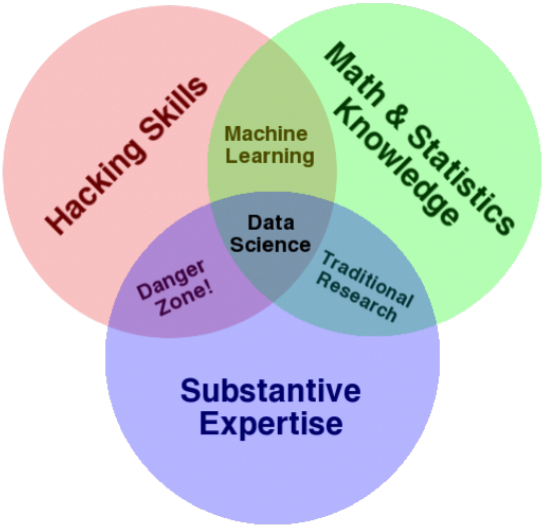


COMM054: Data Science Principles & Practices

UNIVERSITY OF SURREY

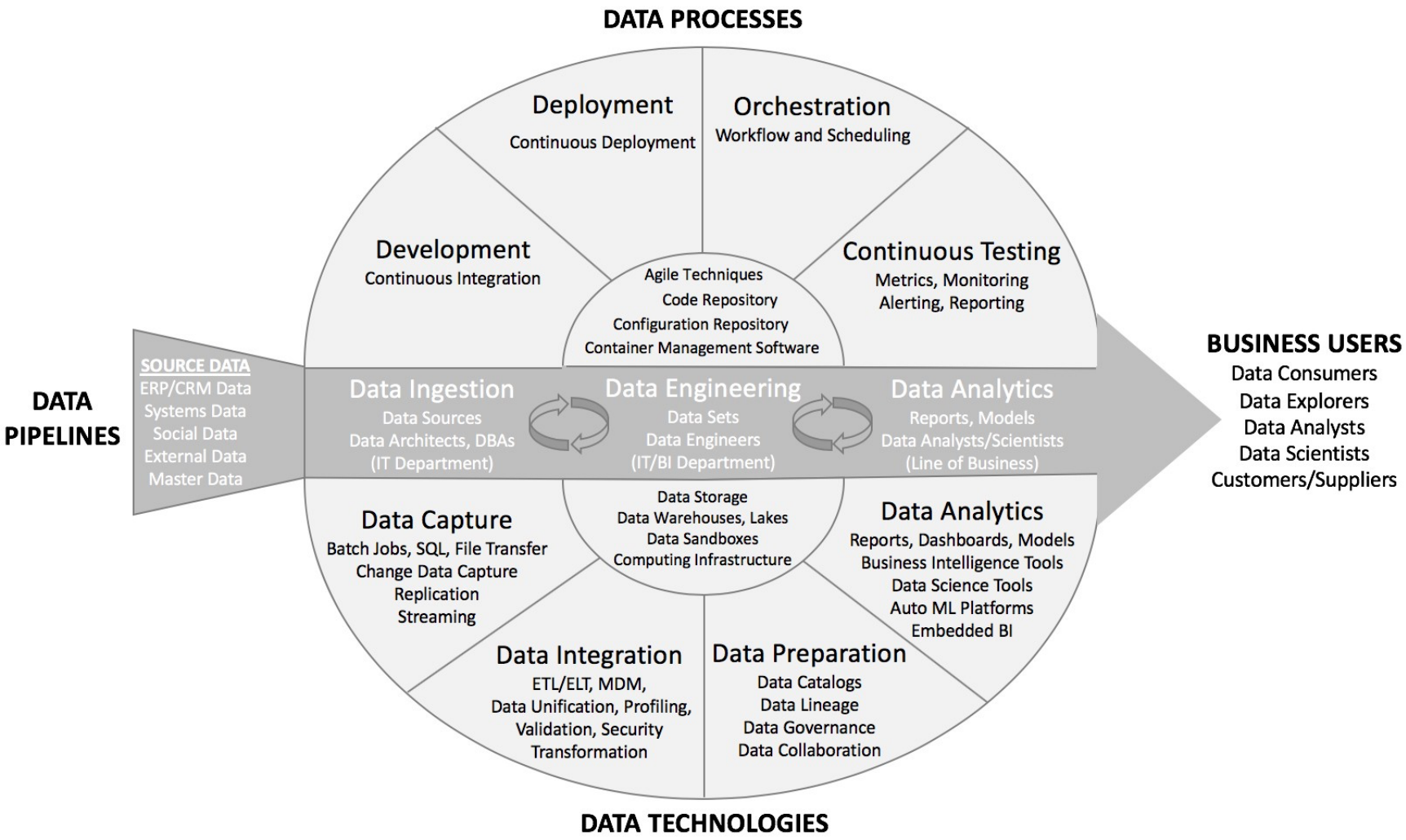
Introduction to Data Munging and Big Data

