5. Notes Textbook and papers

Textbook – Spark – Ch.1

Spark is a unified computing enginge and a set of libraries for parallel data processing on computer clusters. Spark is the most actively developed **open source program** for this task.

It is used in a variety of languages: Python, Java, Scala, R and SQL being the most widely used. And has libraries for diverse tasks.

In a nutshell spark's toolkit is:

Structured streaming	Advanced analytics	Libraries and ecosystems
Structured APIS Data sets, Dataframes and SQL		
Low-level APIs RDDs Distributed Variables		

Spark is:

- **Unified**: to offer a single platform for writing big data application → supports a wide range of tasksProvides consistent APIs used to build an application out of existing libraries
  - Is designed to enable high performace by optimizing across the different libraries and functions of the user program
- **Computing engine:** it is limited to the task of computing: spark handles loading data from storage and performing computationon it.
  - As data is expensive to move, Spark focuses on performing computation over the data, no matter where it resides.
- **Libraries:** the final component is its libraries which build on its design as a unified engine to provide a unified API for common data analysis tasks and tools
  - Spark libraries are actually the bulk of the open source project → are the part of Spark that grown the most since its launch

Spark was born as a computing cluster alternative to the problems that many researchers at Berkely UC in 2009 observed in the Hadoop environment: it helped resolve the problem of inefficient algorithms such as machine leraning algorithms where the data has to be passed over multiple times and in each pass a mapreduce job has to be scheduled on a different machine.

The early version of Spark define d the composable APIs of Spark in terms of functional operations → afterwards, with Spark SQL, a new API for working with strcutred data arrived

Data Frame

To create a *DataFrame* with one columns containing 1000 rows with values between 0 - 999

```
//In Scala
val myRange = spark.range(1000).toDF("number")

//In Python
myRange = spark.range(1000).toDF("number")
```

5.1 Lecture Notes

Outcomes

1. Methodology for Data science is crucial
2. There are multiple different types of pitfalls
3. One has to apply the pricnciples of methodology in its own research
4. Analyze and recognize potential traps

Use Bloom's Taxonomy [scheme of classification]

1. Remember
2. Understand
3. Apply
4. Analyze
5. Evaluate
6. Create

Hull Hypothesis Significance Testing

1. Formulate H0 and Ha
2. Using sampling distribution of the appropriate test statistics determie critical region of size alpha
3. Determine the value of the test statistics from the sample data
4. Check if it falls in the critica region (reject H0) or outside (reject Ha)

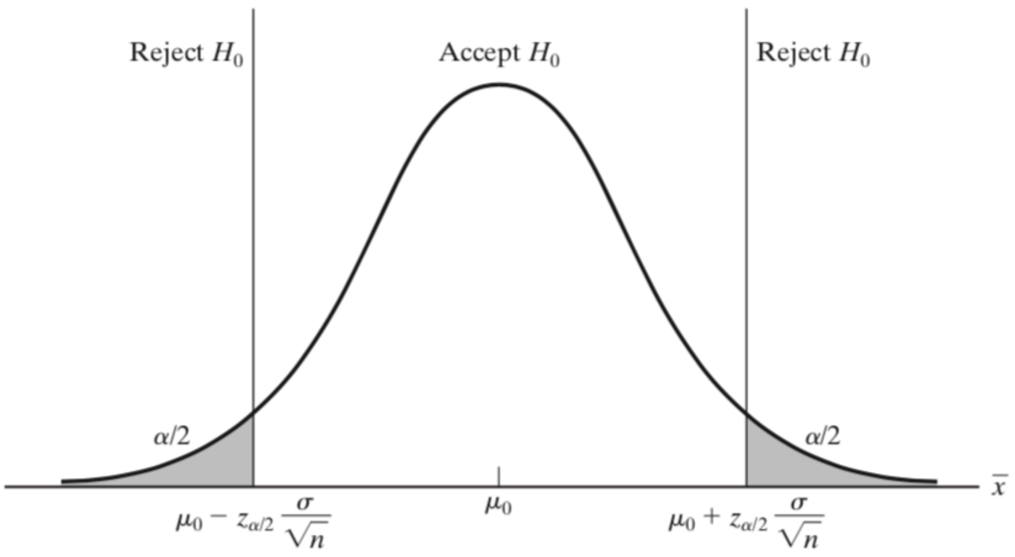


Figure 2: Critical region for two-tailed test (I Miller and M Miller, 2014, p. 360).

P Value

Depends on who one asks to:

- Frequentists: limiting relative frequency, if you could repeat the experiment
- Baysian: subjective, degree of belief, personally defined

if p<alpha, the observed data is inconsistent with null hypothesis —> it must be rejected.

This does not prove that the tested hypothesis is true, only that Type I error (Fals positive) is at most alpha

Nowadays there are academic who are proposing to lower the p value to 0.005 for new discoveries.

Problem of multiple testing

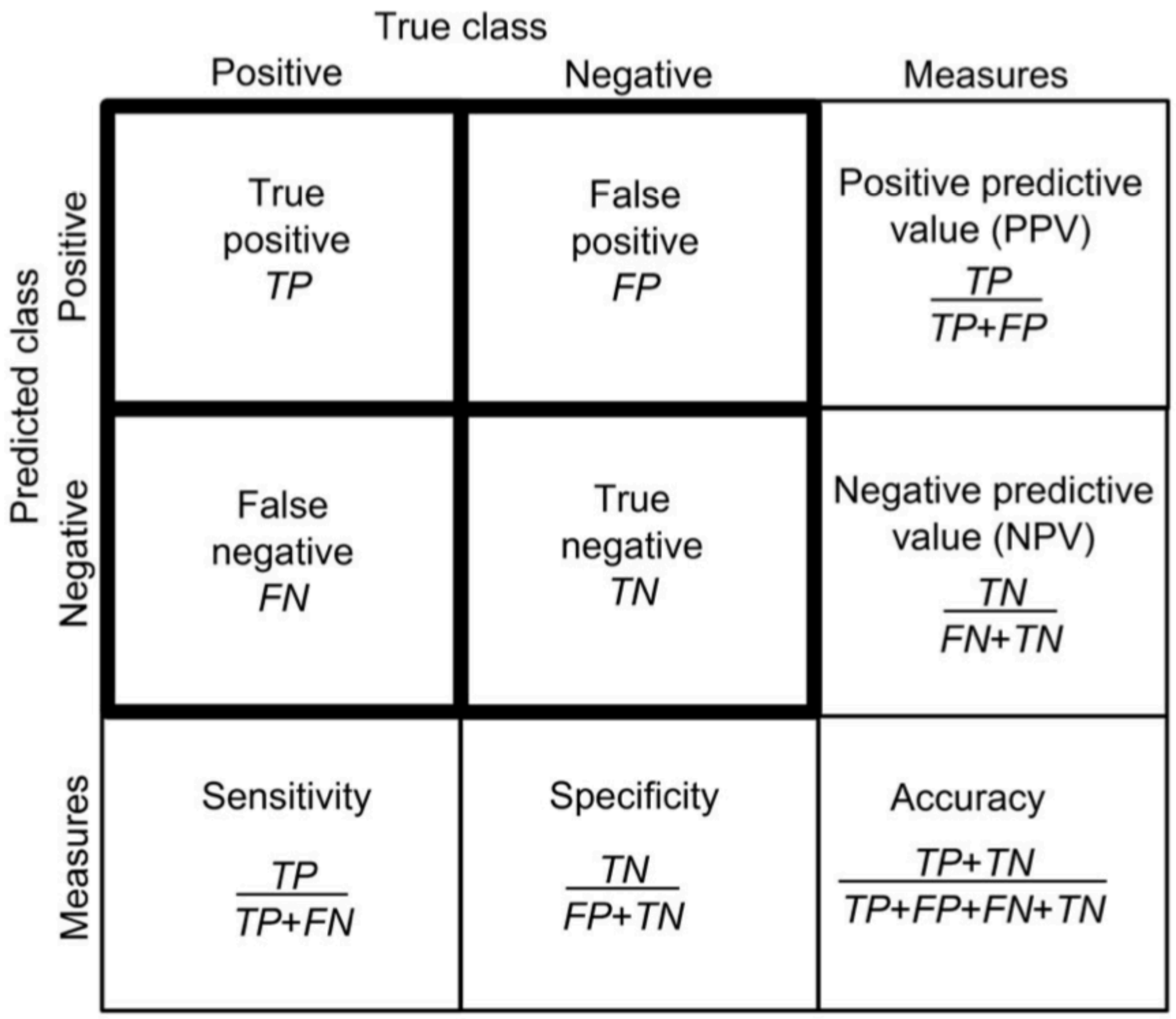
"When pursuing multiple inferences, researchers tend to select the (statistically) significant ones for emphasis, discussion and support for conclusions. An unguarded use of single-inference procedures results in a greatly increased false positive (significance) rate" possible to correct using `p.adjust{stat}`

Prediction

There is a trade-off between prediciton accuracy and interpretability of the model

**Validation set Approach: **split your data set in two parts, a training and an validation/ evaluation set.

Measure the quality of the fit: using mean squared error or error rate



		True class		
		Positive	Negative	Measures
Predicted class	Positive	True positive *TP*	False positive *FP*	Positive predictive value (PPV) $\frac{TP}{TP+FP}$
	Negative	False negative *FN*	True negative *TN*	Negative predictive value (NPV) $\frac{TN}{FN+TN}$
	Measures	Sensitivity $\frac{TP}{TP+FN}$	Specificity $\frac{TN}{FP+TN}$	Accuracy $\frac{TP+TN}{TP+FP+FN+TN}$

Beware of casual statements!

