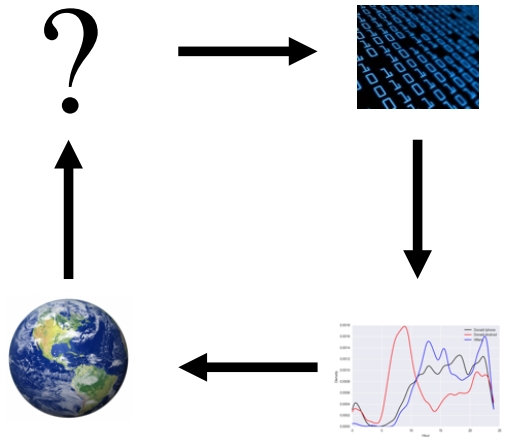
Review

DS 100, Spring 2017

Slides by:

Joe Hellerstein

hellerstein@berkeley.edu



Format of Exam

- When: Thursday 5/11 3PM
- Where: Genetics and Plant Bio 100
- How: Very similar to midterm in structure
- Cheat Sheets: 2!
- Topic areas: See syllabus.

Topics Covered in This Deck

- Relational Algebra and SQL
- Pandas
- Big Data: Data Warehousing, Map-Reduce, Spark
- Data Wrangling

Context and Connections

- We have seen a variety of data manipulation tasks
  - Both data "wrangling" and analysis
  - Diverse languages and frameworks
    - Pandas, SQL, MapReduce, Spark
- Check your fundamentals, study commonalities!
  - Matrix and set operations
  - Relational Algebra & Map/Reduce
    - Can you translated the operators from one into the other?
  - What operations appear frequently?
    - E.g. joins, groupby/sum/product, filtering on rows and columns
    - Relational algebra and linear algebra!

Relational Algebra & SQL

Relational Model & SQL

- Key Ideas:
  - Schemas, instances, queries, views, insert/delete/update
  - Relational algebra operators
  - SQL as a *declarative* language: everything referenced *by value*
    - *Not by "index" as in arrays or dataframes!*
  - Many big data features under the covers: parallel execution, user-defined functions and aggregates
- What you should know:
  - Relational/SQL model basics
  - Relational algebra and basic equivalences
  - Basic SQL structure and syntax. Recognize incorrect logic (as opposed to syntax errors)
- What you don't need to know:
  - You will not be asked to compose queries from scratch
- How to study?
  - Review HW4, SQL notebook files from lecture

Relational Model

- Key Idea:
  - Schema: describes attribute (column) names and types, keys
  - Instance: a *set* (unordered, duplicate-free) of tuples (rows)
- Review the Boat Club from class

```
CREATE TABLE sailors(sid integer PRIMARY KEY, sname text,
                     rating integer, age float);

CREATE TABLE boats(bid integer PRIMARY KEY, bname text, color text);

CREATE TABLE reserves(sid integer, bid integer, day date,
                      PRIMARY KEY (sid, bid, day),
                      FOREIGN KEY sid REFERENCES sailors,
                      FOREIGN KEY bid REFERENCES boats);
```

Boats

bid	bname	color
101	Interlake	blue
102	Interlake	red
104	Marine	red
103	Clipper	green

Reserves

sid	bid	day
28	101	10/10/16
58	103	11/12/96

Sailors

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Note: primary keys underlined

Relational Algebra Operators

Unary Operators: *operate on **single** relation instance*

- **Projection (π):** *Retains only desired columns* (vertical)
- **Selection (σ):** *Selects a subset of rows (horizontal)*
- **Renaming (ρ):** *Rename attributes and relations.*

Binary Operators: *operate on **pairs** of relation instances*

- **Union ($\cup$):** *Tuples in r1 or in r2.*
- **Intersection ($\cap$):** *Tuples in r1 and in r2.*
- **Set-difference (—):** *Tuples in r1, but not in r2.*
- **Cross-product ($\times$):** *Allows us to combine two relations.*
- **Joins ($\bowtie_\theta$, $\bowtie$):** *Combine relations that satisfy predicates*

As you review this material, do you see how it relates to SQL, Pandas and Spark?

SQL Data Model

- Basically relational, but "multisets"
  - I.e. can have duplicate rows in a table, and they matter
    - E.g. for COUNT, AVG, etc.
  - I.e. some tables have *no* primary key!
- FOREIGN KEYs

SQL

A declarative language: say "what" you want in the output, not "how" to get it.

DDL

```
CREATE TABLE (<col> <type>, ...);
```

Understand PRIMARY KEY, FOREIGN KEY clauses.

```
INSERT INTO TABLE values (…), (…), …;
INSERT INTO TABLE SELECT …;
```

```
UPDATE TABLE SET … WHERE …;
```

note: value-based references in UPDATE!