Dr. Manal Helal

17 October 2019

Outline

- Data Munging
  - Pandas
- Big Data
- PySpark


DATA PROCESSES
Deployment
Continuous Deployment
Orchestration
Workflow and Scheduling
Development
Continuous Integration
Continuous Testing
Metrics, Monitoring
Alerting, Reporting
Agile Techniques
Code Repository
Configuration Repository
Container Management Software
DATA PIPELINES
SOURCE DATA
ERP/CRM Data
Systems Data
Social Data
External Data
Master Data
Data Ingestion
Data Sources
Data Architects, DBAs
(IT Department)
Data Engineering
Data Sets
Data Engineers
(IT/BI Department)
Data Analytics
Reports, Models
Data Analysts/Scientists
(Line of Business)
BUSINESS USERS
Data Consumers
Data Explorers
Data Analysts
Data Scientists
Customers/Suppliers
Data Storage
Data Warehouses, Lakes
Data Sandboxes
Computing Infrastructure
Data Capture
Batch Jobs, SQL, File Transfer
Change Data Capture
Replication
Streaming
Data Analytics
Reports, Dashboards, Models
Business Intelligence Tools
Data Science Tools
Auto ML Platforms
Embedded BI
Data Integration
ETL/ELT, MDM,
Data Unification, Profiling,
Validation, Security
Transformation
Data Preparation
Data Catalogs
Data Lineage
Data Governance
Data Collaboration
DATA TECHNOLOGIES