Graphs are essential to any statistical analysis: use anscombe quartet `example(anscombe)`

Get a grip on the quality of the data

- Garbage in-garbage out principle
- Some measure: completeness, validity, accuracy, consistency, availablity, timeliness
- **Beware of missing data: **

Paper - The Parable of Google Flu

The Google Flu Trend (GFT) made headlines because it was predicting more than the double the proportion of doctor vistis for inlfuenza-like illness thatn the center for disease control and prevention. Why, given it was using Big Data?

Quantity of data does not mean that one can ignore the issues of measurement, construct validity and reilabiltiy and dependecies among data. It seems that most big data is seen as an instrument designed to produce valid and reliable prediction no matter the quality of data it uses. **not true.**

The GFT tried to find the best mathces among 50M data points and search terms to fit 1152 data points —> the odds of finding correlated but unrealeated fits were high. The GFT missed on the seasonal influenza and many other ILI occurrences.

The real question is: is GFT useless? No, it can provide helpfu real-time data on influenza activity. Adn by combining GFT and the CDC data, dynamically recalibrating GFT one can improve substantially on the prediction it formulates.

The big problems are/were:

1. The GFT uses google search terms and the algorithms is continually updated
2. Data of GFT are not discoled and one does not know in which way they could correlate
3. There could be a tipical “blue team” dynamics in which Google’s search team influences GFT data due to modification —> reccomended searches, base on what other seraches, will increase the magnitude of certain searches. Beacuse GFT uses the relative prevalence of search terms in its model, improvements in the serach algorithm can adversely affact GFT estimates
4. In this regard, GFT assumes that relative search volume for certain terms is statically related to external events, but search behavior is not just exogenously determined, it is also modifed and cultivated by the behavior of GOogle itself!
5. Scholars should also pay attention to “red team” dynamics —>search and use data or a data-generating process that will give you the results you are searching for

We can learn critical lessons:

- **Transparency and replicability**: is a criical issue as many materias do not meet community standards (also in the case of GFT). It is also impossible to have the same data as Google has.
    - Dangerous for the “standing on the shoulder of giants” effect that research should have —> here not present
- **Big Data to understand Unknown**: does not always work. But it could help, working togther with the previous methods of doing research and forecasts, to bridge the last gaps standing
- **Small data**: offers information that is not contaned in big data and is fundamental not to focus of a big data revolution, rather on a "all data revolution"

Answer to paper

Criticism

Concerns specific to GFT should not be generalized to other big data analyses (see Twitter and influenza)

Response

Data can be manipulated also in these services

Who uses these services and how to use them change continiously

Possible to use them in future, but need to recalibrate methodologies, replicability of results and evalutaion of error and process. (a nice fuck you)

Cardiometabolic Diseases

Basic understanding and assignment

Cardiometabolic are a combination of different diseases:

1. Cardiovascular disease

- CAD or CHD -> leading cause of death in the world (1/3 of all global deaths - 17.7M people)
- Atheroschlerosis: plaque builds on vessel wall and slowly occludes completely the vessels --> provokess a ischemic or hemorrahgic stroke

2. Type 2 diabetes

Perimenopause - transition from menstruation to "unfertility". End is defined as having a year-break from the last cycle

Association between menopause and cardiometabobilic diseases

The rate of diseases of men to women/age inverts at around 60y. First strong increase in men, then around age of 50 women start to get more diseases compared to men.

In 1990s van der Schouw YT, showed that for each year one goes **later** to menopause, the risk of cardiovascular diseases descreases 2%.

Menopause before cardiovascular diseases?

- Smoking accelerates menopause
- LDL cholesterol is correlated with menopause
- Estrogens level: **no association** between those and cardiovascular diseases

Causality is not determined/sure

Genetics

46 chromosomes, that contains DNA that has 4 bases (A,T,G,C). They cannot randomly merged together. C-G, A-T. Our DNA has ca. 3 Billions letters --> we are identical for ca. 99.99% --> our differences are around 1.4M. During DNA replication DNA creates copy mistakes, lukily we have a good repair system

Linkage disequilibrium (LD) -> nearby variats on the chromosome** are correlated**

-> understanding the genetics of a disease may help understand the mechanism that cause them

Sources of information

- SNPs are only one layer of variation in the genome

> We can use these information to find genetic correlation between traits

Mendelian Randomization: look at `mrbase.biocompute.org.uk`

Assignment

Find out the link between the age of menopause and the cardiovascular disease risk

5.2 Laboratory Notes (4)

Beginning

Install and access the packages

```
install.packages("tidyverse") // which is a collection of R packages for data science
install.packages("nycflights13")
install.packages("sparklyr")
library(tidyverse)
library(nycflights13)
```

Summarization and transformation

Groups data {flights} by destination and creat a summary by {destination} showing planes with distinct tailnum by `n_distinc(tailnum)`

```
destinations <- group_by(flights, dest)
```

Use `group_by()` to create a "grouped" copy of a table. dplyr functions will manipulate each "group" separately and then combine the results.
`group_by(.data, ..., add = FALSE)`
Returns copy of table grouped by "…"

```
summarise(destinations,
  planes = n_distinct(tailnum),
  flights = n()
)
```

These apply summary functions to columns to create a new table of summary statistics.
Summary functions take vectors as input and return one value
`summarise(.data, …)`
`Compute table of summaries. summarise(mtcars, avg = mean(mpg))`

Some variations:

```
summarise_all() - Apply funs to every column.
summarise_at() - Apply funs to specific columns.
summarise_if() - Apply funs to all cols of one type.
```

`N_distinct` here efficiently counts the number of unique values in a set of vector, better version than `length(unique(x))`

`Select(.data, …)` selects column as a table

Pipes

We can use these functions as Grammar using pipes `%>%` . This is the same function as before but using pipes

```
group_by(flights, dest) %>%
  summarise(planes = n_distinct(tailnum),
            flights = n()) %>%
  mutate(fl_per_plane = flights / planes)
```