DML

```
SELECT ...
  FROM ...
[WHERE ...]
[GROUP BY ...]
[HAVING ...]
[ORDER BY ...]
[LIMIT ...];
```

SQL cheat sheet

REBELLABS by ZEROTURNAROUND

Basic Queries

```
-- filter your columns
   SELECT col1, col2, col3, ... FROM table1
-- filter the rows
   WHERE col4 = 1 AND col5 = 2
-- aggregate the data
   GROUP by ...
-- limit aggregated data
   HAVING count(*) > 1
-- order of the results
   ORDER BY col2
```

Useful keywords for **SELECTS**:

DISTINCT - return unique results

BETWEEN a **AND** b - limit the range, the values can be numbers, text, or dates

LIKE - pattern search within the column text

IN (a, b, c) - check if the value is contained among given.

Data Modification

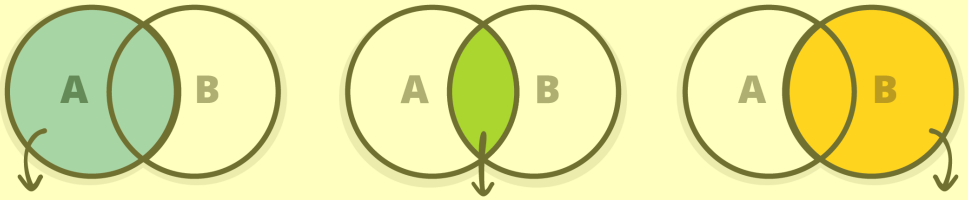
```
-- update specific data with the WHERE clause
   UPDATE table1 SET col1 = 1 WHERE col2 = 2
-- insert values manually
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      VALUES (1, 'Rebel', 'Labs');
-- or by using the results of a query
   INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
      SELECT id, last_name, first_name FROM table2
```

Views

A **VIEW** is a virtual table, which is a result of a query.
They can be used to create virtual tables of complex queries.

```
CREATE VIEW view1 AS
SELECT col1, col2
FROM table1
WHERE ...
```

The Joy of JOINs



LEFT OUTER JOIN - *all rows from table A, even if they do not exist in table B*

INNER JOIN - *fetch the results that exist in both tables*

RIGHT OUTER JOIN - *all rows from table B, even if they do not exist in table A*

Updates on JOINed Queries

You can use **JOIN**s in your **UPDATE**s

```
UPDATE t1 SET a = 1
FROM table1 t1 JOIN table2 t2 ON t1.id = t2.t1_id
WHERE t1.col1 = 0 AND t2.col2 IS NULL;
```

NB! Use database specific syntax, it might be faster!

Semi JOINs

You can use subqueries instead of **JOIN**s:

```
SELECT col1, col2 FROM table1 WHERE id IN
   (SELECT t1_id FROM table2 WHERE date >
      CURRENT_TIMESTAMP)
```

Indexes

If you query by a column, index it!

```
CREATE INDEX index1 ON table1 (col1)
```

Don't forget:

Avoid overlapping indexes

Avoid indexing on too many columns

Indexes can speed up **DELETE** and **UPDATE** operations

Useful Utility Functions

```
-- convert strings to dates:
   TO_DATE (Oracle, PostgreSQL), STR_TO_DATE (MySQL)
-- return the first non-NULL argument:
   COALESCE (col1, col2, "default value")
-- return current time:
   CURRENT_TIMESTAMP
-- compute set operations on two result sets
   SELECT col1, col2 FROM table1
   UNION / EXCEPT / INTERSECT
   SELECT col3, col4 FROM table2;
```

Union - returns data from both queries

Except - rows from the first query that are not present in the second query

Intersect - rows that are returned from both queries

Reporting

Use aggregation functions

COUNT - return the number of rows

SUM - cumulate the values

AVG - return the average for the group

MIN / MAX - smallest / largest value

BROUGHT TO YOU BY XRebel

https://zeroturnaround.com/rebellabs/sql-cheat-sheet/

Also

- TABLESAMPLE BERNOULLI(…)
- Scalar functions (like "map") in SELECT and WHERE
  - Arithmetic, Math functions, string functions…
- Aggregate functions (like "reduce") in SELECT and HAVING
  - count(), sum(), average(), stddev()
  - Note: DISTINCT clause inside aggregates
- Table functions: generate_series()

Pandas and Numpy

Arrays and DataFrames

- Key Ideas:
  - Array is a multi-d array; single type for all cells
  - Pandas DataFrame is like a relation (table); single type per column
    - Unlike relational model, fixed order to rows (allowing array-style indexing)
- What you should know:
  - Basic numpy array handling
  - Key pandas operations (see next slides)
  - Recognize incorrect logic
- What you don't need to know:
  - You will not be asked to write code from scratch
- How to study?
  - Review HW1-3, notebook files from lecture
  - Be able to translate basic SQL block above into Pandas

Topics

- Data Frame & Numpy Arrays
  - Subsets
  - Groupby/Agg
  - Joining

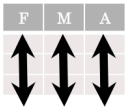
https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf

Data Wrangling with pandas Cheat Sheet

http://pandas.pydata.org

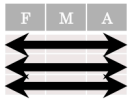
Tidy Data – A foundation for wrangling in pandas

In a tidy data set:



Each **variable** is saved in its own **column**

&



Each **observation** is saved in its own **row**

Tidy data complements pandas's **vectorized operations**. pandas will automatically preserve observations as you manipulate variables. No other format works as intuitively with pandas.