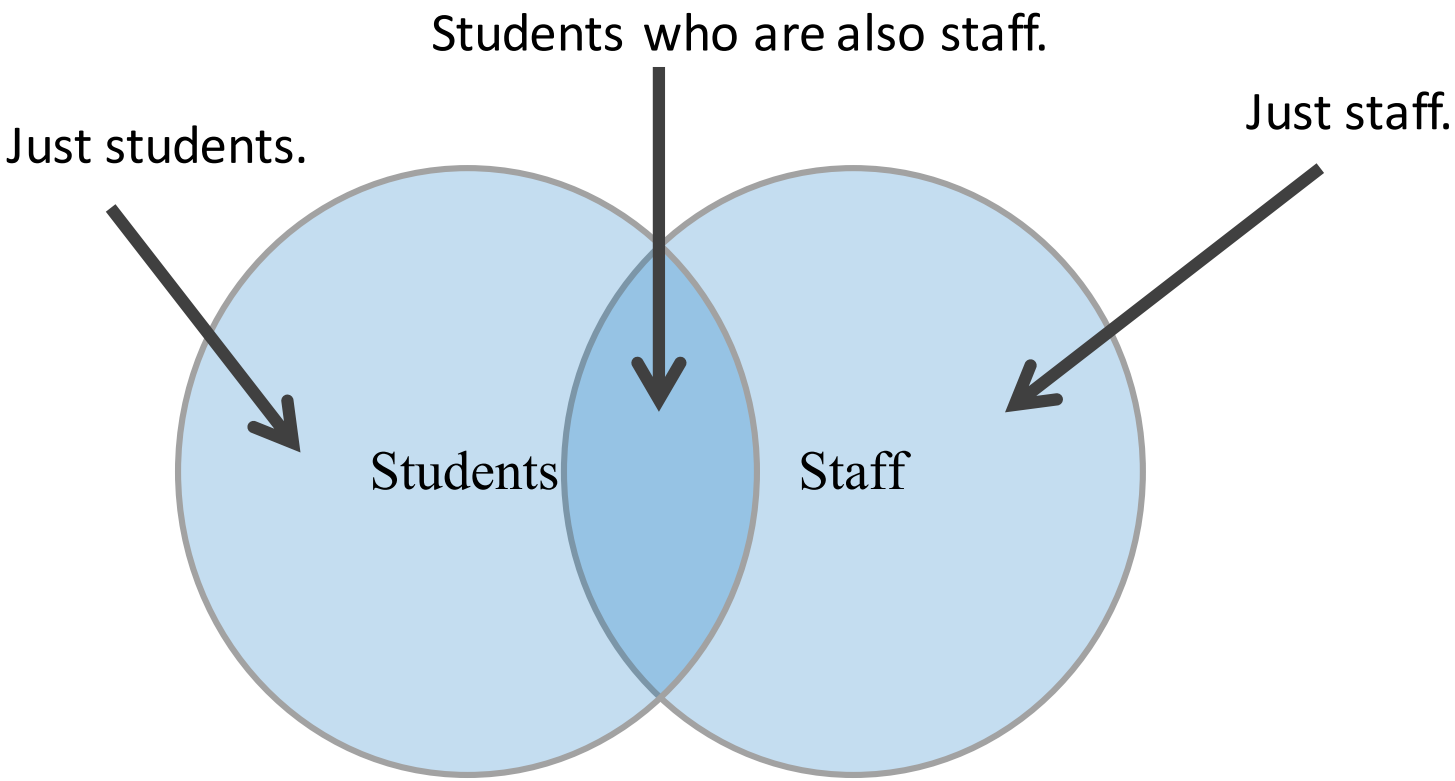
Pandas Data Structures

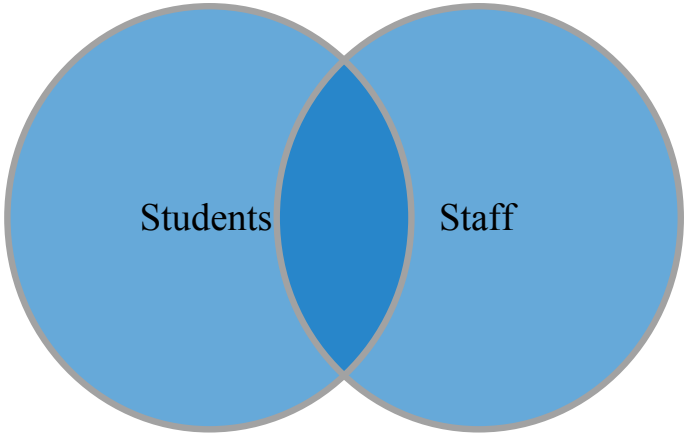
- From last week:
  - It is a data manipulation and analysis library.
  - There is really two-core data structures that are very similar:
    - **Series Object (1 dimensional, a row)**
    - **DataFrame Object (2 dimensional, a table)**
  - Querying
    - *iloc[], for querying based on position*
    - *loc[], for querying rows based on label*
    - *Querying the Data Frame directly*
      - *Projecting a subset of columns*
      - *Using a Boolean mask to filter data*