Here the function `mutate` computes new columns: `mutate(mtcars, gpm = 1/mpg)`

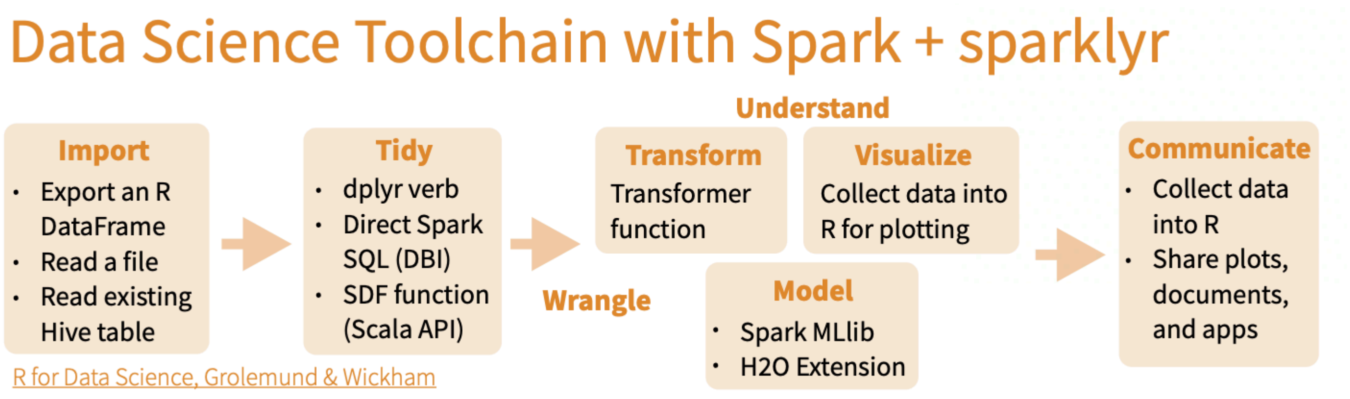
Map reduce in spark

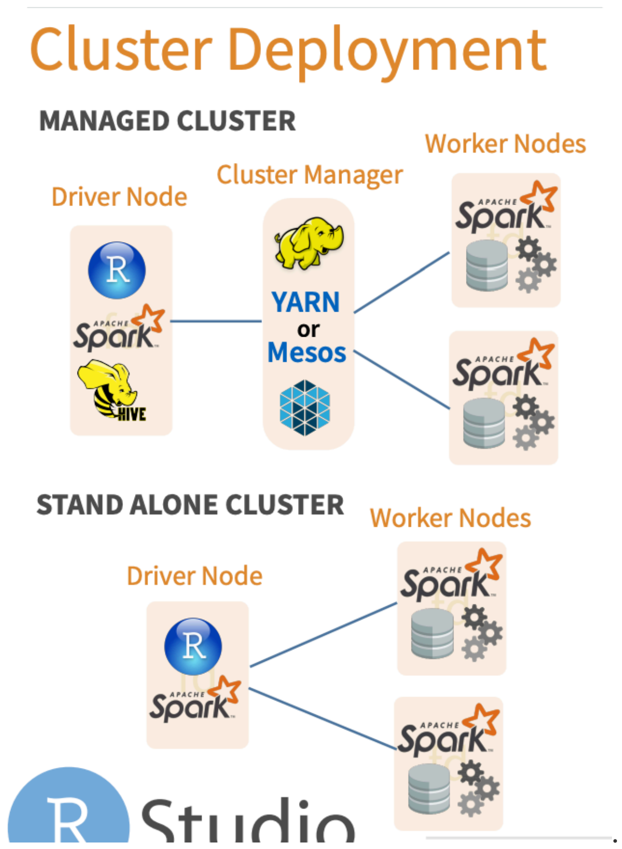
```
TextFile
 "hello Tyth"
 "cool example, eh?"
 "goodbye"

TextFile.map(line => line.split(" ").size)
 2
 3
 1
TextFile.map(line => line.split(" ").size).reduce((a, b) => if (a > b) a else b)
 3
   Steps here, recall `(a, b) => if (a > b) a else b)`
   - op( op(2, 3), 1) evaluates to op(3, 1), since op(2, 3) = 3
   - op( 3, 1 ) = 3
```

Sparklyr

Install spark for R using sparklyr





On a YARN managed cluster:

1. Install RStudio Server or RStudio Pro on one of the existing nodes, preferably an edge node
2. Locate path to the cluster's Spark Home Directory, it normally is "`/usr/lib/spark`"
3. Open a connection `spark_connect(master="yarn-client", version = "1.6.2", spark_home = [Cluster's Spark path])`

`On a spark Standalone cluster`

1. Install RStudio Server or RStudio Pro on one of the existing nodes or a server in the same LAN
2. Install a local version of Spark: `spark_install (version = "2.0.1")`
3. Open a connection spark_connect(`master="spark:// host:port", version = "2.0.1", spark_home = spark_home_dir()`

```r
install.packages("sparklyr")
library(sparklyr)
spark_install(version = "2.1.0")
sc <- spark_connect(master = "local")
```

Now load the flights omitting the missing ones using `na.omit -> drops bands which don't have data`

```r
flights = na.omit(flights)
flights_tbl <- copy_to(sc, flights, "flights", overwrite = TRUE)
```

Use the `summary()` function on these objects to look at how well they fit. To deal with how to evaluate models (e.g., training, validating, etc.) is beyond the scope of this tutorial.

!!! The function `copy_to`
Copy A Local Data Frame To A Remote Src: This function uploads a local data frame into a remote data source, creating the table definition as needed. Wherever possible, the new object will be temporary, limited to the current connection to the source.

Plotting

One can use `ggplot` but it cannot plot smooth line, therefore it's is better to use `geom_abline to plot.`

`ggplot()`

K means

Performing k-means clustering with 2 simple clusters (k=2)

```
km2=ml_kmeans(flights_tbl, 2,
              features = c("dep_delay", "air_time"))
```

The function works like this
`ml_kmeans(x, centers, iter.max = 100, features = tbl_vars(x), compute.cost = TRUE, tolerance = 1e-04, ml.options = ml_options(), …)` where centers is the number of clusters around where to

This can be plotted with

```
Library(tidyverse)
dat =
  select(flights_tbl, dep_delay, air_time) %>%
  sample_n(1000)
ggplot(data = dat,
       aes(x = dep_delay, y = air_time)) +
  geom_point() +
  geom_point(data = km2$centers,
             aes(x = dep_delay,y = air_time,
                 color  = 'Center'))
```

6 Notes textbook and papers

Textbook - Ch. 2

A cluster is a group of machines tied together to pool their resources --> SPark is the framework that allows to coordinate work among them. Spark only manages and coordinates the exectution of tasks across a cluster of computers.

- The cluster of machines is managed by a cluster manager like Spark's standalone cluster manager, YARN or Mesos
- These clusters receive Spark Applications
- The cluster will grant resrouces to application so that they can complete their work

Spark Applications

Each application consists of two processes:

- A *diver* process --> essential for every Spark Application
  - Maintains nformation about The Spark application
  - Respond to a user's input/program
  - Analyzes, distributes and schedules work across the executors
- A set of ***executor*** processes --> responsible for carrying out the work assigned by the driver
  - Execute code sent by driver
  - Report the state of the computation back to the driver node

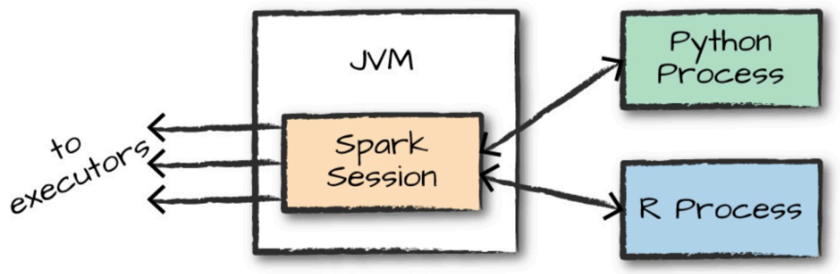


Figure 2-2. The relationship between the SparkSession and Spark's Language API

In addition to its cluster mode, Spark has a local mode in which the driver and exectuors are processes - different threads etc.
The user can alwyas specify how many executors should fall on each noe through configuration

Spark language API

Spark accept variours programming languages:

- Scala
- Java
- Python

- SQL - ANSI SQL 2003 standard
- R (Sparklyr - Open source - and SparkR
  Spark has 2 set of APIs: low-level unstructured, high-level structured

Starting Spark

To start spark one has to create a Spark session
`~/bin/pyspark` to run the shell/interactive Spark application

SparkSession instance: used to exectue user-defined manipulation across the cluster.

- One-to-one corrispondence between SparkSession and Spark Application
  `Spark`
  `myrange = spark.range(1000).toDF("numbers"`
  Creates a dataframe with one columns contaitning 1000 rows (0-999) that represents a distirbuted collection.
  Distirbuted collection: each chunk of rows is on a different executor

Dataframes

Is the most commmon structured API in Spark and represents a table fo data with R and C. The list that defines them is called a *schema*. It is a distributed spreadsheet.
Core abstractions in Spark are: Dataset, DataFrames, SQL Tables adn RDD (resilient distribtued datasets)

Spark breaks up the data into chunks called partitions --> one partition = parallelism of 1, even if one has 1000s of executuros

Transformations

The core data structures are immutable. To change them one needs to tell Spark how to modify them. E.g.
`divisBy2 = myrange.where("numbers % 2 = 0")`
These return no output, because Spark will not act on transformation until we call an action. There are two types of transofmration: those that specify narrow dependencies and those that specify wide dependencies

- **Narrow dependencies**: for each input contribute only to one output: e.g. `where` statement - 1 parititon contributes to 1 output partition
    - Spark will automatically pipeline the tranformation: if we specify multiple filters on DF they will be performed in memory
- **Wide dependencies**: input partitions contribute to many output partitions: this is, e.g., a *shuffle* operation.
    - Spark will write the results to disk

Lazy evaluations

Spark will wait until the last moment before exectuting the graph of computation instructions.
Instead of modifying the data, in Spark one builds up a plan of transformations that one would like to apply to the data.
By waiting until the end of the *plan*, Spark can optimize at the maximum this plan: *predicate pushdown* is an example of this, if we use a filter at the end that only wants one row of data, Spark will actually push the filter

down automatically and execute it at the beginning of the plan/ as close as the source as it can.