M * A

Syntax – Creating DataFrames

	a	b	c
1	4	7	10
2	5	8	11
3	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
        index = [1, 2, 3])
```

Specify values for each column.

```
df = pd.DataFrame(
    [[4, 7, 10],
     [5, 8, 11],
     [6, 9, 12]],
    index=[1, 2, 3],
    columns=['a', 'b', 'c'])
```

Specify values for each row.

		a	b	c
n	v			
d	1	4	7	10
	2	5	8	11
e	2	6	9	12

```
df = pd.DataFrame(
        {"a" : [4 ,5, 6],
         "b" : [7, 8, 9],
         "c" : [10, 11, 12]},
index = pd.MultiIndex.from_tuples(
        [('d',1),('d',2),('e',2)],
         names=['n','v'])))
```

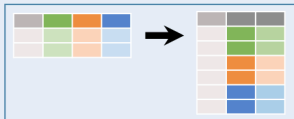
Create DataFrame with a MultiIndex

Method Chaining

Most pandas methods return a DataFrame so that another pandas method can be applied to the result. This improves readability of code.

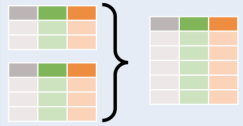
```
df = (pd.melt(df)
        .rename(columns={
                'variable' : 'var',
                'value' : 'val'})
        .query('val >= 200')
    )
```

Reshaping Data – Change the layout of a data set



pd.melt(df)
Gather columns into rows.

df.pivot(columns='var', values='val')
Spread rows into columns.



pd.concat([df1,df2])
Append rows of DataFrames

pd.concat([df1,df2], axis=1)
Append columns of DataFrames

df.sort_values('mpg')
Order rows by values of a column (low to high).

df.sort_values('mpg',ascending=False)
Order rows by values of a column (high to low).

df.rename(columns = {'y':'year'})
Rename the columns of a DataFrame

df.sort_index()
Sort the index of a DataFrame

df.reset_index()
Reset index of DataFrame to row numbers, moving index to columns.

df.drop(['Length','Height'], axis=1)
Drop columns from DataFrame

Subset Observations (Rows)

df[df.Length > 7]
Extract rows that meet logical criteria.

df.drop_duplicates()
Remove duplicate rows (only considers columns).

df.head(n)
Select first n rows.

df.tail(n)
Select last n rows.

df.sample(frac=0.5)
Randomly select fraction of rows.

df.sample(n=10)
Randomly select n rows.

df.iloc[10:20]
Select rows by position.

df.nlargest(n, 'value')
Select and order top n entries.

df.nsmallest(n, 'value')
Select and order bottom n entries.

Subset Variables (Columns)

df[['width','length','species']]
