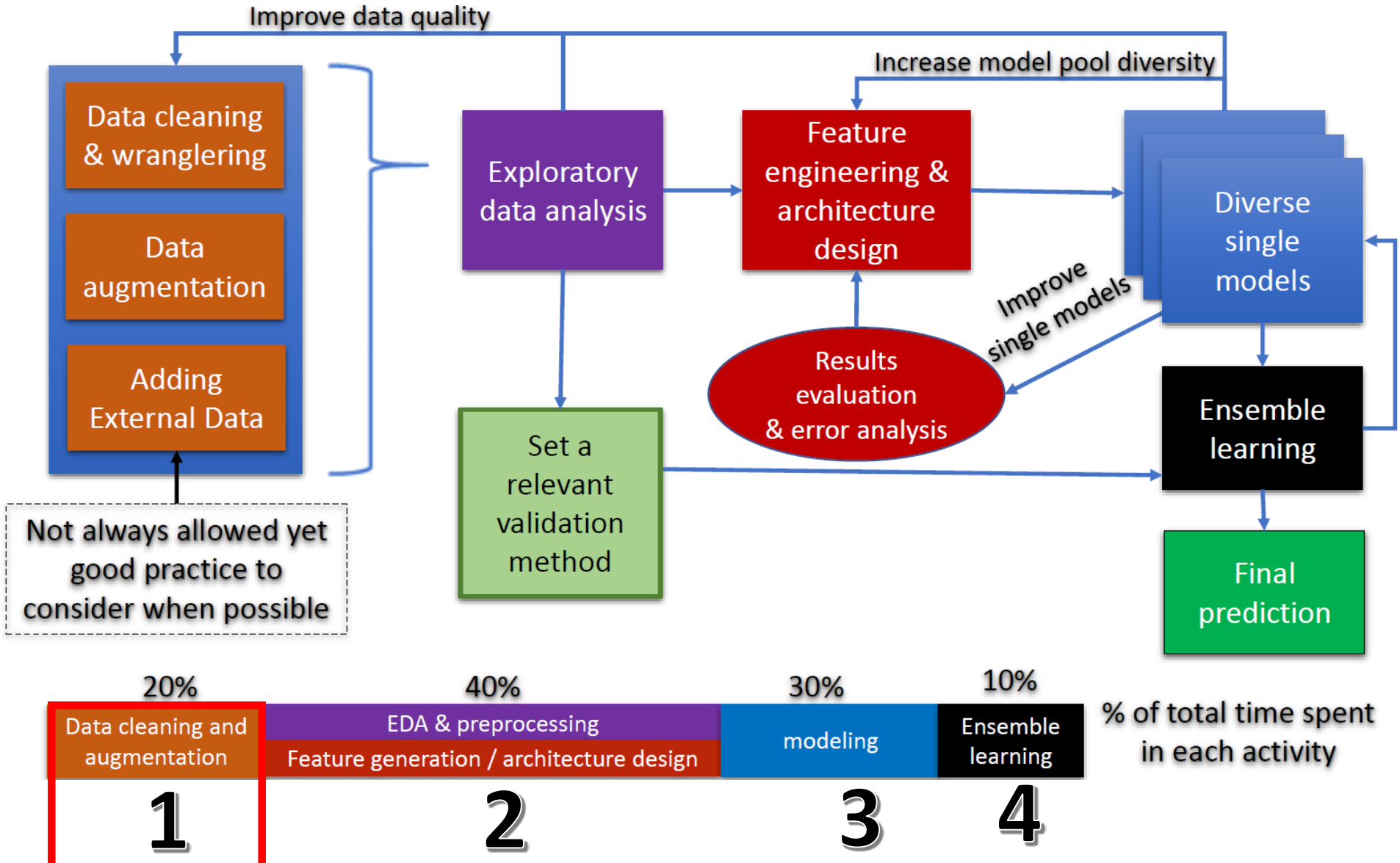
Data Cleaning

Dr Alexander A S Gunawan
aagung@binus.edu
http://sigmetris.com
08175001010

Data Science Project Flow



Cleaning data

- Prepare data for analysis
- Data almost never comes in clean
- Diagnose your data for problems