Actions trigger the computation and are of 3 kinds:

- Actions to view data in console
- Actions to collect data to native objects in the respective langauge
- Actions to write to output data sources

Example

- Schema inference: we want Spark to take th ebest guess at what the schema of out DataFrame should be

```
flightData2015 = spark\
.read\
.option("inferSchema", "true")\
.option("header", "true")\
.csv("/data/flight/csv/2015-summary.csv")
```
*Read is a narrow transformation
*Here # of rows is unspecified beacuse reading is a data transformation and therefore is lazy

```
flightData2015.take(3)
```
if we `take` we start an action and we can see the same result as just looking at `head` of the file

Now we sort the DF, which is a widetranformation that returns a new DF but is not an action

```
flightData2015.sort("count").explain()
```
The `explain` part allows us to peek at the plan that Spark is creating.

- the top of the plan is the end result, the bottom is the source(s) of the data --> that is the reason of "*pushdown*"

We then set the number of parititon to 5, from the 200 shuffle default parititons.
```
spark.conf.set("spark.sql.shuffle.partitions","5")
flightData2015.sort("count").take(2)
```

In images

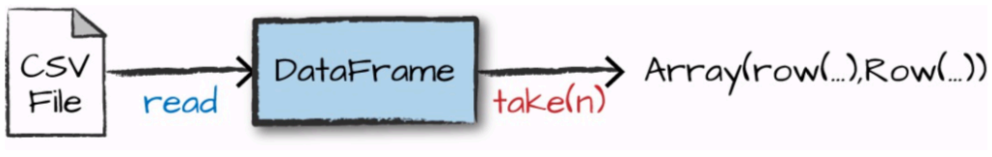


Figure 2-7. Reading a CSV file into a DataFrame and converting it to a local array or list of rows

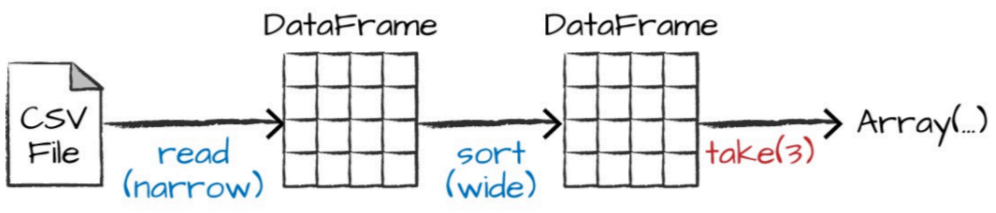


Figure 2-8. Reading, sorting, and collecting a DataFrame

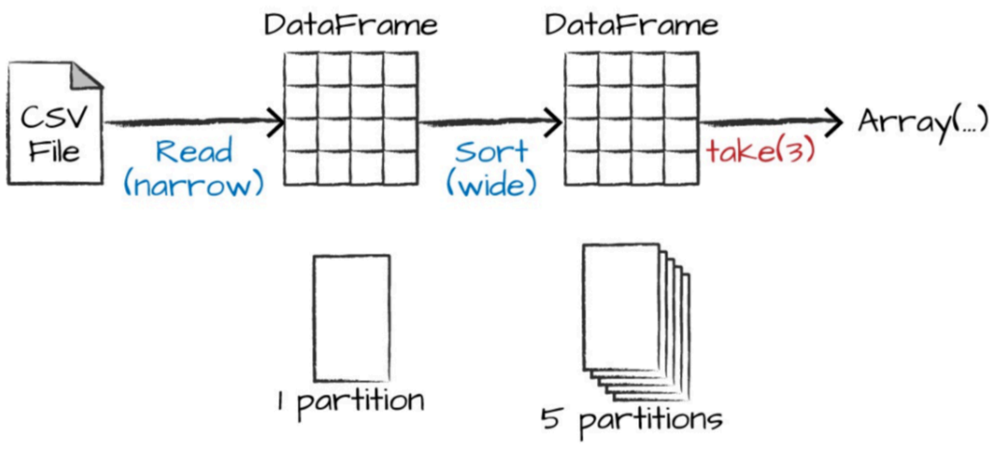


Figure 2-9. The process of logical and physical DataFrame manipulation

*This is the hearth of Spark's programming: functional programming where the same inputs always result in the same outputs when the transformations on that data stay constant—> Spark know how to recompute any partition by performing all the operationg it had before done

DataFrame and SQL

With SparkSQL one can register any DF as a table or view and query it using pure SQL.
--> DF into a table or view
`flightData2015.createOrReplaceTempView("flight_data_2015")`

To query data in SQL we use `spark.sql` function
Example

```
sqlWay = spark.sql ("""
SELECT DEST_COUNTRY_NAME, count(1)
FROM flight_data_205
GROUP BY DEST_COUNTRY_NAME
""")

dataFrameWay = flightData2015\
.groupBy("DEST_COUNTRY_NAME")\
.count()
```

they are the same!

Another example
`from org.apache.saprk.sql.funtions import desc`

```

`flightData2015`
    .groupBy(DEST_COUNTRY_NAME)\
    .sum(“count”)\
    .withColumnRenamed("sum(count)", "destination_total")\
    .sort(desc("destination_total")\
    .limit(5)\
    .show()
    ```
The exectuion plan will differ in order beacuse is a directed acyclic graph (DAG) of transofrmations, each results in a new immutable DF.
We don't always have to collect data, we can just return it somewhere else.
```
```

Textbook - Ch. 10

Spark SQL is one of the most useful and powerful features in Spark. WIth Spark SQL one can run SQL queries against view or tables and use system fucntions or define functions.

SQL

SQL is a domain specific language for expressing realtional operations ver data --> used in all relational databases but also ijn many NoSQL databases in order to make working with thier databases easier.
Before SPark, Hive was the layer to access big Data SQL - however now is better to use Spark SQL.

Why? Because Spark has a unifying API that allows for data (e.g.) to be extracted with SQL, maniputlated with a DF, passed into a ML library and written out to another data sources.

Spark SQL is intended to operate as an online analytic processing database.

Spark and Hive
Hive's metastore is the way in which Hive maintatins table information for use across sessions. With Spark SQL one can connect to the Hive metastore and access table metadata to reduce file listing when accessing information.

How to run SQL queries

To start the command line interface (CLI) one can easily use
`./bin/spark-sql` for options: `--help`

One can also exectue SQL ina ad hoc manner via Spark's language API -
`spark.sql("SELECT 1 + 1").show()` --> this returns a DF that can be then evaluated programmatically

One can also program a bit in SQL and a bit in DF as in