Select multiple columns with specific names.

df['width'] *or* **df.width**
Select single column with specific name.

df.filter(regex='*regex*')
Select columns whose name matches regular expression *regex*.

regex (Regular Expressions) Examples

'\.'	Matches strings containing a period '.'
'Length$'	Matches strings ending with word 'Length'
'^Sepal'	Matches strings beginning with the word 'Sepal'
'^x[1-5]$'	Matches strings beginning with 'x' and ending with 1,2,3,4,5
''^(?!Species$).*'	Matches strings except the string 'Species'

df.loc[:,'x2':'x4']
Select all columns between x2 and x4 (inclusive).

df.iloc[:,[1,2,5]]
Select columns in positions 1, 2 and 5 (first column is 0).

df.loc[df['a'] > 10, ['a','c']]
Select rows meeting logical condition, and only the specific columns .

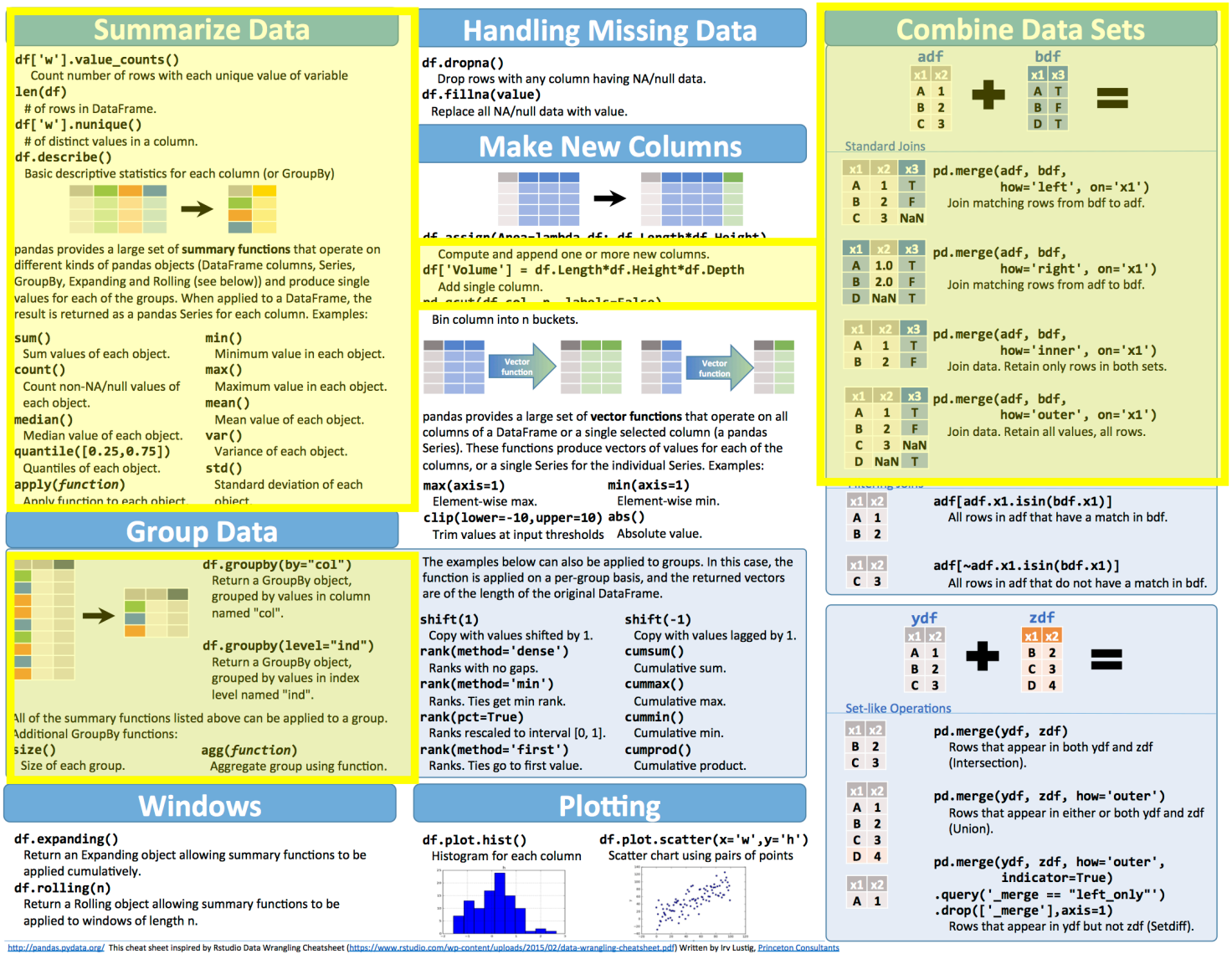
Logic in Python (and pandas)

<	Less than	!=	Not equal to
>	Greater than	df.column.isin(*values*)	Group membership
==	Equals	pd.isnull(*obj*)	Is NaN
<=	Less than or equals	pd.notnull(*obj*)	Is not NaN
>=	Greater than or equals	&,|,~,^,df.any(),df.all()	Logical and, or, not, xor, any, all

Topics

- Data Frame & Numpy Arrays
  - Subsets
  - Indexing
  - Groupby/Agg
  - Joining

https://github.com/pandas-dev/pandas/blob/master/doc/cheatsheet/Pandas_Cheat_Sheet.pdf



Summarize Data

`df['w'].value_counts()`
Count number of rows with each unique value of variable

`len(df)`
# of rows in DataFrame.

`df['w'].nunique()`
# of distinct values in a column.

`df.describe()`
Basic descriptive statistics for each column (or GroupBy)

pandas provides a large set of **summary functions** that operate on different kinds of pandas objects (DataFrame columns, Series, GroupBy, Expanding and Rolling (see below)) and produce single values for each of the groups. When applied to a DataFrame, the result is returned as a pandas Series for each column. Examples:

`sum()` Sum values of each object.
`count()` Count non-NA/null values of each object.
`median()` Median value of each object.
`quantile([0.25,0.75])` Quantiles of each object.
`apply(function)` Apply function to each object.
`min()` Minimum value in each object.
`max()` Maximum value in each object.
`mean()` Mean value of each object.
`var()` Variance of each object.
`std()` Standard deviation of each object.

