# DATA 301: Data Cleaning

### Data Cleaning - Outline

- Why
- Missing Values
- Duplicates
- Strings
- Categorical data
- Numerical Data
- Dates

### Why

#### Data is usually messy.

You can minimize some problems

 For surveys, prefer comboboxes populated with a curated list rather than free form text field

#### Some you cannot

- external datasets (like your first project)
- free form text (like a collection of movie reviews)
- Missing and duplicate values
- Sensor data (outliers, missing values)

#### Either way it has to be cleaned

#### General steps

Remove duplicates

Handle missing data

Process strings

**Process Categorical data** 

**Process Numerical data** 

Normalize Data (essential if data will be used by a

machine learning algorithm)

Process dates (maybe later)

#### General steps

Remove duplicates

Handle missing data

Process strings

Did much of this when introducing project 1

**Process Categorical data** 

**Process Numerical data** 

Normalize Data (essential if data will be used by a

machine learning algorithm)

Process dates (maybe later)

#### General steps

Remove duplicates
Handle missing data

Todays topics

Process strings

**Process Categorical data** 

**Process Numerical data** 

Normalize Data (essential if data will be used by a

machine learning algorithm)

Process dates (maybe later)

First see if there are any

1 df.duplicated().sum()

#### First see if there are any

```
1 df.duplicated().sum()
```

#### If so then verify them visually

```
1 df[df.duplicated()].sort_values(by='name')
```

First see if there are any

```
1 df.duplicated().sum()
```

If so then verify them visually

```
1 df[df.duplicated()].sort_values(by='name')
```

If everything looks fine, get rid of them

```
1 df.drop_duplicates(inplace=True)
```

First see if there are any

```
1 df.duplicated().sum()
```

If so then verify them visually

```
1 df[df.duplicated()].sort_values(by='name')
```

If everything looks fine, get rid of them

```
1 df.drop_duplicates(inplace=True)
```

But there could be extenuating circumstances; What if duplicate is missing data?

First see if there are any

```
1 df.duplicated().sum()
```

If so then verify them visually

```
1 df[df.duplicated()].sort_values(by='name')
```

If everything looks fine, get rid of them

```
1 df.drop_duplicates(inplace=True)
```

But there could be extenuating circumstances; What if duplicate is missing data?

Go to 31\_cleaning\_missing\_and\_duplicate\_data.ipynb

First see if there are any

1 df.duplicated().sum()

t_shirt_size_orig	name	t_shirt_size	weight	
large	Shemeka Tweed	large	138.423257	199
large	Curtis Perry	large	179.943743	201
large	Jean Vanblarcom	large	192.245354	202
med	Marion Murphy	med	110.433988	99
med	Ronald Edwards	med	172.863897	100
med	Kathleen Ringrose	med	143.853752	103
small	Deborah Bradshaw	small	104.820189	0
small	Betty Shannon	small	78.662745	1
small	Mai Audet	small	76.240932	2
small	Pearl Miller	NaN	112.973731	5
small	Yvonne Arroyo	NaN	92.639737	19
small	James Dana	NaN	98.201594	25

	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small	Deborah Bradshaw	small
1	78.662745	small	Betty Shannon	small
2	76.240932	small	Mai Audet	small
5	112.973731	NaN	Pearl Miller	small
19	92.639737	NaN	Yvonne Arroyo	small
25	98.201594	NaN	James Dana	small
		1	).	

Missing values here

First the easy solution; Use sklearns SimpleImputer

weight		t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small	Deborah Bradshaw	small
1	78.662745	small	Betty Shannon	small
2	76.240932	small	Mai Audet	small
5	112.973731	NaN	Pearl Miller	small
19	92.639737	NaN	Yvonne Arroyo	small
25	98.201594	NaN	James Dana	small
		•	).	
		T		