```
spark.sql(""" SELECT DEST_COUNTRY_NAME, sum(count)
FROM some_sql_view GROUP BY DEST_COUNTRY_NAME
""")\
.where("DEST_CUNTRY_NAME like '5%').where("sum(count)' > 10")\
.count()
```
--> SQL to DF

The highest abrsatrcion in Spark SQL is the catalog --> abstraction for the storage of metadata about the data stored in your tables as weell as other things (dbs, fucntion, views)

Tables

Fundamental in Spark. THey are logically equivalent to a DF. They can be joined, filtered, aggregated etc. Tables ALWAYS contain data --> no temporary table, only "view"
If one drops a table, you lose the data that it contains

Managed-unmanaged

- Managed when Spark manages the etadata for a set of files as well as for the data
- Unmanaged when you define a table from files on disk --> saveAsTable on DF creates an unmanaged table

Creating tables
From a variety of sources. One can read the entire Data Source API whiting SQL.

One does not need to define a table and then put data in it --> create on the fly

Commands
`INSERT INTO`: one can also provide a partition specification if you want to write only in a certain partition
`DESCRIBE TABLE` to insert comments
`REFRESH` to read the most recent set of data
`REPAIR TABLE`: refresher the parititons --> new partition information
`DROP TABLE` if it is managed boht the data and the table definition will be removed
`CACHE TABLE` used to chache and unchace

Views

Are a set of transofrmations on top of an existing table --> useful to organize or reuse the query.
Views can be:

- Global - resolved regardless of database and viewable across Spark, removed at end session
- Database - ?
- Per session - not registered to db, removed at end session

To the user views are displayes as tables, except that they simply performa a transofrmation on the source data at query time (`filter`, `select` or `GROUP BY` or even `ROLL UP`)

Databases

Useful tool to organize tables. If you change the db all user-defined tables will remain in the previous one and will have to be queried differenlty

To perform queries use `USE` + db name
After this all queries will try to resolve table names to this db

- To query from different db use: `SELECT * FROM default.xxx` the `default` is important here
- To switch back to the default db: `USE default`
- To see which db you are using: `SELECT current_database`

`SELECT`
`CASE WHEN ...` --> to conditionally replace some values

Complex Types

ARe a departure from standard SQL --> 3 core complex types:

- **Structs**: similar to maps. They are used to create or query nested data in spark
  - To create just wrap a set of columns in parentheses:
  - --> `CREATE VIEW IF NOT EXISTS xxx AS`
  - --> `SELECT (X, Y) as z, count FROM xydb`
  - One can query individual columns, using a *dot syntax*
- **Lists**: `collect_set` or `collect_list`
- Maps

Functions

`SHOW FUNCTIONS` to show a list of functions
`SHOW SYSTEM FUNCTIONS` or `USER`

6.1 Lecture Notes

Esri lecture

Assignments

Mapillary --> capture at least 100 images in a stream of images

Read --> http://www.esri.com/~/media/Files/Pdfs/news/arcuser/0518/Machine Learning in ArcGIS.pdf

Questions: Predictive analysis in combination with spatial analysis can assist in finding hot spots for traffic accidents, as illustrated in the article. In this example are traffic accidents predicted based on a number of factors. Can you think of any additional factor that will make more accurate analysis, and one that creates a less accurate analysis?

- Esri is an end-to-end imagery platform that incorporates image management, data preparation and training, deep learning model integration, and operational workflows that can be shared within your organization or to the public
- Everything is built using ArcGIS online (the workflow is somehting like Deep Learning, Online, Workfoce, Operations)
- Use SkySat imagery that takes any location of Earth at a resuolution of 72cm, twice daily, they use a mosiac dataset to store and manage this data

Train the Image data

- Create a feature using `Create Feature`
- Digitize polygons around strutucres and then check if all the features in the same category had the same integer value to indicate that they are all in the same class structures --> you need an adequate sample
- Export training sites to be modeled using Tensor Flow
  - this produces chip files with the template .emd file
  - Data Scientist does this step
- Run Decect Object Tool with trained data
- Run Configurable App to verify model output and accuracy

Lecture

Describing reality one can do that with

1. Thematic description (what)
2. Geometric description (where)
3. Temporal descritpion (when)

GIS works by storing data as a collection of thematic layers. There are two components:

- Geometric data
- Attribute data

and 2 ways of representing the models:

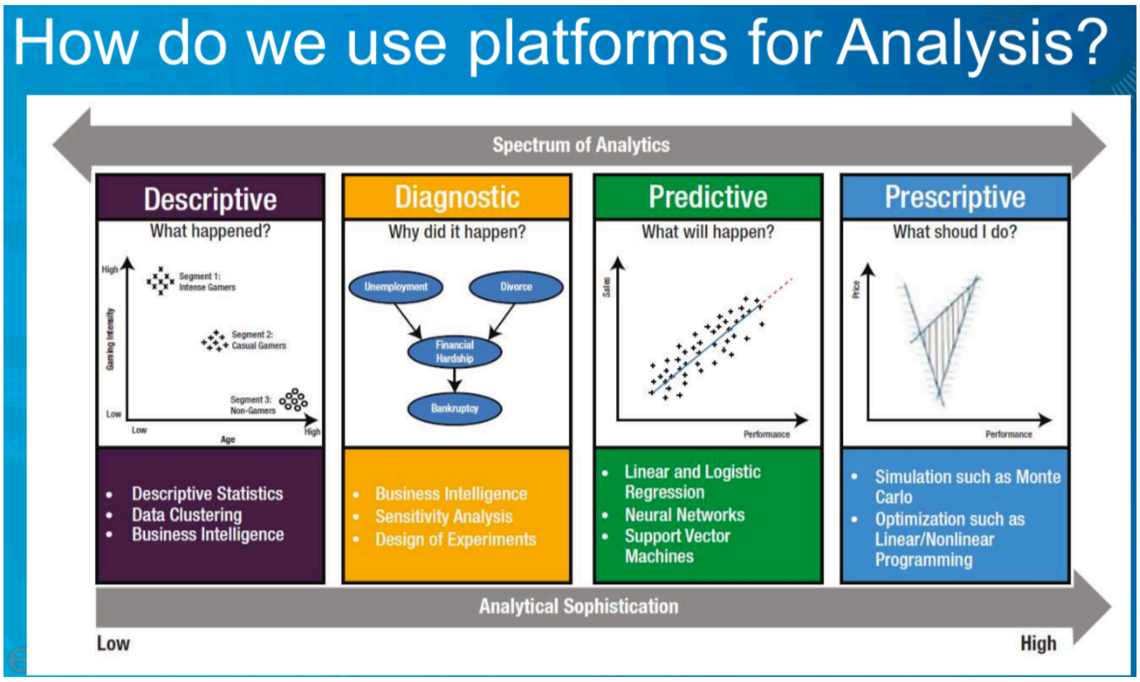
- Vector data model: discrete objects (trees, borders, ...)
- Raster data model: continious phenomena (temperature, slope, elevation, ...)

ArcGIS online has 200Tb of data public, 1.6M user **per Day**, and 5B map tile requests per months

Statistical tools in ArcGIS:

- Classification (ML class, Random Tree, SVM)
- Prediction (EBK, OLS, ...)
- Clustering (Hot Spot Analaysis, Spatially constrained Multivariate Clustering,...)

Some resources:
High resolution maps: https://map.openaerialmap.org


How do we use platforms for Analysis?

Spectrum of Analytics

Descriptive — What happened?
- Descriptive Statistics
- Data Clustering
- Business Intelligence

Diagnostic — Why did it happen?
- Business Intelligence
- Sensitivity Analysis
- Design of Experiments

Predictive — What will happen?
- Linear and Logistic Regression
- Neural Networks
- Support Vector Machines

Prescriptive — What shoud I do?
- Simulation such as Monte Carlo
- Optimization such as Linear/Nonlinear Programming

Analytical Sophistication

Low — High

SQL, NoSQL and Spark

NoSQL have 6 key features:

1. Horizontally scale simple operation
2. Ability to replicate data over many server
3. Simple call inerface or protocla
4. Weaker concurrency model
5. Efficient use of distributed idnexes
6. Dynamically add new attributes

Other Properties

- NoSQL have a shared nothing horizontal scaling --> replicating and parititoning data over many servers
- No ACID trnsactional properties: updates are propagated but limited guaranteess on the consistency of the reads:
    - BASE: basically available, Soft state, Evenutally consisten
    - ACID: atomicicty, consistency, Isolation, Durability
      Giving up ACID constraints, one can achieve higher performance and scalability

If new relational systems can do everything that a NoSQL system can, why chosing NoSQL?

- Relational DBMS have the majority of the market share, built to handle other application loads in the past
- However, not yet seen good benchmarks showing that RDBM can achieve scaling compared to NoSQL
- If you require a lookup, a key value store is adequate and probablu easier to understand than a relational DBMS
- Some application require a flexible schema, allowing each object in a collection to have different attributes, rare to find a RDBMs that allow for this.
- R DBMS makes expensive operations too easy (?)
- RDBM have the majority of market share but many other products arrived

SQL
`CREATE`: create DB objects
`ALTER`: modify structure
`DROP`: delete objects
`TRUNCATE`: delete data in table without altering structure

`CREATE TABLE X {X integer primary key,Y Char Not NULL}`: 2 rows 3 columns
One can create tables with composite keys `{CONSTRAINT name PRIMARY KEY(X,Y)` takes X and Y as primary keys.

`FOREIGN KEY` ?

SQL DML
Data manipulation language
`INSERT INTO X VALUES(...)`
`INSERT INTO (X,Y) VALUES (Z,U)` Z goes to X and U goes to Y
`UPDATE` values in an existing row
`DELETE` a row or a set of rows
Queries
`SELECT` retrives data from one or more tables and creates a new temporary table
`DISTINCT` can be added to the SELECT statement to prevent that duplicate rows are shown
`WHERE` keyword criterai that have to be met - can use OR or AND
`IN` the WHERE clause includes the IN keyword to make sure that certain column values are included in the query result
`NOT` prevents records from being part of the resulting set
`BETWEEN` obious
`LIKE` partailly completes values can be serached for - use wildcards to ifnd records that are equal to a certain string vaue (% multiple characters, _ single charcter)
`ORDER BY` sort results
`JOINS` left outer, right outer, inner join

Why Spark SQL?
Compatible with Hive and can be used with DataFrames
`./bin/spark-sql`
`spark.sql("SELECT 1 + 1").show()`

A table is logically equivalent to a DataFrame

If then statements
```
SELECT
CASE WHEN XXX THEN yyy
CASE WHEN XXX THEN uuu
ELSE END
FROM ...
```

6.2 Laboratory Notes (5)

Analysis in SQL for Spark

First steps

To check checksums, if they are downloaded into XY
`sha1sum -c XY`
If they do not match, redownload the data

Assignment:: preprocess data

- `SparkContext()`
  - Create a SparkContext that loads settings from system properties (for instance, when launching with `./bin/spark-submit`).
  - `SparkContext(String master, String appName, SparkConf conf)` --> Alternative constructor that allows setting common Spark properties directly
- `SparkSession`:
  - `getOrCreate()` --> Gets an existing SparkSession or, if there is no existing one, creates a new one based on the options set in this builder.
  - `appName()` --> sets a ame for the applicatoin

RDDs