Handling Missing Data

`df.dropna()`
Drop rows with any column having NA/null data.

`df.fillna(value)`
Replace all NA/null data with value.

Make New Columns

`df.assign(Area=lambda df: df.Length*df.Height)`
Compute and append one or more new columns.

`df['Volume'] = df.Length*df.Height*df.Depth`
Add single column.

`pd.qcut(df.col, n, labels=False)`
Bin column into n buckets.

Vector function → Vector function

pandas provides a large set of **vector functions** that operate on all columns of a DataFrame or a single selected column (a pandas Series). These functions produce vectors of values for each of the columns, or a single Series for the individual Series. Examples:

`max(axis=1)` Element-wise max.
`min(axis=1)` Element-wise min.
`clip(lower=-10,upper=10)` Trim values at input thresholds
`abs()` Absolute value.

Group Data

`df.groupby(by="col")`
Return a GroupBy object, grouped by values in column named "col".

`df.groupby(level="ind")`
Return a GroupBy object, grouped by values in index level named "ind".

All of the summary functions listed above can be applied to a group.
Additional GroupBy functions:

`size()` Size of each group.
`agg(function)` Aggregate group using function.

The examples below can also be applied to groups. In this case, the function is applied on a per-group basis, and the returned vectors are of the length of the original DataFrame.

`shift(1)` Copy with values shifted by 1.
`rank(method='dense')` Ranks with no gaps.
`rank(method='min')` Ranks. Ties get min rank.
`rank(pct=True)` Ranks rescaled to interval [0, 1].
`rank(method='first')` Ranks. Ties go to first value.
`shift(-1)` Copy with values lagged by 1.
`cumsum()` Cumulative sum.
`cummax()` Cumulative max.
`cummin()` Cumulative min.
`cumprod()` Cumulative product.

Windows

`df.expanding()`
Return an Expanding object allowing summary functions to be applied cumulatively.

`df.rolling(n)`
Return a Rolling object allowing summary functions to be applied to windows of length n.

Plotting

`df.plot.hist()`
Histogram for each column

`df.plot.scatter(x='w',y='h')`
Scatter chart using pairs of points

Combine Data Sets

adf + bdf =

Standard Joins

`pd.merge(adf, bdf, how='left', on='x1')`
Join matching rows from bdf to adf.