Common data problems

- Inconsistent column names
- Missing data
- Outliers
- Duplicate rows
- Untidy
- Need to process columns
- Column types can signal unexpected data values

Unclean data

	Continent	Country	female literacy	fertility	population
0	ASI	Chine	90.5	1.769	1.324655e+09
1	ASI	Inde	50.8	2.682	1.139965e+09
2	NAM	USA	99.0	2.077	3.040600e+08
3	ASI	Indonésie	88.8	2.132	2.273451e+08
4	LAT	Brésil	90.2	1.827	NaN

- Column name inconsistencies
- Missing data
- Country names are in French

Visually inspect

```
In [3]: df.head()
Out[3]:
 Continent    Country   female literacy  fertility   population
0       ASI       Chine             90.5      1.769  1.324655e+09
1       ASI        Inde             50.8      2.682  1.139965e+09
2       NAM         USA             99.0      2.077  3.040600e+08
3       ASI   Indonésie             88.8      2.132  2.273451e+08
4       LAT      Brésil             90.2      1.827           NaN

In [4]: df.tail()
Out[4]:
 Continent          Country      female literacy  fertility    population
0        AF  Sao Tomé-et-Principe             90.5      1.769  1.324655e+09
1       LAT                 Aruba             50.8      2.682  1.139965e+09
2       ASI                 Tonga             99.0      2.077  3.040600e+08
3       OCE             Australia             88.8      2.132  2.273451e+08
4       OCE                Sweden             90.2      1.827           NaN
```

Visually inspect

```
In [5]: df.columns
Out[5]: Index(['Continent', 'Country ', 'female literacy',
'fertility', 'population'], dtype='object')

In [6]: df.shape
Out[6]: (164, 5)

In [7]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 5 columns):
Continent           164 non-null object
Country             164 non-null object
female literacy     164 non-null float64
fertility           164 non-null object
population          122 non-null float64
dtypes float64(2), object(3)
memory usage: 6.5+ KB
```

Data type of each column

```
In [1]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 5 columns):
continent           164 non-null object
country             164 non-null object
female literacy     164 non-null float64
fertility           164 non-null object
population          122 non-null float64
dtypes float64(2), object(3)
memory usage: 6.5+ KB
```

Frequency counts: population

```
In [6]: df.population.value_counts(dropna=False).head()
Out[6]:
NaN             42
5.667325e+06     1
3.773100e+06     1
1.333388e+06     1
1.661115e+08     1
Name: population, dtype: int64
```

Tidy data

- "Tidy Data" paper by Hadley Wickham, PhD
- Formalize the way we describe the shape of data
- Gives us a goal when formatting our data
- "Standard way to organize data values within a dataset"

Principles of tidy data

- Columns represent separate variables
- Rows represent individual observations
- Observational units form tables

	name	treatment a	treatment b
0	Daniel	-	42
1	John	12	31
2	Jane	24	27

Converting to tidy data

	name	treatment a	treatment b
0	Daniel	-	42
1	John	12	31
2	Jane	24	27

	name	treatment	value
0	Daniel	treatment a	-
1	John	treatment a	12
2	Jane	treatment a	24
3	Daniel	treatment b	42
4	John	treatment b	31
5	Jane	treatment b	27

- Better for reporting vs. better for analysis
- Tidy data makes it easier to fix common data problems

Converting to tidy data

- The data problem we are trying to fix:
  - Columns containing values, instead of variables
- Solution: pd.melt()

Melting

```
In [2]: pd.melt(frame=df, id_vars='name',
   ...:         value_vars=['treatment a', 'treatment b'],
   ...:         var_name='treatment', value_name='result')
Out[2]:
     name     treatment  result
0  Daniel  treatment a       _
1    John  treatment a      12
2    Jane  treatment a      24
3  Daniel  treatment b      42
4    John  treatment b      31
5    Jane  treatment b      27
```