```
// To create text file RDDs: `sc.textFile()`
`epi_path = "/home/labuser/epidemiology/data/epi_tutorial_7.csv"`
`chd_RDD = sc.textFile(chd_path)`
`men_RDD = sc.textFile(men_path)`
`t2d_RDD = sc.textFile(t2d_path)`
```

To extract first line from RDD `meno\_header = meno\_RDD.first()`

To transform from RDD to dataframe - split Data set

```
`temp\_var = meno\_RDD.filter(lambda line: line != meno\_header).map(lambda k: k.split("\\t"))`
// first uses the first row to create the column header ` line != meno\_header` and then splits the map on tabs to divide the columns/values
```

Create a Data Frame from the data just split: `chd\_df = temp\_var.toDF(header.split(";"))`

- To lookup data types in every column: `chd\_df.schema()`
- To show first 20 rows: `chd\_df.show()`
- To show all the column names: `chd\_df.columns`

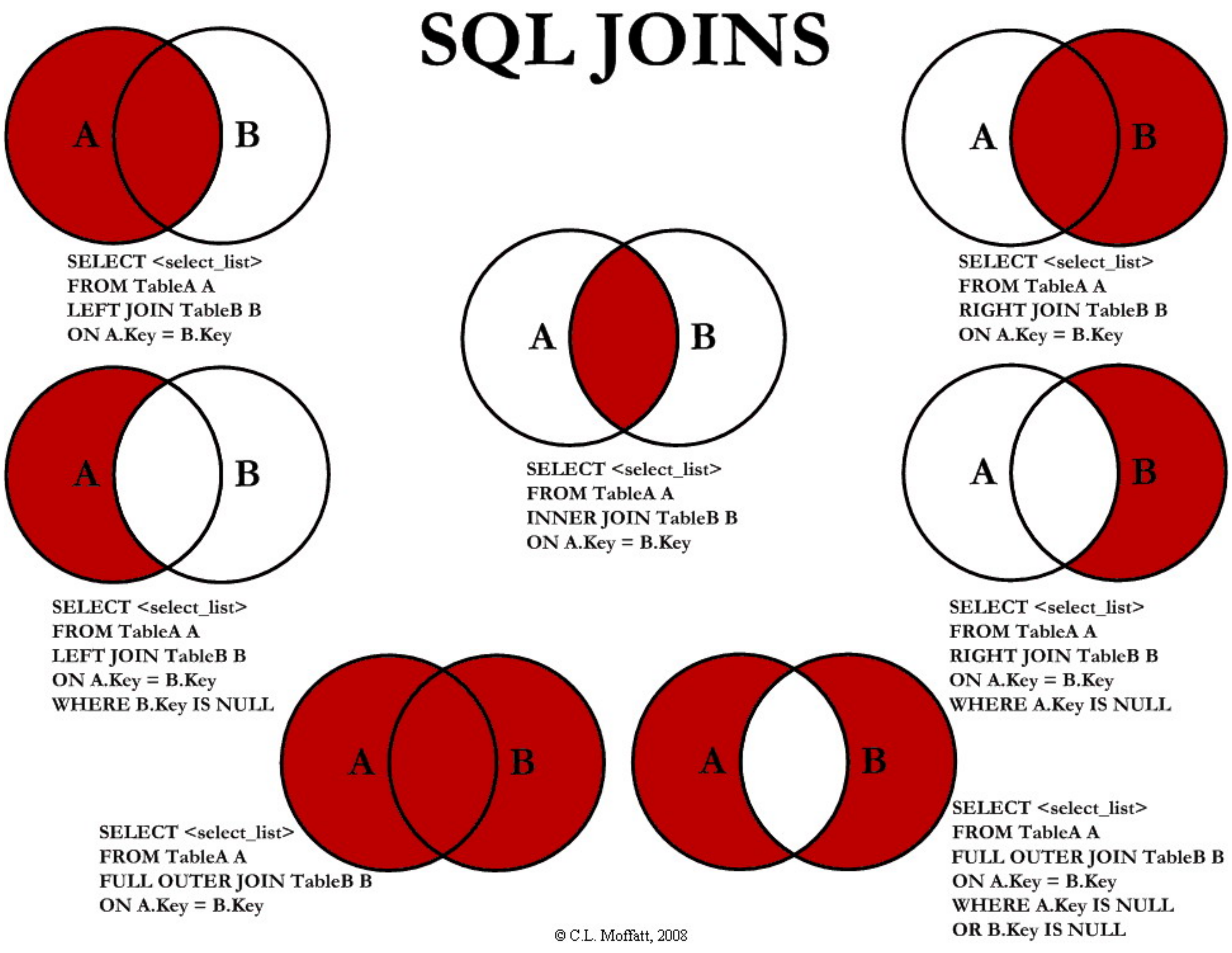
- To create a matrix with onlu the columns selected `chd\_df.select(["markername", "chr", "bp\_hg19"]).show()`
- To create matrix from DF: `matrix = df.select(["MarkerName","allele1","allele2"])`
- To show all columns: `meno\_df.columns`
- To show the type of the columns: `meno\_df.schema()`

Exercise 1

Creates a dataframe: and divides on the delimiter ";" + infers the schema automatically with "inferSchema=TRUE"

```
chd_df = spark.read.option("delimiter",";").csv(chd\_path, inferSchema=True, header=True)
```

Join dataset on markername into one integrated raw data file:



```
//Creates a function that takes for each line takes the 2 values and merge them togheter separated by ":"
`from pyspark.sql.functions import col`
`def map_chr_bp_hg19(line):`
`    chr_val = line[1]`
`    bp_hg19_val = line[2]`
`    return [str(chr_val) + ':' + str(bp_hg19_val)] + list(line)`

```

```
// Add an additional column to the CHD-(cardio)-dataframe
`df_chd_header = chd_df.schema.names` get names of the schema - name of columns?
`df_chd_chrpos = chd_df.rdd.map(map_chr_bp_hg19)`
`df_chd_chrpos = df_chd_chrpos.toDF(["Chr:Position"] + df_chd_header)`
```

Join the CARDIO-dataframe and the MENOPAUSE-dataframe, the CHD and menopause

```
df_join_chd_men = df_chd_chrpos.join(men_df, 'markerName', 'inner').cache()
df_join_chd_men_t2d = df_join_chd_men.join(t2d_df, 'Chr:Position', 'inner').cache()
```

Record how many markers cannot be joined

```
# get the number of rows form the loaded data frames
chd_count = df_chd_chrpos.count()
men_count = men_df.count()
t2d_count = t2d_df.count()

# count the row in the joined data-frame
chd_men_t2d_count = df_join_chd_men_t2d.count()

# print the results
print ("Numer of joined lines:  {0: 10d}".format(chd_men_t2d_count))
print ("Removed rows from cata: {0: 10d}".format(chd_count - chd_men_t2d_count))
print ("Removed rows from meno: {0: 10d}".format(men_count - chd_men_t2d_count))
print ("Removed rows from meta: {0: 10d}".format(t2d_count - chd_men_t2d_count))
```

Signal line that have words with empty value

```
// Define function that return false for each word that is empty
def remove_incomplete_rows(line):
    for word in line:
        if word == "" or word == None:
            return False
    return True
```

Remove duplicated rows

```
epi_unique_df = df_join_chd_men_t2d.dropDuplicates().cache()
print ("Number of duplicate lines removed: {0}".format(chd_men_t2d_count -
epi_unique_df.count()))
```

Remove incomplete lines

```
epi_complete_df = df_join_chd_men_t2d.rdd.filter(remove_incomplete_rows).toDF()
```

Map every row to a csv-like string - return each value in data followed by a “;”

```
def toCSVLine(data):
    return ';'.join(str(d) for d in data)
```

Save the datafile as a comma separated csv-file

```
lines = df_clean_epi.rdd.map(toCSVLine)
lines.saveAsTextFile(epi_path)
```

This is used to split a line into words (thanks to flatMap) and then

```
def split_words(line):
  return line.split()
```

```python

def create_pair(word):
  return (word, 1)

def sun_counts(a, b):
  return a + b

pairs_RDD = text_RDD.flatMap(split_words).map(create_pair)
wordcounts_RDD = pairs_RDD.reduceByKey(sun_counts, numPartitions = 1)

wordcounts_RDD.saveAsTextFile("/home/labuser/wordcount_spark/output");
```

How does `reduce by key` work?