Setting Data in Pandas

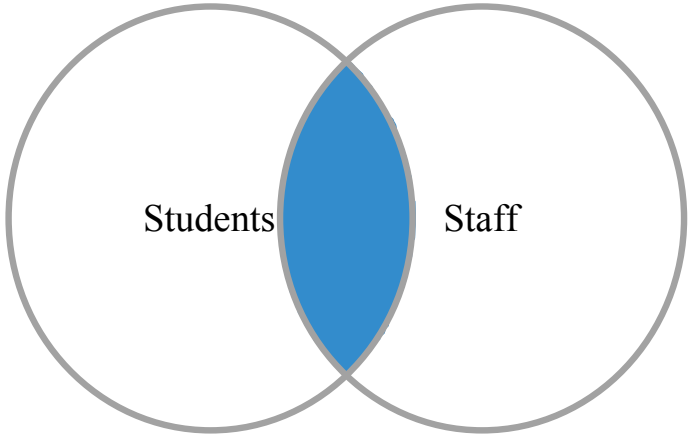
- To add new Data: as long as the index is shared,
  - df[column]=[a,b,c]
  - to assign a different value for every row, hardcode the values into a list, then pandas will unpack them and assign them to the rows as long as the list of equal size as the rows.
- To set default data (or overwrite all data):
  - df[column]=2
- Otherwise: give each rows a unique index, and assign the new column identifier to the series and pandas will put missing values in for us.

Venn Diagram

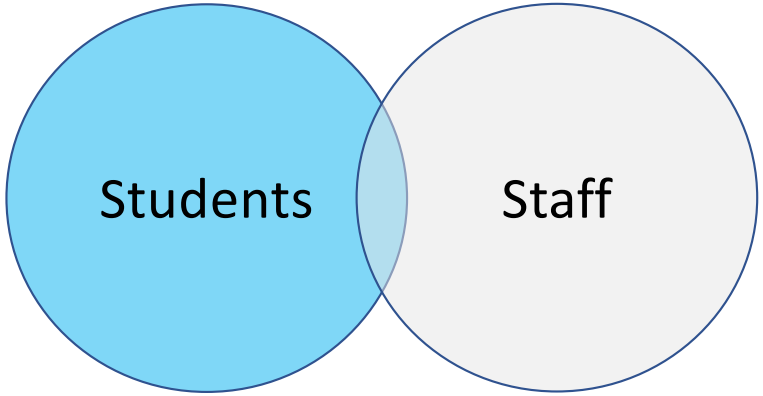




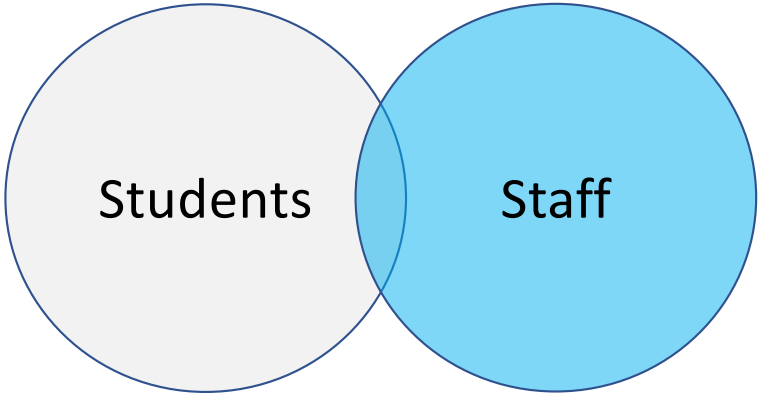
Full outer join (union)



Inner join (intersection)



Left join (All Students and only matching staff records)



Right join (All Staff and only matching students records)

Method Chaining

- Bad Practice:
  - Chain Indexing:
    - *df.loc[“Washtenaw”][“TotalPopulation”]*
    - *Generally bad, pandas could return a copy of a view depending upon numpy*
  - Code smell
    - *If you see a][you should think carefully about what you are doing (Tom Augspurger)*
- Method Chaining is a good practice, as methods on an object return a reference to that object.