Pivot: un-melting data

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5
2	2010-02-02	tmax	27.3
3	2010-02-02	tmin	14.4

element	tmax	tmin
date		
2010-01-30	27.8	14.5
2010-02-02	27.3	14.4

Using pivot when you have duplicate entries

```
In [3]: import numpy as np

In [4]: weather2_tidy = weather.pivot(values='value',
   ...:                               index='date',
   ...:                               columns='element')
Out[4]:
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-9-2962bb23f5a3> in <module>()
      1 weather2_tidy = weather2.pivot(values='value',
      2                                index='date',
----> 3                                columns='element')
ValueError: Index contains duplicate entries, cannot reshape
```

Pivot

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5
2	2010-02-02	tmax	27.3
3	2010-02-02	tmin	14.4

	date	element	value
0	2010-01-30	tmax	27.8
1	2010-01-30	tmin	14.5
2	2010-02-02	tmax	27.3
3	2010-02-02	tmin	14.4
4	2010-02-02	tmin	16.4

Pivot table

- Has a parameter that specifies how to deal with duplicate values
- Example: Can aggregate the duplicate values by taking their average

```
In [5]: weather2_tidy = weather.pivot_table(values='value',
   ...:                                     index='date',
   ...:                                     columns='element',
   ...:                                     aggfunc=np.mean)
Out[5]:
element     tmax tmin
date
2010-01-30  27.8 14.5
2010-02-02  27.3 15.4
```

Concatenating many files

- Leverage Python's features with data cleaning in pandas
- In order to concatenate DataFrames:
  - They must be in a list
  - Can individually load if there are a few datasets
  - But what if there are thousands?
- Solution: glob function to find files based on a pattern

Globbing

- Pattern matching for file names
- Wildcards: * ?
  - Any csv file: *.csv
  - Any single character: file_?.csv
- Returns a list of file names
- Can use this list to load into separate DataFrames

Merging data

- Similar to joining tables in SQL
- Combine disparate datasets based on common columns

	state	population_2016
0	California	39250017
1	Texas	27862596
2	Florida	20612439
3	New York	19745289

	name	ANSI
0	California	CA
1	Florida	FL
2	New York	NY
3	Texas	TX

Merging data

```
In [1]: pd.merge(left=state_populations, right=state_codes,
   ...:          on=None, left_on='state', right_on='name')
Out[1]:
       state   population_2016       name   ANSI
0  California         39250017  California   CA
1       Texas         27862596       Texas   TX
2     Florida         20612439     Florida   FL
3    New York         19745289    New York   NY
```

Duplicate and missing data

Duplicate data

- Can skew results
- '.drop_duplicates()' method

	name	sex	treatment a	treatment b
0	Daniel	male	-	42
1	John	male	12	31
2	Jane	female	24	27
3	Daniel	male	-	42

Drop duplicates

```
In [1]: df = df.drop_duplicates()

In [2]: print(df)
     name     sex treatment a  treatment b
0  Daniel    male            -           42
1    John    male           12           31
2    Jane  female           24           27
```

Missing data

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2.0
1	NaN	1.66	Male	No	Sun	Dinner	3.0
2	21.01	3.50	Male	No	Sun	Dinner	3.0
3	23.68	NaN	Male	No	Sun	Dinner	2.0
4	24.59	3.61	NaN	NaN	Sun	NaN	4.0

- Leave as-is
- Drop them
- Fill missing value

Fill missing values with .fillna()

- Fill with provided value
- Use a summary statistic

Complex cleaning

- Cleaning step requires multiple steps
  - Extract number from string
  - Perform transformation on extracted number
- Python function

Apply

```
In [1]: print(df)
        treatment a  treatment b
Daniel           18           42
John             12           31
Jane             24           27

In [2]: df.apply(np.mean, axis=0)
Out[2]:
treatment a    18.000000
treatment b    33.333333
dtype: float64
```