```scala
pairs.reduceByKey((a, b) => a + b)
pairs.reduceByKey((a: Int, b: Int) => a + b)
pairs.reduceByKey((accumulatedValue: Int, currentValue: Int) => accumulatedValue + currentValue)
pairs.reduce((accumulatedValue: List[(String, Int)], currentValue: (String, Int)) => {
  //Turn the accumulated value into a true key->value mapping
  val accumAsMap = accumulatedValue.toMap
  //Try to get the key's current value if we've already encountered it
  accumAsMap.get(currentValue._1) match {
    //If we have encountered it, then add the new value to the existing value and overwrite the old
    case Some(value : Int) => (accumAsMap + (currentValue._1 -> (value + currentValue._2))).toList
    //If we have NOT encountered it, then simply add it to the list
    case None => currentValue :: accumulatedValue
  }
})
```

7.1 Lecture Notes

Guest CoreLifeAnalyitcs

The cures of tomorrow are in the data one can't analysize today!

HC StratoMineR workflow:

- Upload data, preporcess, select parameter, controal quality
- Normòliza transofr and standrdize data
- Multiple Imputation
- Reduce data
- Select HIT
- Data mining
- Reporting

8.1 Lecture Notes

Big Data in Reserach, Privacy and Data Protection

GDPR: Handbook EU data protection law. Privacy - "ability of being completely hidden from others" nowadays very hard|impossible. The right of privacy, by the EU court of human rights encompasses the physical, psychological and moral aspects of the personal integrity, identity and autonomy of individuals. The EU court of Justice: "only affected when the data affected may allow very precise conclusions to be drawn concerning the private lives of the persons whose data has been retained".

Objective of** personal data protection**:

- Respect for individual rights and interests
- Social and economical progress
- Privacy is a negative obligation to the governments to respect their citizens

The "**Nothing to hide**" argument:

- Ability to hide secrets and not to have them displayed"
- Do we have curtains?

Right to privacy:

- European court of human right and EU court of Justice -> right to privacy is affected when data may allow precise conclusions to be drawn concering the private lives of the persons whose data has been retained

GDPR: does not apply if subject has given explicit consent or if an apporpriate research exemption from this prohibition is laid down in national law - special cateogires of personal data are prohibited from being processed.

Regulation of Big Data in reserach

Open questions:
(within the consent or anonymise paradigm)

1. **Anonymity**
    - Might be** impossible to guarantee anonymity**: absolute anonymity is a myth!
    - To determine if someone is identifiable:
        - Accounts should be taken off all the means reasonably likely - all objective factors such as the costs of and the maount of time required to identify, taking into consideration the technology at the time of the processing and the development - to be used by either by the controller or by another person (controlling or reading the data) to identify the account
    - Access to data is proteced - until there is a link that can de-anonimyze the data, the data is NOT anonymous
    - Irreversible anonymisiation requires:
        - Extensive stripping and reduces data quality
        - Esclusion of data linkage
            - Ways forward?

   i. Regarding Two-way coded data as de-identified. However pseudonymisation is not a method of anonymisation (GDPR)
   ii. Anonymisation should be avoided. No obligation to protect data or interests while risk of re-identification and group risks remain

-> Anonymisation is an unsafe strategy for ensuring protection of individual rights and interests
-> Is a difficult practice without compromising utility of the dataset
-> Pseudonymisation is a useful method to reduce privacy risks without preclusing the applicability of the law

2. **Consent**
   - One needs to obtain consent to use the data -> it is a problem
   - With each failure of privacy self-
   - Problems:
     - Cognitive - people do not read privacy policies
     -
   - Ways forward?
     - Project specific consent is too costly and time consuming!
     - Consent at a one-off event: legal validity? Ethical acceptability?
     - Dynamic consent: consent as an ongoing process - participatory approach

-> Consent **does not protect**, only gives way to understand how data are used

(Outside the consent or anonymise paradigm)
3. **Research Exemption**
* Should there be another legal basis than consent? There is no research is that
* There are other research topics: consent kept to minimum, however there is strong justification in the public interest.
* GDPR: says it is necessary
* Safeguards? Literature says: limiting data access and use; opt-out registration; psudonymisation; authorisation by ethics committee - every data bank could have one?; engage in public participation
* GDPR: very vague norms to address this; privacy by design + good information governance

Purpose limitation: you are allowed to give the data to anyone as long it is research. However, a medical doctor can use data, but it cannot take the data and then give it out to research (?)
Storage limitation: data can be stored, but only in accordance with Art 89 (scientific reserach)
-> very difficult to protect data but at the same time make research possible

Definition of scientific research?
* If research is not in public interest, then that is not research and you need to obtain consent

Problem of data vs. metadata

Mental Healthcare

What is changing? From patient to citizen, from Doctors to ICT, from treatment based on trial to clinical intelligence

E-mental health: is chaning on three levels

1. Research -real time sensors
2. Conent - applications

3. Organization of care - from hospital to networks of healthcare organizations

Until recently there was not too much data about patients, in particular about their daily lives. Nowdays one can use real-time analyitics and trackers [to come to have a digital fenotype of a particular patient]. All these trackers make an influx of data that many hospital cannot deal with/ don't know how to use
--> all these data create a "P4" medicine: predictive, preventive, personalized and participating! It is knowledge that help and is made through combining domains.

Where are we now?

Little knowledge about the brain. Knowledge about syntoms (this fenotype) but not about the composition of the brain. In 1980, invented the DSM with cluster of syntoms categorized with names to be shared with practicians around the world.

- Thanks to imagining the next frontier was investigating changing in the brain activity of patients who were categorized as patients with synthoms
- To look at patient and search for **patterns** and **structures** both physically and in their family/society surrounding them --> understand the logic behind it
- They tried to go from linear to complex thinking and check the influence of various variables on the patient.
    - Used public and private data sources
    - Made workgroups with different research focus [EPD]
    - Aggregated and prepared data from all these sources, intergrated them to do exploration and use them in practice

Questions they wanted to answer?

- Aggression during admission
- Model to predict effectiveness of antidepressants
- Pilot of wearables to collect real life data on stress --> create personal profile out of these datasets
- Not ethical to collect info about genetics
- Ethical problems possible and present --> focus on health improvement, measure using this proxy

Natural Language Processing in Psychiatry

- 80% of data is unstructured --> treatment plan, incident reports, memos, questionnaires AND doctor's and nurse notes
- How to reuse this data: it's domain specific, has spelling mistakes and abbreviations + ideosyncracies of doctors and nurses
- Instead of redefining registration protocols, the focus here is to translate them into useful data

Steps of preprocessing

1. De-identification
   2. Developed DEDUCE which is automatica de-identification method
   3. Uses fuzzy string matching and lookup lists with regular expressions, good results generally
   4. Uses many pre-compiled lists --> can cause false positives with other out-of-the scope names

- Recall how much information is filtered
- Precision how much infomration that should not be filtered is filtered for every 4 matches, one was a false positive

Some example topics tried

- Sentiment analysis trained on book reviews
- Clustering of reports based on cause of incidents; manual clustering is way better than automatic (found: medication is often a cause of violence)
- Information retreval in free text (e.g. smoking status of patients)
- Text classification - assessing violence risk

Assessing violence risk:

- 1/8 displays violent behavior during ammission with nurses or doctors
  - can be both verbal or physical
- Assessment tools don't work very well for all the populations (selection bias) --> one cannot be certain these tools will actually work well - developed a lot of years ago and on specific populations
- They wanted to improve this using already present data.
- Word2vec --> takes context words and uses a model to check if there appears a target word inside these contex. One runs this through all corpus --> allows to carry out mathematical operations on words
- Doc2vec --> this same idea can be applied to entire documents (represented as a bag of word)

Model

- Take the text notes, put the into doc2vec, translate into numbers and then average --> we have a numerical representation of a patient
  Modelling these data: RNN
- Process each note and **classify** the admission using recurrent neural network
  - Training phases: repeating for all admissions
  - Output of the model is a probability [0,1]
- Cross validation over all dataset
- To measure performance: area under Curve
  - One problem with accuracy is that the clusters here are unbalanced, high accuracy does not mean much
  - **Use AUC**: True and False positive Rate = TP/(TP+FN) and FP/(FP+TN) --> depending on the threhold for high and low risk we move patients from one to the other
- Work in progress: make classificaiton available and understandable to the patient

Young Innovator Programme: [digital literacy] Everyone can code -- Apple --> teaching while amazing children/ exploring the city/ meeting new people; coding session; Sketch walks; "Sessions"

8.2 Laboratory Notes

Load Data files into notebook
Note that only the SAMPLE data is used, if you want to test the real data you can uncommend the following lines