(a, b) (c, d): Scales

- **Ratio scale:**
  - ***units are equally spaced***
  - ***mathematical operations of +-/* are all valid***
  - ***E.g. height and weight***
- **Interval scale:**
  - ***units are equally spaced, but there is no true zero***
- **Ordinal scale:**
  - ***the order of the units is important, but not evenly spaced.***
  - ***Letter grades such as A+, A are a good example***
- **Nominal scale:**
  - ***categories of data, but the categories have no order with respect to one another.***
  - ***E.g.Teamsofasport.***

Other interesting Pandas Functions

- Apply:
  - Functional programming languages had a basic function called **map**. it takes a function and an iterable as parameters, like a list, that you want the function to be applied to. The results are that the function is called against each item in the list, and there's a resulting list of all of the evaluations of that function.
  - Python has a similar function called **applymap**. In applymap, you provide some function which should operate on each **cell** of a DataFrame, and the return set is itself a DataFrame. There is also **apply** function that applies a parameter function across all of the **rows** in a DataFrame.
- GroupBy
  - Group By takes some column name or names and splits the dataframe up into chunks based on those names.
  - It returns a dataframe group by object, Which can be iterated upon, and then returns a tuple where the first item is the group condition, and the second item is the data frame reduced by that grouping.
  - Since it's made up of two values, you can unpack this, and project just the column that you're interested in, to calculate the average.

What Defines Big Data

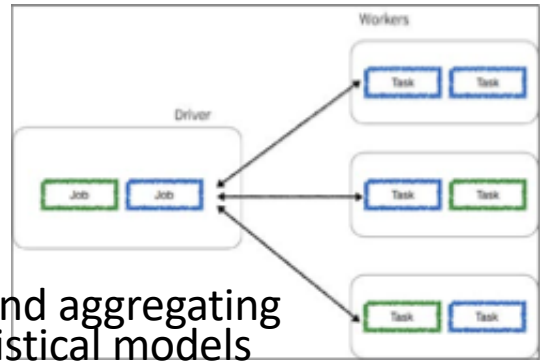
Big Data Characteristics

- Volume
  - The quantity of generated and stored data. The size of the data determines the value and potential insight, and whether it can be considered big data or not.
- Variety
  - The type and nature of the data. This helps people who analyze it to effectively use the resulting insight. Big data draws from text, images, audio, video; plus it completes missing pieces through data fusion.
- Velocity
  - Big data is often available in real-time. Two kinds of velocity related to big data are the frequency of generation and the frequency of handling, recording, and publishing.
- Veracity
  - The data quality can vary greatly, affecting the accurate analysis.
- Other Characteristics:
  - Exhaustive, Fine-grained and uniquely lexical, Relational, Extensional, Scaleability, Value, Variability

Big Data Volume

- In 2013 the whole world produced around 4.4 zettabytes of data; that is, 4.4 billion terabytes!
- By 2020, we (as the human race) are expected to produce ten times that.
- In 2004 Google published the paper “MapReduce: Simplified Data Processing on Large Clusters”, leading to the Hadoop ecosystem including abstraction layers such as Pig, Hive, and Mahout.
  - Main drawback: reading and writing to disk.
- In 2012, Spark addressed this by in-memory computations making it 100x faster than Hadoop (for in-memory computations), or 10x faster on disc.

Apache Spark



- Open-source powerful distributed platform to read, transform, querying and aggregating data, and processing engine, as well as train and deploy sophisticated statistical models with ease.
- Offers APIs accessible from Java, Scala, Python, R and SQL.
  - Python's pandas or R's data.frames or data.tables.
- Build applications or package them up as libraries to be deployed locally, standalone mode, YARN, Mesos, Kubernetes, Nomad or perform quick analytics interactively through notebooks.
- Contains several already implemented and tuned algorithms, statistical models, and frameworks:
  - MLlib: ML for machine learning,
  - GraphX and GraphFrames for graph processing,
  - and Spark Streaming (DStreams and Structured).
- Various Data Sources: HDFS, Apache Cassandra, Apache HBase, and S3.
- Job is defined as a DAG (Direct Acyclic Graphs) and is divided into tasks performed by workers.