`pd.merge(adf, bdf, how='right', on='x1')`
Join matching rows from adf to bdf.

`pd.merge(adf, bdf, how='inner', on='x1')`
Join data. Retain only rows in both sets.

`pd.merge(adf, bdf, how='outer', on='x1')`
Join data. Retain all values, all rows.

`adf[adf.x1.isin(bdf.x1)]`
All rows in adf that have a match in bdf.

`adf[~adf.x1.isin(bdf.x1)]`
All rows in adf that do not have a match in bdf.

ydf + zdf =

Set-like Operations

`pd.merge(ydf, zdf)`
Rows that appear in both ydf and zdf (Intersection).

`pd.merge(ydf, zdf, how='outer')`
Rows that appear in either or both ydf and zdf (Union).

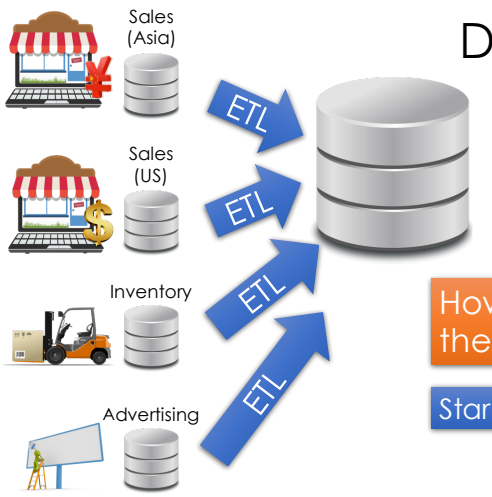
`pd.merge(ydf, zdf, how='outer', indicator=True) .query('_merge == "left_only"') .drop(['_merge'],axis=1)`
Rows that appear in ydf but not zdf (Setdiff).

http://pandas.pydata.org/ This cheat sheet inspired by Rstudio Data Wrangling Cheatsheet (https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf) Written by Irv Lustig, Princeton Consultants

Big Data

Big Data

- Key Ideas:
  - Schema on load (data warehousing) vs. on use (data lakes)
  - Fault-tolerant file systems
  - Data-parallel computing (e.g. MapReduce)
  - Statistical Query Pattern
- What you should know:
  - Star schema design
  - File system data layouts and failure handling
  - Map vs. Reduce
  - Spark Transformations vs. Actions
  - How to compute basic linear algebra operations in statistical query pattern
- What you don't need to know:
  - You will not be asked to write code from scratch
  - Fault tolerance in MapReduce
- How to study?
  - Review HW 7 and lecture notebooks



Data Warehouse

Collects and organizes historical data from multiple sources

How is data organized in the Data Warehouse?

Star Schemas

Data Lake*

Store a copy of all the data

- in one place
- in its original "natural" form

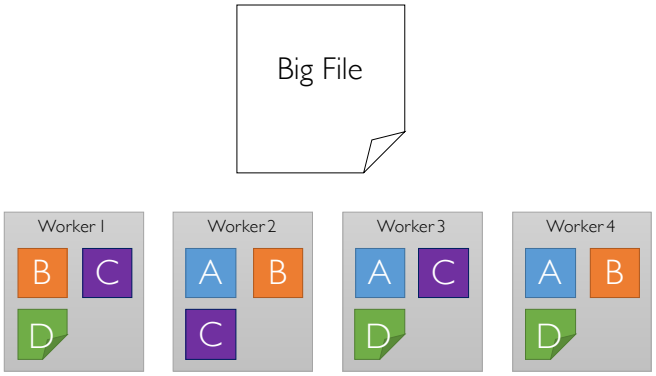
Enable data consumers to choose how to transform and use data.

- *Schema on Read*

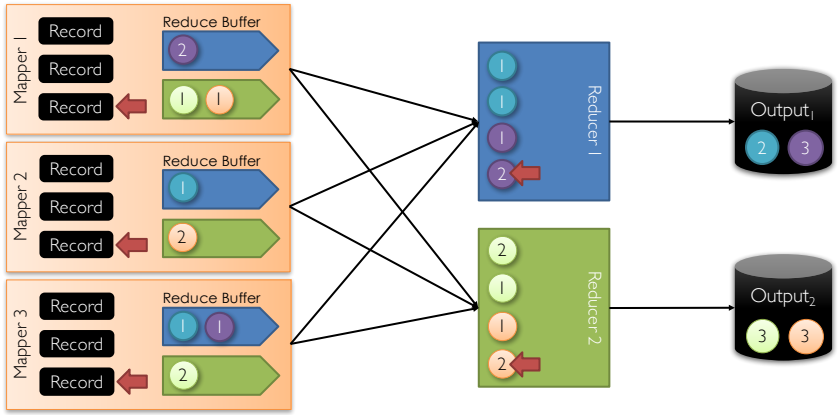
Enabled by new Tools:
Map-Reduce & Distributed Filesystems

What could go wrong?

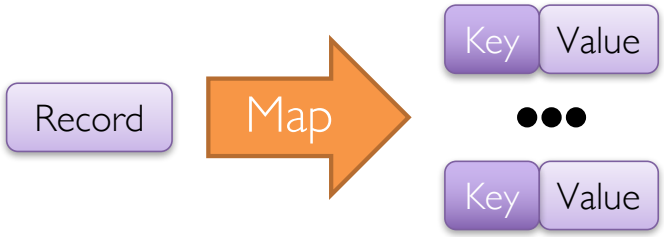
Fault Tolerant Distributed File Systems (e.g., HDFS)



The Map Reduce System



Parallel Computation



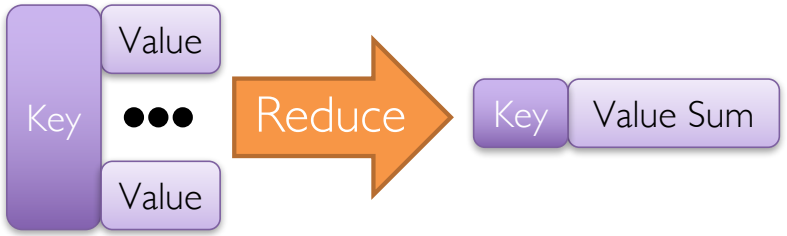
Example: *Word-Count*

