct count

(2)
- To remove duplicates from your dataframe
  - Syntax: <dataframe_new>=<dataframe>.dropDuplicates(['<var>']).show(50)

(2)
- To check missing value or non-missing values
  - Syntax: (use isNotNull() for non-missing)
  - <dataframe>.where(<dataframe>.<var>.isNull()).count()

Code:

1. Will give exact Distinct Count
```
df_schematable.select(countDistinct('ROW_ID')).show()
```
Ans: 50

Will give approximate Distinct Count
```
df_schematable.select(approxCountDistinct('ROW_ID')).show()
```
Ans: 47

2. Remove duplicate values from file
```
df_duplicate=df_schematable.dropDuplicates(['CUSTOMER_NAME']).show(50)
```

3. Count Missing Values
```
df_schematable.where(df_schematable.CUSTOMER_NAME.isNull()).count()
df_schematable.where(df_schematable.CUSTOMER_NAME == " ").count()
```

4. Counting Missing Values
```
df_schematable.where(df_schematable.CUSTOMER_NAME.isNotNull()).count()
df_schematable.where(df_schematable.CUSTOMER_NAME <> " ").count()
```

DISCOVER

Data Period: *Handling date variables to calculate difference & check data period*

SAS equivalent: INPUT(), PUT(), MDY(), INTCK()

Comments

(1)

- To calculate difference between two datses
  - Convert variable from string into date by creating user define function
    - Syntax: func = udf (lambda x: datetime.strptime(x, '%M/%d/%Y'), DateType())
  - Apply user define function
    - Syntax: <dataframe_new> = <dataframe>.withColumn('<newvar>',func(col('<var>')))
  - <dataframe>.printschema() to check/confirm
  - Calculate difference
    - Syntax: <dataframe_new>=<dataframe>.withColumn('<newvar>',datediff('<fromdate>', '<todate>'))
  - Efficient way to calculate difference
    - Syntax: <dataframe_new>=<dataframe>.withColumn('<newvar>',datediff(func(col('<fromdate>')),func(col('<todate>'))))
- Reference: List of date and other functions

http://spark.apache.org/docs/latest/api/scala/index.html#org.apache.spark.sql.functions$

Code:

```
1. Converting string into date
➢ func =  udf (lambda x: datetime.strptime(x, '%M/%d/%Y'), DateType())
Converting SHIP_DATE string into S_DATE date
➢ df_datetest = df_schematable.withColumn('S_DATE',func(col('SHIP_DATE')))
➢ df_datetest.printSchema()
Converting ORDER_DATE string into O_DATE date
➢ df_datetest1 = df_datetest.withColumn('O_DATE',func(col('ORDER_DATE')))
➢ df_datetest1.printSchema()
Print Days difference variable ‘DURATION’
➢ df_datetest1.select(datediff('S_DATE','O_DATE').alias('DURATION')).show()
APPEND days difference variable ‘DURATION’
➢ df_datetest2=df_datetest1.withColumn('DURATION', datediff('S_DATE','O_DATE'))
➢ df_datetest2.show()
Efficient way
➢ df_efficient=df_schematable.withColumn('DURATION',datediff(func(col('SHIP_DATE')),func(col('ORDER_DATE'))))
➢ df_efficient.show()
```

DISCOVER

Data Sorting & Sub-setting: *to define segments and/or filter and/or sort population*

SAS equivalent: IF, IF-THEN-ELSE statements, PROC SORT

Comments

(1)
- Subset or filter data for a value/segment
  - Syntax:
    <dataframe_new>=<dataframe>.filter(<dataframe>["<var>"] == '<variablevalue')
  - filter() – Filters row with given condition and returns true results

(2)
- Subset or filter data based on length
  - Syntax:
    <dataframe>.where(length(col('<var>'))<=12).show()

(3)
- Conditional evaluation
  - Syntax:
    <dataframe_new>=<dataframe>.withColumn('<newvar>',when(<dataframe>.<var> == '<variablevalue>',<truevalue>).otherwise(<falsevalue>))

(4)
- Sorting dataframe
  - Syntax:
    <datafreme_new>=<dataframe>.sort(('var>'),ascending=True)
  - ascending=False equals descending

Code:

```
1. Filter or subset data
➢df_filtersql=df_schematable.filter(df_schematable["CUSTOMER_NAME"] == 'Claire Gute')
➢ df_filtersql.show()


2. Filtering data based on length of variable
➢ df_schematable.where(length(col('CUSTOMER_NAME'))<=12).show()

3. IF-THEN-ELSE
➢ df_cond=df_schematable.withColumn('IND',when(df_schematable.CUSTOMER_NAME == 'Claire Gute',1).otherwise(0))
➢ df_cond.show()

4. Data sorting Ascending or Descending
➢ df_sorttab=df_schematable.sort(('CUSTOMER_NAME'),ascending=True)
➢ df_sorttab.show()
```

DISCOVER

Data Transformation: *to derive/define new variables or modify existing ones*

SAS equivalent: RENAME,INPUT, PUT, SUBSTR, LOG, EXP, INVERSE

Comments

(1)

- Renaming a variable
  - Syntax:
    <dataframe_new>=<dataframe>.selectExpr("<oldvar1> as <newvar1>","<var2>")