#### First the easy solution; Use sklearns SimpleImputer

Installed with Anaconda

from sklearn.impute import SimpleImputer

	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small	Deborah Bradshaw	small
1	78.662745	small	Betty Shannon	small
2	76.240932	small	Mai Audet	small
5	112.973731	NaN	Pearl Miller	small
19	92.639737	NaN	Yvonne Arroyo	small
25	98.201594	NaN	James Dana	small

name t\_shirt\_size\_orig

### Handle missing data (np.Nan)

First the easy solution; Use sklearns SimpleImputer



	-	_	_		_	-	_ ,
199	138.423257		large	Shemeka Tweed			large
201	179.943743		large	Curtis Perry			large
202	192.245354		large	Jean Vanblarcom			large
99	110.433988		med	Marion Murphy			med
100	172.863897		med	Ronald Edwards			med
103	143.853752		med	Kathleen Ringrose			med
0	104.820189		small	Deborah Bradshaw			small
1	78.662745		small	Betty Shannon			small
2	76.240932		small	Mai Audet			small
5	112.973731		NaN	Pearl Miller			small
19	92.639737		NaN	Yvonne Arroyo			small
25	98.201594		NaN	James Dana			small
				<i>)</i> .			

Imputation strategy, can be mean, median (numeric only), most frequent or constant (numeric and strings)

weight t\_shirt\_size

Shemeka Tweed

Jean Vanhlarcom

Deborah Bradshaw

Betty Shannon

Mai Audet

Pearl Miller

Yvonne Arrovo

Marion Murphy Ronald Edwards

Curtis Perry

name t\_shirt\_size\_orig

large

large

large

small

small

small

small

### Handle missing data (np.Nan)

#### First the easy solution; Use sklearns SimpleImputer

imp = imp.fit(df med[['t shirt size']])



y='most\_frequent')

Imputation strategy, can be mean, median (numeric only), most frequent or constant (numeric and strings)

weight t\_shirt\_size

large

large

large

small

small

NaN

NaN

138.423257

179.943743

192 245354

110.433988

172.863897 143.853752

78.662745

76.240932

92.639737

5 112.973731

Fit the imputer to the data, in this case calculate the most Frequent value seen

Shemeka Tweed

Jean Vanhlarcom

Marion Murphy

Ronald Edwards

Deborah Bradshaw

Betty Shannon

Mai Audet

Pearl Miller

Yvonne Arroyo

James Dana

Curtis Perry

name t\_shirt\_size\_orig

large

large

large

small

small

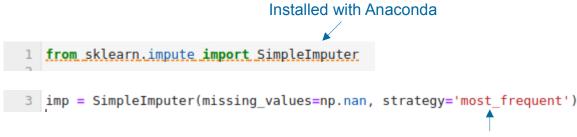
small

small

#### Handle missing data (np.Nan)

#### First the easy solution; Use sklearns SimpleImputer

imp = imp.fit(df med[['t shirt size']])



Imputation strategy, can be mean, median (numeric only), most\_frequent or constant (numeric and strings)

weight t\_shirt\_size

large

large

large

small

small

NaN

NaN

138.423257

179.943743

192 245354

110.433988

172.863897

143.853752

78.662745

76.240932

92.639737

98.201594

5 112.973731

Fit the imputer to the data, in this case calculate the most Frequent value seen

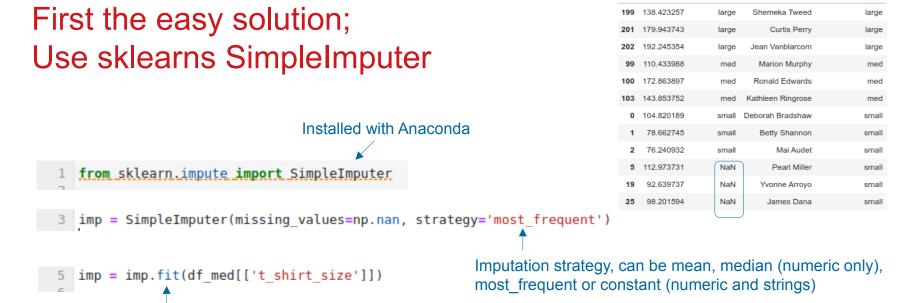
```
7 df_med['impute_t_shirt_size']=imp.transform(df_med[['t_shirt_size']])
```

Transform the data using the imputer, in this case calculate the most Frequent value seen and place df\_med['it in impute\_t\_shirt\_size']

name t\_shirt\_size\_orig

weight t\_shirt\_size

### Handle missing data (np.Nan)



Fit the imputer to the data, in this case calculate the most Frequent value seen

```
7 df_med['impute_t_shirt_size']=imp.transform(df_med[['t_shirt_size']])
```

Transform the data using the imputer, in this case calculate the most Frequent value seen and place df\_med['it in impute\_t\_shirt\_size']

But you can usually do better than this ...