RDD

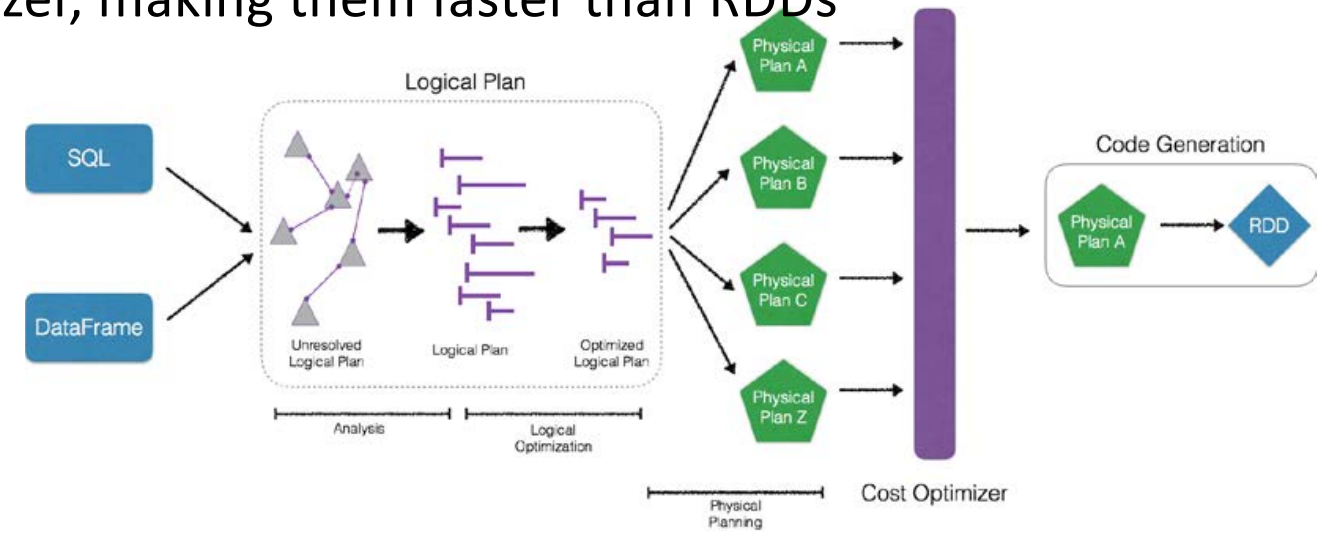
- Resilient Distributed Datasets (RDDs) are schema-less data structures distributed collection of immutable Java Virtual Machine (JVM) objects that are calculated against, cached and stored in memory.
- RDDs expose actions such as count, collect. Actions return the values from the dataset.
- RDDs expose some coarse-grained transformations (such as map(...), reduce(...), and filter(...) . Spark apply transformations to the data in parallel, resulting in both increased speed and fault-tolerance. Transformations return another RDD.
- Spark registers transformations, which provides data lineage. This creates an ancestry tree for each intermediate step in the form of a graph, which enables workers to recreate their partitions of data rather than depending on replication in case of data loss. http://ibm.co/2ao9B1t.

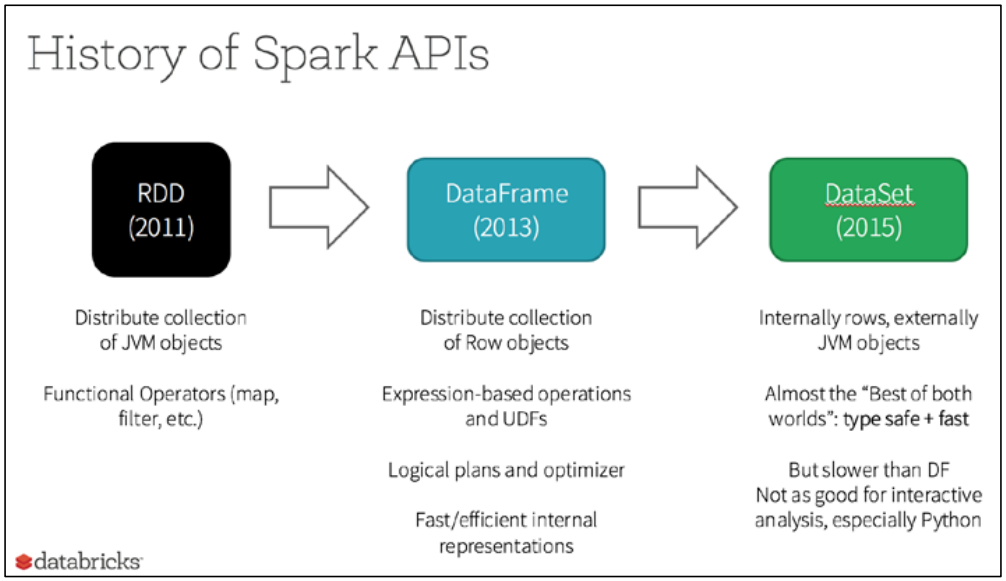
For the latest list of transformations and actions, please refer to the Spark Programming Guide at `http://spark.apache.org/docs/latest/programming-guide.html#rdd-operations`.