(2)

- Substring first n characters
  - Syntax:
    <dataframe_new>=<dataframe>.withColumn('<newvar>',substring(<dataframe>.<var>,<startingposition>,<n>))

(3),(4),(5),(6),(7)

- Variable Transformation
  - Syntax:
    <dataframe_new>=<dataframe>.withColumn('<newvar>',log('<var>'))
  - exp(exponential),sqrt(squareroot)
  - Multiplication: col('<var1>')*col('<var2>')
  - Division or Inverse: 1/col('<var>')
  - Concatenate:
    concat(col('<var1>'),col('<var2>'))

(8)

- Coverting data types from string to float
  - Syntax:
    df_cond1=df_cond.withColumn('<newvar>',<dataframe>.<var>.cast('float'))

Code:

1. Rename attribute RET_ORDER_ID as ORDER_ID

```
df_as3=df_schematable.selectExpr("ORDER_ID as ORDER_ID","ORDER_DATE")
```

2. Substring first 5 letters of CUSTOMER_NAME

```
df_substr=df_schematable.withColumn('CNAME',substring(df_schematable.CUSTOMER_NAME,1,3))
```

3. Natural log transformation

```
df_cond1=df_cond.withColumn('ROW_ID_log',log('ROW_ID'))
```

4. Natural exponential transformation

```
df_cond1=df_cond.withColumn('ROW_ID_log',exp('ROW_ID'))
```

5. Inverse of a column

```
df_cond1=df_cond.withColumn('ROW_ID_log',1/col('ROW_ID'))
```

6. Multip;ying two columns

```
df_cond1=df_cond.withColumn('ROW_ID_log',col('ROW_ID')*col('IND'))
```

7. Concatenate two variables

```
df_cond1=df_cond.withColumn('ROW_ID_log',concat(col('ROW_ID'),col('IND')))
```

8. Converting from string to float

```
df_cond1=df_cond.withColumn('ROW_ID_log',df_cond.ROW_ID.cast('float'))
```

Exercise

Exercise I:

1) Load input file 2 return.csv and convert into dataframe
2) Check columns, data types & variable lengths
3) Print observations
4) Count observations, check for missing values & eliminate duplicates
5) Sort Data

Exercise II:

1) Sample 20 observations from dataframe “df_schematable”
2) Store sample 20 observations in new dataframe ‘df_sample’
3) Convert the data type of date variables SHIP_DATE & ORDER_DATE from string to date.
4) Create corresponding new date variables ‘S_DATE’ & ‘O_DATE’
5) Create new dataframe “df_sample_date”

Exercise III:

1) For dataframe “df_schematable”, Create indicator variable ‘IND’ and assign value ‘1’ if AMNT > 20000, else ‘0’. Create new dataframe “df_amnt”

Exercise: *Answers*

Exercise I:

1) Load input file 2 return.csv and convert into datafram
2) Check columns, data types & variable lengths
3) Print observations
4) Count observations, check for missing values & elimi
5) Sort Data

- `ret=sc.textFile("///user/spati23/train/return.csv")`
- `ret_col= ret.map(lambda word: word.split(','))`
- `df_ret_table= ret_col.map(lambda p: Row(RETURN_FLAG=p[0], RET_ORDER_ID=p[1]))`
- `df_schtab_ret = sqlContext.createDataFrame(df_ret_table)`
- `df_schtab_ret.select('RET_ORDER_ID').show()`

Exercise II:

1) Sample 20 observations from dataframe “df_schematable”
2) Store sample 20 observations in new dataframe ‘df_sample’
3) Convert the data type of date variables SHIP_DATE & ORDER_DATE from string to date.
4) Create corresponding new date variables ‘S_DATE’ & ‘O_DATE’
5) Create new dataframe “df_sample_date”

```
df_sample=df_schematable.limit(20)
func =  udf (lambda x: datetime.strptime(x, '%M/%d/%Y'), DateType())
df_shipdate=df_sample.withColumn('S_DATE',func(col('SHIP_DATE')))
df_sample_date=df_shipdate.withColumn('O_DATE',func(col('ORDER_DATE')))
```

Exercise III:

- For dataframe “df_schematable”, Create indicator variable ‘IND’ and assign value ‘1’ if AMNT > 20000, else ‘0’.

```
df_step=df_schematable.withColumn('AMNT',col('ROW_ID')*500)
df_shipdate1=df_step.withColumn('S_DATE',func(col('SHIP_DATE')))
df_amnt=df_shipdate1.withColumn('IND',when(df_step.AMNT>20000,1).otherwise(0))
```

DISCOVER

Data Joins / Merging: *to define segments and/or filter and/or sort population*

SAS equivalent: MERGE STATEMENT, PROC SQL - JOIN

Comments

(1)

- After completing the exercise, two dataframes df_amnt & df_sample_date will be created

- To join two dataframes
  - Drop duplicates based on key variables
  - Syntax: <dataframe_new>=<dataframe>.dropDuplicates(['keyvar1', 'keyvar2'])
  - Create Alias
  - Syntax: <aliasname>=<dataframe>.alias("<aliasname>")