```python
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("epidemiology").getOrCreate()
sc = spark.sparkContext

chd_paper_path = "/home/labuser/epidemiology/data/CHD_paper.csv"
men_paper_path = "/home/labuser/epidemiology/data/MENO_paper.csv"
t2d_paper_path = "/home/labuser/epidemiology/data/T2D_paper.csv"

epi_path = "/home/labuser/epidemiology/data/epi.csv"
```

Read into DF

```python
chd_df = spark.read.option("delimiter",";").csv(chd_path, inferSchema=True, header=True)
men_df = spark.read.option("delimiter",";").csv(men_path, inferSchema=True, header=True)
t2d_df = spark.read.option("delimiter",";").csv(t2d_path, inferSchema=True, header=True)
```

Sort on p-value to show the top hits
P-value is a column

```python
chd_df_sorted = chd_df.orderBy("p_dgc").cache()
men_df_sorted = men_df.orderBy("P").cache()
t2d_df_sorted = t2d_df.orderBy("P-value").cache()

chd_df_sorted.show()
men_df_sorted.show()
t2d_df_sorted.show()
```

Check overlap between top hits

```python
epi_df = spark.read.csv(epi_path).toDF('Chr:Position', 'markername', 'chr', 'bp_hg19', 'effect_allele',
                                       'noneffect_allele', 'effect_allele_freq', 'median_info', 'model', 'beta',
                                       'se_dgc', 'p_dgc', 'het_pvalue', 'n_studies', 'allele1', 'allele2', 'HapMap_eaf',
                                       'effect', 'stderr', 'p', 'Allele1', 'Allele2', 'Effect', 'StdErr', 'P-value',
                                       'TotalSampleSize')

epi_df.show()
```

Open top hits as mentioned in the papers

```python
chd_paper_df = spark.read.option("delimiter","\t").csv(chd_paper_path, inferSchema=True,
header=True)
men_paper_df = spark.read.option("delimiter","\t").csv(men_paper_path, inferSchema=True, header=True)
t2d_paper_df = spark.read.option("delimiter","\t").csv(t2d_paper_path, inferSchema=True, header=True)
```

```
// Join the data sets from the papers
t2d_men_paper_df = t2d_paper_df.join(men_paper_df, 'markername', 'inner').cache()
t2d_chd_paper_df = t2d_paper_df.join(chd_paper_df, 'markername', 'inner').cache()
men_chd_paper_df = men_paper_df.join(chd_paper_df, 'markername', 'inner').cache()

// join the data sets from the papers with the epi dataset
epi_men_paper_df = epi_df.join(men_paper_df, 'markername', 'inner').cache()
epi_t2d_paper_df = epi_df.join(t2d_paper_df, 'markername', 'inner').cache()
epi_chd_paper_df = epi_df.join(chd_paper_df, 'markername', 'inner').cache()
```

If you count a DF which was joined “inner” with another one, you can see the hits in both databases - if there were similar values on the “markername” variable that you are joining them on

```
print ("Epi shares {0} hits with cardio.".format(epi_chd_paper_df.count()))
```

Add rank to sorted data: per each line,

```
def map_ranking(line):
    return [int(line[1])] + list(line[0])

# create headers for the rank tables
chd_df_header = chd_df_sorted.schema.names
men_df_header = men_df_sorted.schema.names
t2d_df_header = t2d_df_sorted.schema.names

# create the rank tables (trait plus rank)
chd_rank_df = chd_df_sorted.rdd.zipWithIndex().map(map_ranking).toDF(["Rank"] + chd_df_header).cache()
men_rank_df = men_df_sorted.rdd.zipWithIndex().map(map_ranking).toDF(["Rank"] + men_df_header).cache()
t2d_rank_df = t2d_df_sorted.rdd.zipWithIndex().map(map_ranking).toDF(["Rank"] + t2d_df_header).cache()
```

Select top hits and check overlapping items

```
chd_rank_50_df = chd_rank_df.limit(50)
men_rank_50_df = men_rank_df.limit(50)
t2d_rank_50_df = t2d_rank_df.limit(50)

# find overlap with epi-dataset
epi_chd_rank_50_df = epi_df.join(chd_rank_50_df, 'markername', 'inner').cache()
epi_men_rank_50_df = epi_df.join(men_rank_50_df, 'markername', 'inner').cache()
epi_t2d_rank_50_df = epi_df.join(t2d_rank_50_df, 'Chr:Position', 'inner').cache()

# count overlapping items
print ("Cardio rank 50 has {0} overlapping hits with epi.".format(epi_chd_rank_50_df.count()))
print ("Diabetes rank 50 has {0} overlapping hits with epi.".format(epi_men_rank_50_df.count()))
print ("Menopause rank 50 has {0} overlapping hits with epi.".format(epi_t2d_rank_50_df.count()))
```

Create dataframe with overlapping items

```python
def create_ranking_table(epi_lit_df):
    data_headers = "SNPid", "cad_p","cad_rank","meno_p","meno_rank","T2D_p","T2D_rank"
    data_matrix = []

    for n in epi_lit_df.rdd.collect():
        marker = n["markername"]
        chr_ps = n["Chr:Position"]
        query_cata = chd_rank_df.filter(chd_rank_df["markername"] == marker).select("p_dgc", "rank").first()
        query_meno = men_rank_df.filter(men_rank_df["markername"] == marker).select("p", "rank").first()
        query_meta = t2d_rank_df.filter(t2d_rank_df["Chr:Position"] == chr_ps).select("P-value", "rank").first()

        try:
            data_matrix.append((marker, query_cata[0], query_cata[1], query_meno[0], query_meno[1],
                                query_meta[0], query_meta[1]))
        except TypeError:
            continue

    return sqlContext.createDataFrame(data_matrix, data_headers)

epi_chd_rank_50_table = create_ranking_table(epi_chd_rank_50_df).orderBy("cad_p").cache()
epi_men_rank_50_table = create_ranking_table(epi_men_rank_50_df).orderBy("meno_p").cache()
epi_t2d_rank_50_table = create_ranking_table(epi_t2d_rank_50_df).orderBy("T2D_rank").cache()

# display top 50 overlap
print ("=== {0} ===".format("Cardio"))
epi_chd_rank_50_table.show()

print ("=== {0} ===".format("Menopause"))
epi_men_rank_50_table.show()

print ("=== {0} ===".format("Diabetes"))
epi_t2d_rank_50_table.show()
```

9. Lecture Notes

Synthesis and Trends

- **Role of Data science and its impacts**
- **Knowledge DIscover Process** --> Applied Data Science with the CRISP DM method
- **Data analystics** --> traps in big data analysis: p-values & multiple testing, replicability, overfitting and constructing validity

Knowledge Discovery processes in applied Data Science

- 4Vs: volume, velocity, veracity and Variety
- SQL vs. NoSQl
- Big data vs Data warehousing: Bottm up --> from observation to theory is BD, Top-down is DWH; cold path: historical Data Store vs. Real Time hot path predicitve model
- Ethics

Data platform

- On premise: GOogle chrome, MS office
- Infrastrucutre as a service: google compute engine, azure VM
- Platform as a service: Google app Engine or Azure's Cortana
- Software as a service: Google apps or MS Office 365

Alpha Go Zero
Reinforcement learning which learns solely form self-play

Self & scalability
Deployability
Applicability
Explainagiblity
Manageability - TensorFlow, PyTorch, Docker

Data govenrmance, Data operations management and Data development drive all other decision of the Data Science topics