What if you calculate missing values Based on weight.

	weight	L_SIIIIL_SIZE	name	t_siiit_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small	Deborah Bradshaw	small
1	78.662745	small	Betty Shannon	small
2	76.240932	small	Mai Audet	small
5	112.973731	NaN	Pearl Miller	small
19	92.639737	NaN	Yvonne Arroyo	small
25	98.201594	NaN	James Dana	small
		(	)	

weight t shirt size

name t shirt size orig

### Handle missing data (np.Nan)

What if you calculate missing values Based on weight.

#### Calculate average weight for each t-shirt size 1 avgs = df\_better.groupby('t\_shirt\_size').mean()

```
2 avgs.weight
t shirt size
```

large 177.410759 med 138.508626 small 101.173410 Name: weight, dtype: float64

t_silit_size_orig	name	1_311111_3126	weight	
large	Shemeka Tweed	large	138.423257	199
large	Curtis Perry	large	179.943743	201
large	Jean Vanblarcom	large	192.245354	202
med	Marion Murphy	med	110.433988	99
med	Ronald Edwards	med	172.863897	100
med	Kathleen Ringrose	med	143.853752	103
small	Deborah Bradshaw	small	104.820189	0
small	Betty Shannon	small	78.662745	1
small	Mai Audet	small	76.240932	2
small	Pearl Miller	NaN	112.973731	5
small	Yvonne Arroyo	NaN	92.639737	19
small	James Dana	NaN	98.201594	25

weight t shirt size

# What if you calculate missing values Based on weight.

#### Calculate average weight for each t-shirt size

```
2 avgs.weight

t_shirt_size
large 177.410759
med 138.508626
small 101.173410
Name: weight, dtype: float64
```

	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small	Deborah Bradshaw	small
1	78.662745	small	Betty Shannon	small
2	76.240932	small	Mai Audet	small
5	112.973731	NaN	Pearl Miller	small
19	92.639737	NaN	Yvonne Arroyo	small
25	98.201594	NaN	James Dana	small
			J.	

#### Use that info to impute missing values based on user weight

```
#map works on a column apply works on a row, which means we have access to the entire row
   def func(row):
       if row.t_shirt_size is np.NaN:
           #get a list of differences between this weight and average weights
           lst_vals = [abs(row.weight-val) for val in avgs.weight]
           #get the index of the minimum value
9
           min val = min(lst vals)
           min_index=lst_vals.index(min_val)
10
11
12
           #return t shirt size corresponding to this index
13
           return avgs.index[min index]
14
       #its not missing, return what's there
       return row.t shirt size
16 df better['impute t shirt size'] = df.apply(func, axis=1)
```

# What if you calculate missing values Based on weight.

#### Calculate average weight for each t-shirt size

```
2 avgs.weight

t_shirt_size
large 177.410759
med 138.508626
small 101.173410
Name: weight, dtype: float64
```

	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small	Deborah Bradshaw	small
1	78.662745	small	Betty Shannon	small
2	76.240932	small	Mai Audet	small
5	112.973731	NaN	Pearl Miller	small
19	92.639737	NaN	Yvonne Arroyo	small
25	98.201594	NaN	James Dana	small
			J.	

#### Use that info to impute missing values based on user weight

```
#map works on a column apply works on a row, which means we have access to the entire row
3 def func(row):
       if row.t shirt size is np.NaN:
           #get a list of differences between this weight and average weights
           lst_vals = [abs(row.weight-val) for val in avgs.weight]
           #get the index of the minimum value
           min val = min(lst vals)
10
           min_index=lst_vals.index(min_val)
11
12
           #return t shirt size corresponding to this index
13
           return avgs.index[min index]
14
       #its not missing, return what's there
       return row.t_shirt_size
16 df better['impute t shirt size'] = df.apply(func, axis=1)
```

#### Go to 31\_cleaning\_missing\_and\_duplicate\_data.ipynb

# Categorical data

Coming shortly