- Merge two dataframes
  - Syntax : join(other file, on=Key, how=JoinType)
  - <datafraem_new>=<alias1>.join(<alias2>,(<alias1>.<keyvar1> == <alias2>.<keyvar1>) & (<alias1>.<keyvar2> == <alias2>.<keyvar2>),"<jointype>").select('<alias1>.*')
  - select('df1.*') keeps columns from dataframe1. To keep columns from dataframe2, select('df1.*','df2.S_DATE')

Code:

1. Join Files

Drop Duplicates from the file to be merged

```
df_sample_date_unique=df_sample_date.dropDuplicates(['ORDER_ID','S_DATE'])
```

Create Alias for two dataframes to be merged

#Table1

```
df1=df_amnt.alias("df1")
```

#Table2

```
df2=df_sample_date_unique.alias("df2")
```

#join tables Join(other table, on=COL1, how=inner)

```
df_joined=df1.join(df2,(df1.ORDER_ID == df2.ORDER_ID) & (df1.S_DATE == df2.S_DATE), "left").select('df1.*')
```

#Display

```
df_joined.count()
df_joined.show(50)
```

- You can use following types of join:
  - inner
  - outer
  - left_outer
  - right_outer

DISCOVER

Cheat Sheet : SAS v Python: Module I

Statistics	SAS	Python (Pyspark)
Load / Import textfile Dataset	PROC IMPORT	sc.textFile("*/file_path*")
Show schema / Display contents	PROC CONTENTS	Input_file_df.show()
Select Column	*KEEP Statement*	df.select(*columns*).show()
Print Schema	PROC CONTENTS	df.printSchema().show()
Filter	*Where =*	df.filter(*condition*).show()
Sort	*PROC SORT*	df.sort(columns).show()
Join	*MERGE Statement*	df1.join(df2,*join_col*,*join_type).*show()
Group	BY Statement	df.groupBy(*column*).show()
Data distribution	*PROC FREQ*	df.describe(*column*).show()
Distinct	*PROC SORT*	df. approxCountDistinct(column).show()
Date Functions	*INTX,(). INTCK()*	df.datediff('end_date_col', 'start_date_col').show()
Derived Column	column1' * 'column2	df.select('column1' * 'column2).show() or df.select('column' + 12)
Type Casting	*INPUT() , PUT()*	df.select('column'.cast(data_type)).printSchema()

Module II: **Data Analysis**

Welcome! Learn how to systematically apply statistical and/or logical techniques to describe and illustrate, your data.

- **MEAN, MEDIAN, STANDARD DEVIATION**
- **PERCENTILE, MIN, MAX**
- **FREQUENCY DISTRIBUTION & CROSS TABULATION**
- **RANDOM & STRATIFIED SAMPLING**

DISCOVER

Variable Screening: *based on business sense & missing values*

SAS equivalent: DROP

Comments

(1)

- Initial Variable Screening: Drop variables that don't make business sense or can't be used
  - Syntax: <_dataframe_new>=<dataframe>.drop('<dropvar>')

(2)

- Measures of Central Tendency & Dispersion:
  - Screen out variable when stddev=0
  - Syntax: <_dataframe_new>=<dataframe>.describe()
  - describe() – Count, Mean, Stddev, Min, Max for numeric variables
  - For categorical variable, it will just give count

(3)

- Keep variables that make analytical sense
  - Syntax: <dataframe_new>=<dataframe>.select('var1', 'var2', 'var3')

(4)

- Exclude variables based on high % of missing
  - Syntax: <dataframe>.where(<dataframe>.<var> == " ").count()

Code:

1. Droping variables based on business sense

```
df_drop=df_amnt.drop('CUSTOMER_NAME')
df_drop.show()
```

2. Dropping variables when stddev=0

```
df_summary=df_amnt.describe()
df_summary.show()
```

3. Keep variables that only make analytical sense

```
df_analysis=df_amnt.select('ROW_ID','AMNT','IND'
)
df_analysis.show()
```

4. Drop variable with high % of missing values

```
df_amnt.where(df_amnt.AMNT == " ").count()
```

Distribution: *for Trend Analysis*

SAS equivalent: PROC MEANS, PROC UNIVARIATE, PROC FREQ

Comments

(1)
- Measures of Central Tendency & Dispersion:
  - Screen out variable when stddev=0
  - Syntax: <_dataframe_new>=<dataframe>.describe()

(2)
- Cross Tabulation:
  - Syntax: <data_frame>.crosstab('<var1>', '<var2>').show()

(3)
- Central Tendency/Dispersion by segment
  - Syntax: <dataframe>.filter(<dataframe>.<var><cond>).describe().show()

(4)
- Univariate Disrtibution
  - Syntax: <dataframe>.groupBy(<dataframe>.<var>).count()

(5)
- Mean of each group
  - Syntax: <dataframe>.groupby('<groupbyvar>').agg({'<var>':'<summaryfunction>'}).show()

Code:

1. Mean & Standard Deviation, Min & Max
```
df_summary=df_amnt.describe()
df_summary.show()
```

2. Cross Tabulation
```
df_amnt.crosstab('SHIP_MODE','IND').show()
```

3. Count observations beyond a threshold
```
df_amnt.filter(df_amnt.AMNT>=10000).describe().show()
```

4. Univariate Distribution
```
df_group=df_amnt.groupBy(df_amnt.ORDER_DATE).count().show()
```

5. Mean of each group
```
df_amnt.groupby('SHIP_MODE').agg({'AMNT':'mean'}).show()
```

DISCOVER

Data Analysis: *Applying SQL queries on DataFrame*

SAS equivalent: PROC SQL

Comments

(1)
- To run SQL queries
  - Register data frame as Table
  - Synatx: <dataframe>.registerAsTable('<tablename>')

(2)
- Calculating Max using SQL query
  - Synatx: sqlContext.sql('select <groupbyvar>, max(<var>) from <tablename> group by <groupbyvar>').show()

(3)
- Calculating Median or Percentiles
  - sqlContext.sql('select percentile_approx(<var>,<n-tile>) from <tablename>').show()

Code:

1. Registering Dataframe as Table

```
df_amnt.registerAsTable('df_amnt_table')
```

2. Calculating Max using SQL query in Spark

```
sqlContext.sql('select SHIP_MODE, max(AMNT) from
df_amnt_table group by SHIP_MODE').show()
```

3. Calculating Median or Percentiles

```
sqlContext.sql('select
percentile_approx(AMNT,0.5) from
df_amnt_table').show()
```

Data Sampling: *from BIG DATA for analysis*

SAS equivalent: RANUNI, PROC SURVEYSELECT

Comments

(1)

- Random Sampling
  - Syntax:
    <dataframe_new>=<dataframe>.sample(<x>,<y>,<z>)
  - <x> -
    - With Replacement = True,
    - Without Replacement = False
  - <y> - Sampling Fraction
  - <z> - Seed

(2)

- Strategies Sampling

➢ Syntax:
<dataframe_new>=<dataframe>.sampleBy("<clasvar>",fractions={"<category1>":0.25,"<category2>":0.2},seed=26).show()

Code:

1. Random Sampling without replacement

```
➢ df_sample1=df_amnt.sample(False,0.2,26)
➢ df_sample2=df_amnt.sample(False,0.2,27)

➢ df_sample1.count(),df_sample2.count()
```

Random Sampling withreplacement

```
➢ df_sample1=df_amnt.sample(True,0.2,26)
➢ df_sample2=df_amnt.sample(True,0.2,27)

➢ df_sample1.count(),df_sample2.count()
```

2. Stratified Sampling

```
➢ df_stratified=df_amnt.sampleBy("SHIP_MODE",fractions={"First Class":0.25,"Standard Class":0.2},seed=26).show()
```

DISCOVER

Cheat Sheet : SAS v Python: Module II

Statistics	SAS	Python (Spark Dataframe)
Mean	PROC MEANS PROC UNIVARIATE PROC SQL	df.select([mean('colomn')])
Min	*PROC MEANS MIN ()*	df.select([max('column')])
Max	*PROC MEANS MAX ()*	df.select([mean('column')])
Stddev	PROC MEANS PROC UNIVARIATE	df.select([stddev('column')])
Distribution	*PROC FREQ* *PROC SQL*	df.describe()
Cross Tabulation	*PROC FREQ*	df.stat.crosstab("column1","column2")

Module III: **Data Modeling**

Welcome! Learn how to systematically apply statistical and/or logical techniques to describe and illustrate, your data.

- **Linear Regression**
- **Logistic Regression**
- **Decision Tree**

DISCOVER

Data Modeling & Advanced Analytics: *Packages – When to use Which?*

- **Scikit Learn written in python code** for machine learning. Built on NumPy, SciPy and matplotlib, this library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction.
  - **Single-machine setting**: When the dataset fits into the memory of a single machine and there is no scaling expected
  - **Spark Scikit integration** is available and adds great value when doing 'gridsearch i.e. (building models with different set of parameters'). It runs different models with combination of hyper paramet in parallel across eash Spark Executor but on the same machine.

- **Spark Mllib:** is Spark's machine learning (ML) library
  - **Distributed setting:** When the dataset sizes are large or you want to add more machines at a problem, MLlib is a preferable option since it is scales well for distributed computation.\

- **Spark ML:** refer to the MLlib DataFrame-based API, particularly designed for DataFrames

- The subsequent slides focusses on Data Modeling & Machine learning leveraging R, Sci-kit learn & Spark Scikit Integration

DISCOVER

Linear Regression: *to estimate predictors & predict continuous dependent*

SAS equivalent: PROC REG

R code:

```
#Load Train and Test datasets
#Identify feature and response variable(s) and values must be numeric and numpy arrays
x_train <- input_variables_values_training_datasets
y_train <- target_variables_values_training_datasets
x_test <- input_variables_values_test_datasets
x <- cbind(x_train,y_train)
# Train the model using the training sets and check score
linear <- lm(y_train ~ ., data = x)
summary(linear)
#Predict Output
predicted= predict(linear,x_test)
```