```
Map(docRecord) {
  for (word in docRecord) {
    emit (word, 1)
  }
}
```

Key Value

Group By Key

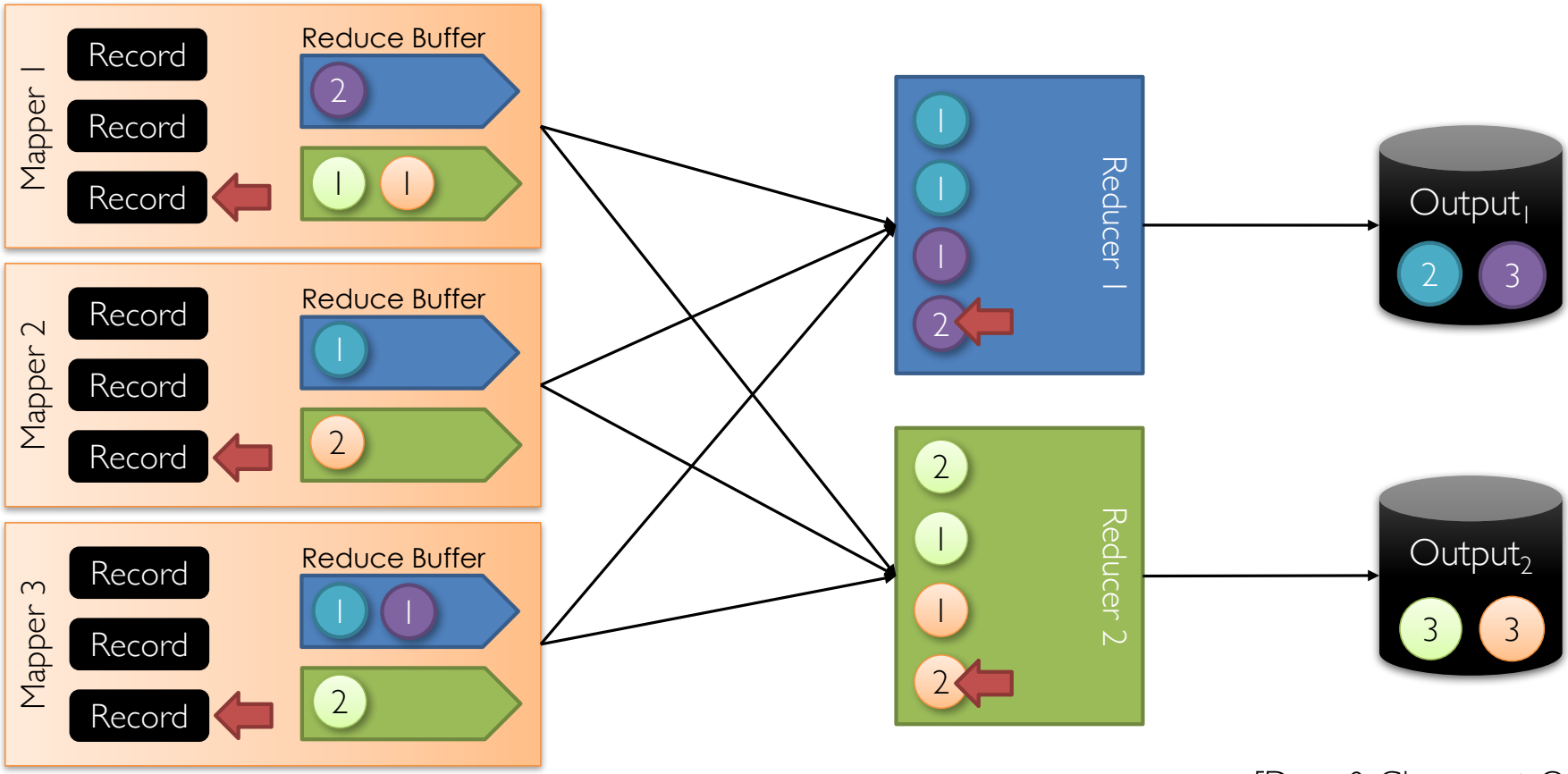
Parallel Computation




