text → "rows" → Array of arrays ①
sep='\n'. ↳ Single array └→ map (works on these arrays based iterable)

dataframe ↑
↳ Rows (values, idx).

—o—x—o—

FILTER.

↳ Applied on each row of the parent array. ↳ Array ↳ Removes the entire array if not passed

rdd. Filter (lambda x : True).
↳ All pass filter.

~~Map~~ Filter is applied of iterable ↳ Here Array of (Arrays)
↳ Applied on (works on object inside the parent/wrapper iterable)

—o—x—o—

Distinct
↳ Returns unique objects inside array
~~Left to right~~

→ rdd. flatmap (lambda x : x.split(" ")) . distinct()
. collect()

(2) Left to Right → Chaining ∴ new instances created + method used.

—o— x —o—

Groupby key.
↳ Requires "input" in a particular format
INPUT :- ↳ Array of tuples. (key, value) ↳ Tuple.
↳ of (key, val)
↳ To attain this we need to analyse + use flatmap and maps

↳ Group/Aggregate by key.
+ Apply some function.
↳ map.
or
mapvalues.

Returns → Key, ~~Statistic~~ Estimate of the iterable.

③ Reduce By Key (fn)

∴ Requisites
- ↳ Input ⇒ Iterable of tuples
  - ↳ (key, val)
- +
- ↳ Function
  - ↳ Reducer → lambda x, y : f(x, y)
    - ↳ One value per key.

Always chk. for the kind of object one/u are applying ur fn on.
- ↳ string
- array
- integer.

Str. " Obj — Create appropriate fn for each.

INPUT

Ex: txt_file. —I→ split (' ')
- ↳ '/n' sep.
- |→ object = Array
- ↓ 2 possibilities.
  - MAP → Array of arrays.
    - → Map(Lambda x : sum(x)) → Array of tuple. [(Key, est)].
  - FLATMAP → Array of strings.
    - ↓ MAP [la x : (x, 1)]
    - ↓ Array of tuples.

Array of tuple ↳(key, iterable) ⟵ GroupBy Key (). ~~map(lambda x : sum(x)).~~ ⟵

[(key, final_estimate)_ . - - ()] Array of tuples. ⟵ Reduce By Key (lambda x, y : x + y). ⟵ (Requisite of GroupByKey, ReduceByKey.)

→ ...

ACTIONS → Will always run the architecture
↳ Needs/consumes processing mem$\boxed{4}$ory.
↳ Methods that renders the pipeline.
↳ Ex. → ∘ Collect ()
↳ List all the values in the wrapper iterable. ↱ object

∘ Count():
↳ length of the wrapper list.

∘ countByValue () [Map (lam x : (x, 1) → groupByKey () → Map (lamda x: x sum (x))]
↳ Value_counts().

∘ saveAsText.File ()
↳ save the file to the specified path.
↳ Can also create folder. ~~In existing folder.~~ ↱ saved.
↳ mkdir ().

∘→ Spark rdd instance ↱ p (by default) = 2. ↱∴ 2 partition for one action
↳ By default creates partitions/clusters of EC2 on which it perform transformations.

Will save these partitions separately.

key

→ We can increase or decrease the # of clusters for working on the data. (5)

↳ rdd.repartition(number_of_clusters = 5)

rdd.getNumPartitions~~X~~() 5 clusters. ↵
↳ ~~Attri~~ Method
↳ count of clusters.

↳ ∴ Now process will be divided across 5 nodes.

o Repartition → Fn for Rdd
↳ Method → Non-action based → Will only affect the architecture.

↓

How to distribute memory for use.

Each Rdd instance can have different ↵

Works like Git hub commits.

o set of functions / transformation

o Clusters in which the data will be distributed.

Returns → Key ,

Average → we need → count of keys + sum of keys ⑥
↳ Big streaming data

groupBy or ReduceGroupBy needs input in tuple of (key, val) format.
↳ format.

This val will itself be a tuple of sum, count.

INPUT (tx → sep = '\n' line.
Row → Name, Rate)
↳ sep = ','

Map(lambda x : (x.split(',')[0], (int(x.split(',')[1], 1)))

Array of tuples
↳ name, (tuple of sum, ct)

ReduceGroupBy(lambda x, y: (x[0]+y[0], x[1]+y[1]))
↳ Will fetch/aggregate ~~simil~~ same keys + Apply fn on second part of the tuple.

→ (name, (sum, ct))

Array of tuples.

Map(lambda x : (x[0], x[1][0] / x[1][1]))

→ (name, avg).

Array of tuples-