Python Code:

```
#Import Libraries numpy, pandas, scikit learn etc
from sklearn import linear_model
#Load Train and Test datasets #Identify feature and response variable(s) and values must be numeric and numpy arrays
x_train=input_variables_values_training_datasets
y_train=target_variables_values_training_dataset
x_test=input_variables_values_test_datasets
# Create linear regression object
linear = linear_model.LinearRegression()
# Train the model using the training sets and check score
linear.fit(x_train, y_train)
linear.score(x_train, y_train)
#Equation coefficient and Intercept
print('Coefficient: \n', linear.coef_)
print('Intercept: \n', linear.intercept_)
#Predict Output
predicted= linear.predict(x_test)
```

DISCOVER

Logistic Regression: : *to estimate predictors & predict discrete dependent*

SAS equivalent: PROC LOGISTIC

R code:

```
x <- cbind(x_train,y_train)
# Train the model using the training sets and check score
logistic <- glm(y_train ~ ., data = x,family='binomial')
summary(logistic)
#Predict Output
predicted= predict(logistic,x_test)
```

Python Code:

```
#Import Library
from sklearn.linear_model import LogisticRegression
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create logistic regression object
model = LogisticRegression()
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Equation coefficient and Intercept
print('Coefficient: \n', model.coef_)
print('Intercept: \n', model.intercept_)
#Predict Output
predicted= model.predict(x_test))
```

DISCOVER

Decision Tree: *Supervised learning algorithm for classification problems*

SAS equivalent: Decision Tree

R code:

```
library(rpart)
x <- cbind(x_train,y_train)
# grow tree
fit <- rpart(y_train ~ ., data =
  x,method="class")
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Python Code:

```
#Import Library  & libraries like pandas, numpy...
from sklearn import tree
#Assumed you have, X (predictor) and Y (target)
for training data set and x_test(predictor) of
test_dataset
# Create tree object
model =
  tree.DecisionTreeClassifier(criterion='gini')
# for classification, here you can change the
algorithm as gini or entropy (information gain) by
default it is gini
# model = tree.DecisionTreeRegressor() for
regression
# Train the model using the training sets and
check score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

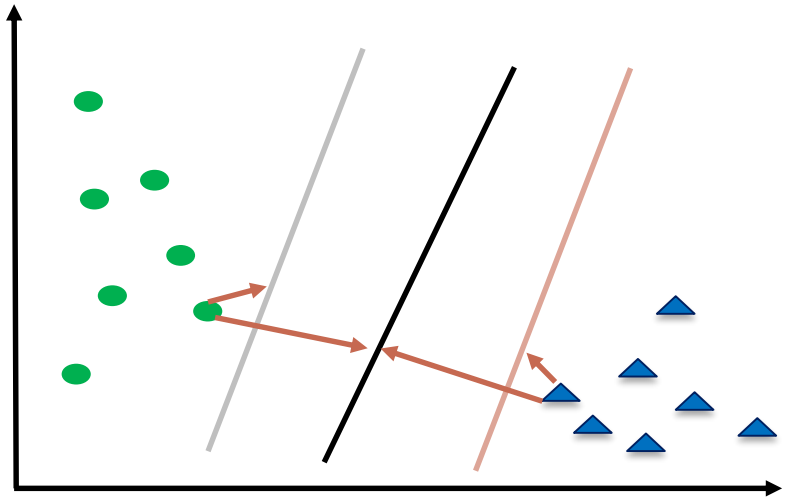
Module IV: **Advanced Analytics & Machine Learning**

Welcome! Learn how to systematically apply statistical and/or logical techniques to describe and illustrate, your data.

- **Support Vector Machine**
- **Naïve Bayes**
- **K-Nearest Neighbors**
- **Random Forests**
- **Gradient Boosting**
- **Principal Component Analysis**
- **K-Means**

Support Vector Machine(SVM): *Supervised learning algorithm for classification problems*

- For example, if we only had two features like Height and Hair length of an individual,
  - Plot these two variables in two dimensional space where each point has two co-ordinates
  - These co-ordinates are known as **Support Vectors**
  - find some line that splits the data between the two differently classified groups of data.



- In the example shown above,
  - the line which splits the data into two differently classified groups is the *black* line
    - since the two closest points are the farthest apart from the line.
    - This line is our classifier.
- Then, depending on where the testing data lands on either side of the line, that's what class we can classify the new data as.

DISCOVER

Support Vector Machine(SVM): *Supervised learning algorithm for classification problems*

SAS equivalent: PROC SVM

R code:

```
library(e1071)
x <- cbind(x_train,y_train)
# Fitting model
fit <-svm(y_train ~ ., data = x)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Python Code:

```
#Import Library
from sklearn import svm
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create SVM classification object
model = svm.svc()
# there is various option associated with it, this is simple for classification.
# Train the model using the training sets and check score
model.fit(X, y)
model.score(X, y)
#Predict Output
predicted= model.predict(x_test)
```

DISCOVER

Naïve Bayes: *Classification technique based on Bayesian Theorem for multiple classes*

- It is a classification technique based on Bayesian Theorem with an assumption of independence between predictors,
- In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature
- Naive Bayes is known to outperform even highly sophisticated classification methods.
    - P(c\x) = [P(x\c) * P(c)] / P(x)
        - *P*(*c*|*x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).
        - *P*(*c*) is the prior probability of *class*.
        - *P*(*x*|*c*) is the likelihood which is the probability of *predictor* given *class*.
        - *P*(*x*) is the prior probability of *predictor*.
- This algorithm is mostly used in text classification and with problems having multiple classes.

DISCOVER

Naïve Bayes: *Classification technique based on Bayesian Theorem for multiple classes*

SAS equivalent: E-Miner

R code:

```
library(e1071)
x <- cbind(x_train,y_train)
# Fitting model
fit <-naiveBayes(y_train ~ ., data = x)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Python Code:

```
#Import Library
from sklearn.naive_bayes import GaussianNB
#Assumed you have, X (predictor) and Y (target) for training data set and x_test(predictor) of test_dataset
# Create SVM classification object
model = GaussianNB()
# there is other distribution for multinomial classes like Bernoulli Naive Bayes,
# Train the model using the training sets and check score
model.fit(X, y)
#Predict Output predicted=
model.predict(x_test)
```

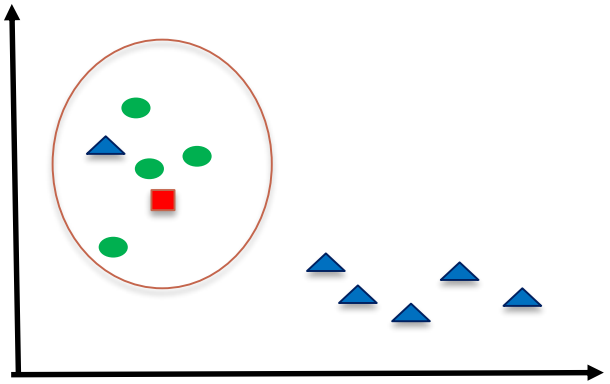
DISCOVER

K-Nearest Neighbor: *Pattern recognition algorithm for classification & regression*

- Can be used for both classification and regression problems
- K nearest neighbors is a simple algorithm that stores all available cases and classifies new cases by a majority vote of its k neighbors.
- The case being assigned to the class is most common amongst its K nearest neighbors measured by a distance function.
- These distance functions can be Euclidean, Manhattan, Minkowski and Hamming distance. First three functions are used for continuous function and fourth one (Hamming) for categorical variables.
- If K = 1, then the case is simply assigned to the class of its nearest neighbor.
- At times, choosing K turns out to be a challenge while performing KNN modeling.
- KNN is computationally expensive
- Variables should be normalized else higher range variables can bias it

For K=5, want to find which class ■ belongs:

- Majority vote belongs to ● i.e. 4/5
- With good degree of confidence, it can be concluded, ■ belongs to ●



DISCOVER

K-Nearest Neighbor: *Pattern recognition algorithm for classification & regression*

SAS equivalent: PROC DISCRIM METHOD=NPAR K=

R code:

```
library(knn)
x <- cbind(x_train,y_train)
# Fitting model
fit <-knn(y_train ~ ., data = x,k=5)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

Python Code:

```
#Import Library
from sklearn.neighbors import
KNeighborsClassifier
#Assumed you have, X (predictor) and Y (target)
for training data set and x_test(predictor) of
test_dataset
# Create KNeighbors classifier object model
KNeighborsClassifier(n_neighbors=6)
# default value for n_neighbors is 5
# Train the model using the training sets and
check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

DISCOVER

Random Forest : *Ensemble of decision tree to solve for classification problem*

- **Random Forest:**
  - Collection of decision tree known as “Forest”.
  - Samples are drawn at random, hence “Random Forest”
- **How Random Forest Works:**
  - Each tree is planted & grown as follows:
    - If the number of cases in the training set is N, then sample of N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
    - If there are M input variables,
      - a number m<<M is specified such that at each node,
      - m variables are selected at random out of the M and
      - the best split on these m is used to split the node.
      - The value of m is held constant during the forest growing.
    - Each tree is grown to the largest extent possible. There is no pruning

DISCOVER

Random Forest: *Ensemble of decision tree to solve for classification problem*

SAS equivalent: E-MINER version 6 or above

R code:

```
library(randomForest)
x <- cbind(x_train,y_train)
# Fitting model
fit <- randomForest(Species ~ ., x,ntree=500)
summary(fit)
#Predict Output
predicted= predict(fit,x_test)
```