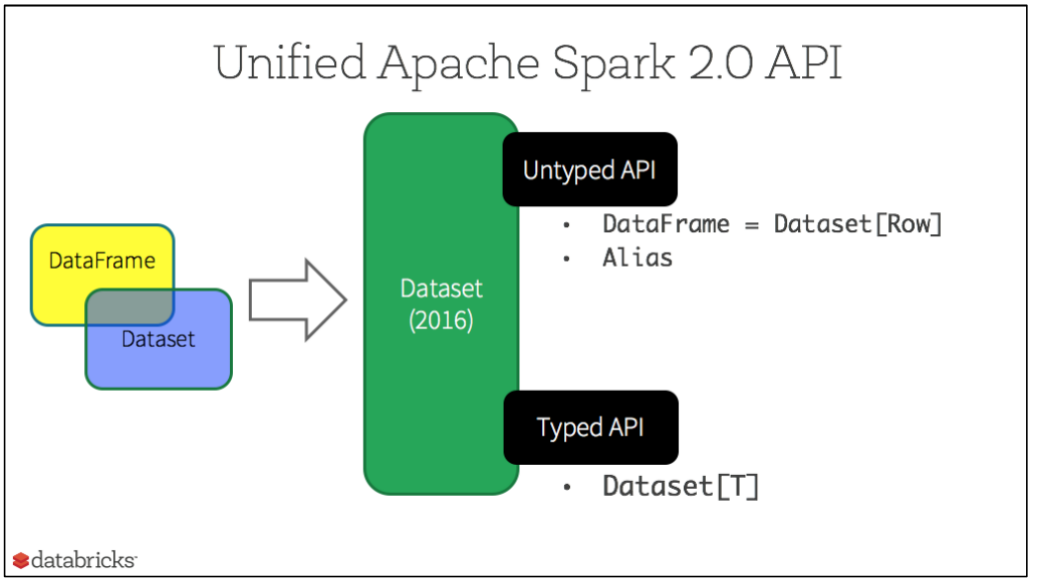
DataFrames

- DataFrames, like RDDs, are immutable collections of data distributed among the nodes in a cluster. However, unlike RDDs, in DataFrames data is organized into named columns like tables in RDBMs.
- DataFrames' major benefit is that the Spark engine initially builds a logical execution plan and executes generated code based on a physical plan determined by a cost optimizer, making them faster than RDDs

For more information, check out *Deep Dive into Spark SQL's Catalyst Optimizer* (`http://bit.ly/271I7Dk`) and *Apache Spark DataFrames: Simple and Fast Analysis of Structured Data* (`http://bit.ly/29QbcOV`)







Source: From Webinar Apache Spark 1.5: What is the difference between a DataFrame and a RDD?
http://bit.ly/29JPJSA

Source: A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets
http://bit.ly/2accSNA

Spark Sessions

- SparkConf, SparkContext, SQLContext, and HiveContext are what you need to execute your various Spark queries for configuration, Spark context, SQL context, and Hive context respectively.
- The SparkSession is essentially the combination of these contexts including StreamingContext.