```
Reduce(word, counts) {
  emit (word, SUM(counts))
}
```

Map: Idempotent
Reduce: Commutative and Associative

The Map Reduce System

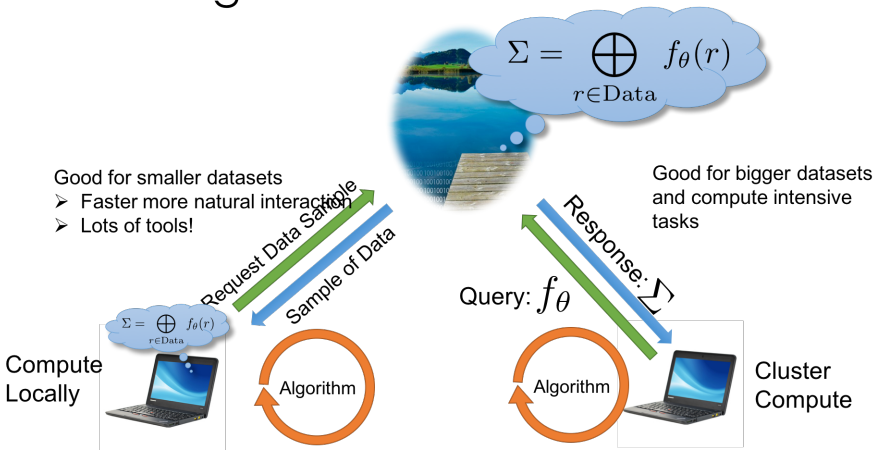


[Dean & Ghemawat, OSDI'04]

Spark

- A newer MapReduce framework
  - With higher-level programming APIs: **Dataframes & SQL**
- Data representations:
  - *Resilient Distributed Dataset (RDD):* A collection of **objects**
    - Objects could be text, images, python dictionaries …
  - *DataFrame:* A collection of **rows** with common schema
- Computation
  - **Transformations:** users apply functions to the distributed collections which return new distributed collections
    - E.g., map, flatmap, filter, reduceByKey
  - **Actions**: users apply functions which return values:
    - E.g., count, sum, collect, take, first, reduce

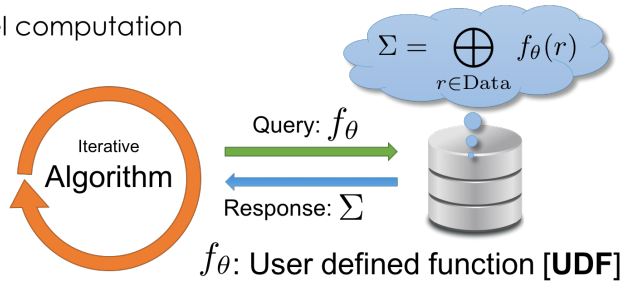
Interacting With the Data



Statistical Query Pattern
Common Machine Learning Pattern

Computing aggregates of user defined functions

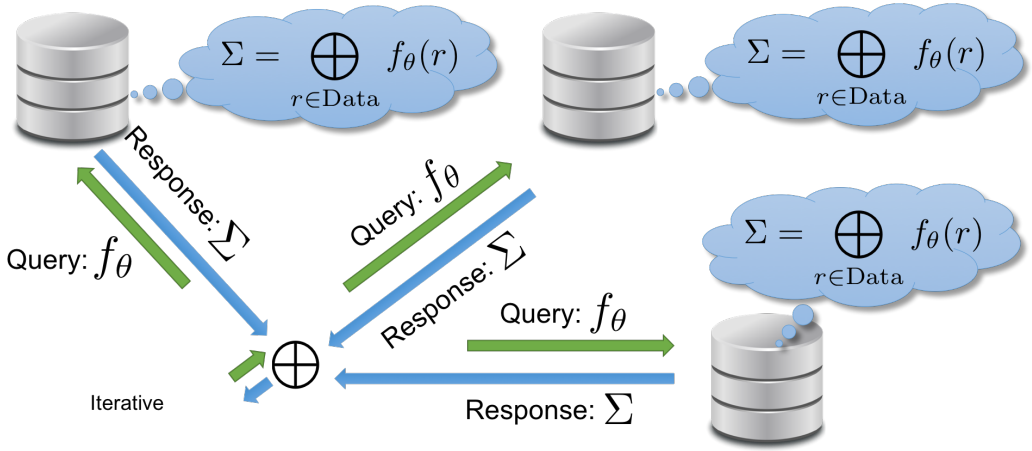
Data-Parallel computation



f_θ: User defined function [**UDF**]

$\bigoplus$: User defined aggregate [**UDA**]

D. Caragea et al., *A Framework for Learning from Distributed Data Using Sufficient Statistics and Its Application to Learning Decision Trees*. Int. J. Hybrid Intell. Syst. 2004



Data Wrangling

Data Wrangling

- Key Ideas:
  - Wrangling output determines your downstream analytics
  - Assessing data quality like querying + vis
  - Data transformation also like querying!
  - A petri dish for many of the topics in class

- What you should know:
  - Structural issues: arrays, relations, and converting between
  - Granularity: assessing it (keys) and changing it (grouping)
  - Faithfulness: finding and handling outliers
  - Basic UNIX commands

- What you don't need to know
  - UNIX command flags
  - Trifacta Wrangler
  - Memorizing "Structure, Granularity, Faithfulness, Temporality, Scope" (that's just to organize the ideas)
  - Live-coding Pandas/Python details like "read_csv" or strip() or head()

- How to study?
  - Lecture notes
  - HW3

Rough Guide to Topics

- **Structure:** **the "shape" of a data file**
- **Granularity:** **how fine/coarse is each datum**
- **Faithfulness:** **how well does the data capture "reality"**
- Temporality: how is the data situated in time
- Scope: how (in)complete is the data

We didn't really discuss temporality/scope much

Structure

- Relations and Relational languages (algebra, SQL)
- Arrays and Linear Algebra
- Coarse structure
  - Record and field delimiters
  - Common schemas
  - Nested?
- Value encoding: nominal vs. ordinal vs. quantitative
- See connections to SQL and DataFrames, n-d Arrays

Granularity

- Keys as an indicator of granularity
- Granularity can be coarsened via groupby
  - Grouping columns become a primary key of result
- Granularity can be refined via join
  - With more detailed tables
  - Foreign keys
  - Understand the # of matches for various joins

Faithfulness

- Data *in context*
  - Application context
    - E.g. human beings are unlikely to be 1000 years old
    - Constraints: relation to "types" or "domains"
      - E.g. "123456789" is not a valid zip code
      - E.g. the word "spunow" is not in the english dictionary
  - Cross-record context
    - Detecting Outliers
      - "Center": e.g. average and median
      - "Spread": e.g. stdev, inter-quartile range
    - Resolving Outliers
      - Trimming: setting outlier values to N/A or NULL
      - Winsorizing: setting outlier values to the nearest non-outlier value