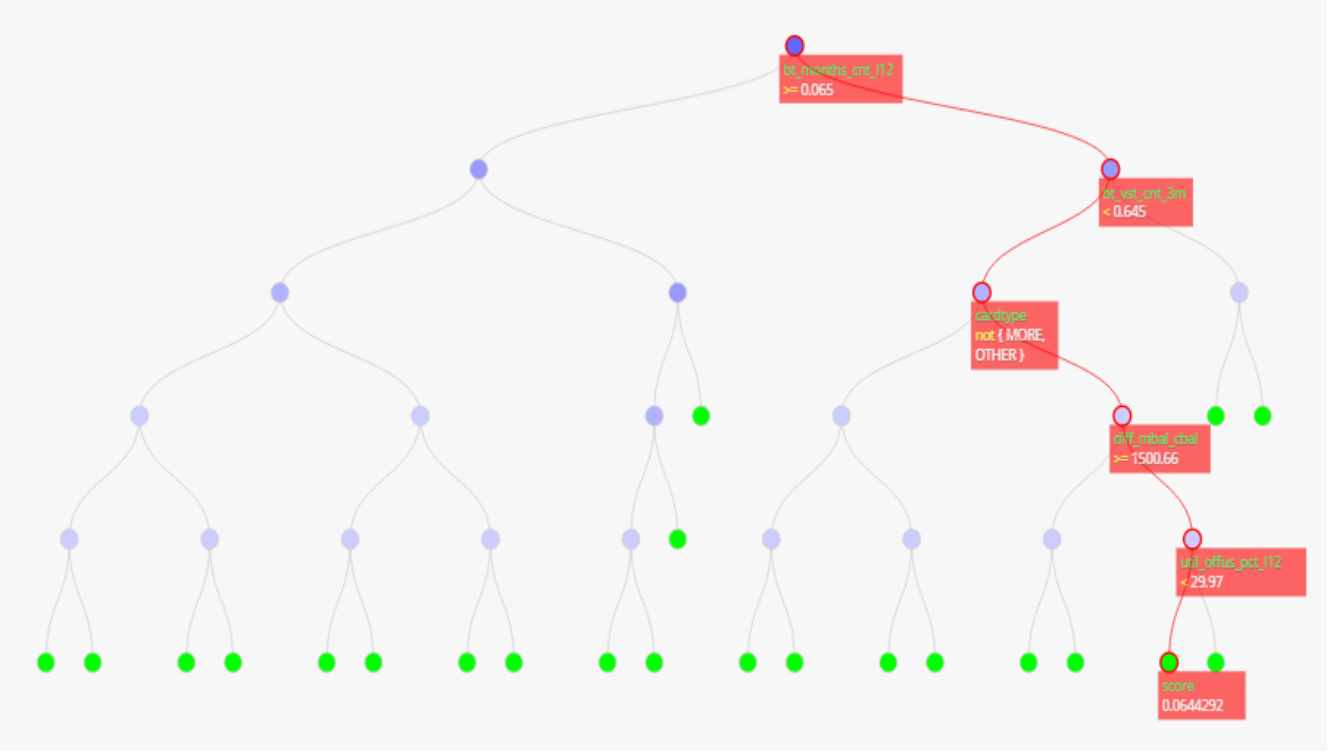
Python Code:

```
#Import Library
from sklearn.ensemble import
  RandomForestClassifier
#Assumed you have, X (predictor) and Y (target)
for training data set and x_test(predictor) of
test_dataset
# Create Random Forest object
model= RandomForestClassifier()
# Train the model using the training sets and
check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

Gradient Boosting: *Ensemble of decision tree to solve for classification problem*

How does Gradient Boosting Work:

- Ensemble of Trees similar to Random Forest. The trees starts with weak learner and grow as follows
  - The misclassified observations are assigned heavier weights in next iterations for greater focus
  - The adaptive learning continues building ensemble trees with focus on minimizing bias (Actual – Predicted)
  - The specified learning rate parameter can control rate of residual minimization. Slow v Fast learners



Gradient Boosting: *Grid Search to find best model and minimize risk of overfitting*

Gradient Boosting:

- When we train the GBT model, we evaluate error which is function of Bias and Variance
- Gradient Boosting primary aim is to minimize Bias i.e. Actual – Predicted)
- This can potentially cause model elasticity and increase risk of overfitting
- To minimize the risk of overfitting, GBT model provides multiple parameters that can be actioned by grid search
- Gridsearch helps to test multiple model & find best model

How does Grid Search in Gradient Boosting Work:

- **Learning Rate:** The rate at which error is minimized. Low rate will slow down the rate of learning & increase number of trees & vice-versa. Ideal Range: 5% to 20%
- **Tree Depth**: number of tree layers. Ex: 2,3,4,5 etc
- **Minimum node weight:** minimum records in each leaf node
- **Model Validation: K-fold Cross validation.** The data set is divided into k subsets, and the holdout method is repeated k times. Each time, one of the k subsets is used as the test set and the other k-1 subsets are put together to form a training set

Challenges:

- Computationally intensive
- Recommend to leverage Spark-Scikit learn integration

DISCOVER

Gradient Boosting: *Ensemble of decision tree to solve for classification problem*

SAS equivalent: E-MINER

R code:

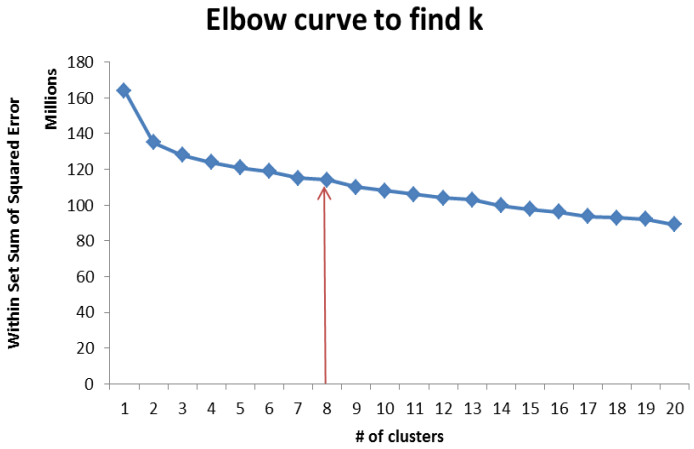
```
library(caret)
x <- cbind(x_train,y_train)
# Fitting model
fitControl <- trainControl( method =
"repeatedcv", number = 4, repeats = 4)
fit <- train(y ~ ., data = x, method = "gbm",
trControl = fitControl,verbose = FALSE)
predicted= predict(fit,x_test,type= "prob")[,2]
```

Python Code:

```
#Import Library
from sklearn.ensemble import
GradientBoostingClassifier
#Assumed you have, X (predictor) and Y (target)
for training data set and x_test(predictor) of
test_dataset
# Create Gradient Boosting Classifier object
model=
GradientBoostingClassifier(n_estimators=100,
learning_rate=1.0, max_depth=1, random_state=0)
# Train the model using the training sets and
check score
model.fit(X, y)
#Predict Output
predicted= model.predict(x_test)
```

K-Means: *Unsupervised learning to solver clustering problem with no dependent variable*

- **How K-Means forms cluster**
  - K-means picks k number of points for each cluster known as centroids.
  - Each data point forms a cluster with the closest centroids i.e. k clusters.
  - Finds the centroid of each cluster based on existing cluster members. Here we have new centroids.
  - As we have new centroids, repeat step 2 and 3.
  - Find the closest distance for each data point from new centroids and get associated with new k-clusters.
  - Repeat this process until convergence occurs i.e. centroids does not change.



- **How to determine value of K:**
  - In K-means, we have clusters and each cluster has its own centroid.
  - **Within SS:** Sum of square of difference between centroid and the data points within a cluster
  - **Total SS:** Sum of square values for all the clusters are added,
  - **Between SS**: Total SS – Within SS
  - As the number of cluster increases, the sum of squared distance decreases sharply up to some value of k, and then flattens. That point suggest adding more clusters don't add value in explaining the population

DISCOVER

K-Means: *Unsupervised learning to solver clustering problem with no dependent variable*

SAS equivalent: PROC FASTCLUS, E-MINER

R code:

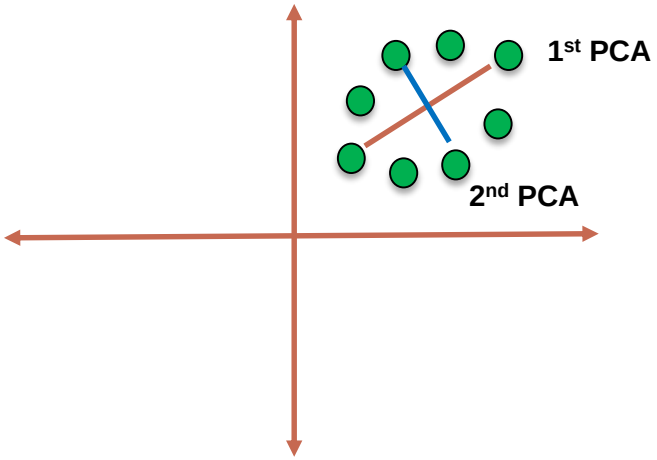
```
library(cluster)
fit <- kmeans(X, 3)
# 5 cluster solution
```

Python Code:

```
#Import Library
from sklearn.cluster import KMeans
#Assumed you have, X (attributes) for training
data set and x_test(attributes) of test_dataset
# Create KNeighbors classifier object model
k_means = KMeans(n_clusters=3, random_state=0)
# Train the model using the training sets and
check score
model.fit(X)
#Predict Output
predicted= model.predict(x_test)
```

Principal Component Analysis: *for Dimensionality/Variable Reduction*

- Variables are transformed into a new set of variables,
- They are linear combination of original variables.
- These new set of variables are known as **principle components.**
- First principal component should explain highest variation and each succeeding component has the highest possible variance thereafter in monotonically decreasing way
- The second principal component must be orthogonal to the first principal component. I
  - In other words, it does its best to capture the variance in the data that is not captured by the first principal component.
  - For two-dimensional dataset, there can be only two principal components.



- Graph is a snapshot of the data and its first and second principal components.
- Notice that second principle component is orthogonal to first principle component.

DISCOVER

Principal Component Analysis: *for Dimensionality/Variable Reduction*

SAS equivalent: PROC PRINCOMP

R code:

```
library(stats)
pca <- princomp(train, cor = TRUE)
train_reduced <- predict(pca,train)
test_reduced <- predict(pca,test)
```

Python Code:

```
#Import Library
from sklearn import decomposition
#Assumed you have training and test data set as
train and test
# Create PCA obeject
pca= decomposition.PCA(n_components=k)
#default value of k =min(n_sample, n_features)
# For Factor analysis
#fa= decomposition.FactorAnalysis()
# Reduced the dimension of training dataset using
PCA
train_reduced = pca.fit_transform(train)
#Reduced the dimension of test dataset
test_reduced = pca.transform(test)
```

DISCOVER

References:

- Python Basics and interactive course
  https://www.codecademy.com/learn/python
- Hadoop basics & Architecure
  https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
- Spark Programing guide
  http://spark.apache.org/docs/1.6.1/index.html
- AnalyticsVidhya
- DataBricks
- GitHub