

# 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

Many machine learning algorithms require data to be in numerical format(linear regression, neural networks, clustering), but not all (random forest). If you are using data for input to an algorithm that requires numerical data, then there are some additional steps.

Process Categorical data

Normalize Data

Process dates (if needed)

Reduce dimensionality

# General steps

Remove duplicates

Handle missing data

Process strings

Did much of this when introducing project 1

Many machine learning algorithms require data to be in numerical format (linear regression, neural networks, clustering), but not all (random forest). If you are using data for input to an algorithm that requires numerical data, then there are some additional steps.

Process Categorical data

Normalize Data

Process dates (if needed)

Reduce dimensionality

# General steps

Remove duplicates  
Handle missing data  
Process strings

Today's topics

Many machine learning algorithms require data to be in numerical format (linear regression, neural networks, clustering), but not all (random forest). If you are using data for input to an algorithm that requires numerical data, then there are some additional steps.

Process Categorical data  
Normalize Data  
Process dates (if needed)  
Reduce dimensionality

# General steps

Remove duplicates  
Handle missing data  
Process strings

Today's topics

Many machine learning algorithms require data to be in numerical format (linear regression, neural networks, clustering), but not all (random forest). If you are using data for input to an algorithm that requires numerical data, then there are some additional steps.

Process Categorical data

Process Numerical data

Normalize Data

Process dates (if needed)

Reduce dimensionality

This is not a complete list of steps

# Remove duplicates

First see if there are any

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



# Remove duplicates

First see if there are any

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

If so then verify them visually

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

# Remove duplicates

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)
```

# Remove duplicates

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 a duplicate row is missing some data?

# Remove duplicates

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 some data?

Go to [31\\_cleaning\\_missing\\_and\\_duplicate\\_data.ipynb](#)

# Handle missing data (np.Nan)

	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

# Handle missing data (np.Nan)

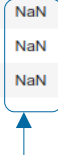
	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small	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

Missing values here

# Handle missing data (np.Nan)

First the easy solution;  
Use sklearn's 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



# Handle missing data (np.Nan)

First the easy solution;  
Use sklearn's SimpleImputer

Installed with Anaconda

```
1 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



# Handle missing data (np.Nan)

First the easy solution;  
Use sklearn's SimpleImputer

Installed with Anaconda

```
1 from sklearn.impute import SimpleImputer
```

```
3 imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

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

	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

# Handle missing data (np.Nan)

First the easy solution;  
Use sklearn's SimpleImputer

Installed with Anaconda

```
1 from sklearn.impute import SimpleImputer
```

```
3 imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

```
5 imp = imp.fit(df_med[['t_shirt_size']])
```

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

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

	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

# Handle missing data (np.Nan)

First the easy solution;  
Use sklearn's SimpleImputer

Installed with Anaconda

```
1 from sklearn.impute import SimpleImputer
```

```
3 imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

```
5 imp = imp.fit(df_med[['t_shirt_size']])
```

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']

	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

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

# Handle missing data (np.Nan)

First the easy solution;  
Use sklearn's SimpleImputer

Installed with Anaconda

```
1 from sklearn.impute import SimpleImputer
```

```
3 imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

```
5 imp = imp.fit(df_med[['t_shirt_size']])
```

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']

	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

But you can usually do better than this ...

# Handle missing data (np.Nan)

What if you calculate missing values  
Based on weight.

	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

# 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
```

	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

# 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
```

	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

Use that info to impute missing values based on user weight

```
1 #map works on a column apply works on a row which means we have access to the entire row
2
3 def func(row):
4     if row.t_shirt_size is np.NaN:
5         #get a list of differences between this weight and average weights
6         lst_vals = [abs(row.weight-val) for val in avgs.weight]
7
8         #get the index of the minimum value
9         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
15    return row.t_shirt_size
16 df_better['impute_t_shirt_size'] = df.apply(func, axis=1)
```

# 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
```

	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

Use that info to impute missing values based on user weight

```
1 #map works on a column apply works on a row which means we have access to the entire row
2
3 def func(row):
4     if row.t_shirt_size is np.NaN:
5         #get a list of differences between this weight and average weights
6         lst_vals = [abs(row.weight-val) for val in avgs.weight]
7
8         #get the index of the minimum value
9         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
15    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](#)



# Cardinality

**Cardinality:** the number of distinct elements in a set. For our purposes the number of unique values in a column

# Categorical data

Categorical data can be subdivided into 2 types

Ordinal data– data that has an order, can be sorted

- ex. t-shirt size (small<medium<large)
- The average of a small and large is medium

Nominal data – data that has no order

- ex. t-shirt color (Red, Blue, Green) one is not greater than another
- The average of Red and Green is not Blue

# Categorical data

Categorical data can be subdivided into 2 types

Ordinal data– data that has an order, can be sorted

- ex. t-shirt size (small<medium<large)
- The average of a small and large is medium

Nominal data – data that has no order

- ex. t-shirt color (Red, Blue, Green) one is not greater than another
- The average of Red and Green is not Blue

Both types need to be encoded numerically in order to be used by many ML models. But their encoding techniques differ depending on the type of model used.

# Ordinal data

Ordinal data— data that has an order, can be sorted

- ex. t-shirt size (small<medium<large)

Since it has an order, just convert it to a number

```
size_mapping = {'small':1, 'medium':2, 'large':3}  
df.t_shirt_size = df.t_shirt_size.map(size_mapping)
```

	weight	t_shirt_size	t_shirt_color	name
0	87.478379	small	black	Timothy Bunch
1	101.982078	small	black	Miguel Williams
2	114.504086	small	orange	Tommy Jennings
3	95.567857	small	red	Willie Ledet
4	109.106926	small	orange	David Smith
...	...	...	...	...
295	149.039786	large	green	Irene Glover
296	189.241702	large	orange	Theresa Tomlin
297	173.061783	large	red	Rebekah Millar
298	178.617007	large	red	Melinda Bonner
299	193.698527	large	blue	Frank Gonzalez

300 rows × 4 columns

Transform



	weight	t_shirt_size	t_shirt_color	name
0	87.478379	1	black	Timothy Bunch
1	101.982078	1	black	Miguel Williams
2	114.504086	1	orange	Tommy Jennings
3	95.567857	1	red	Willie Ledet
4	109.106926	1	orange	David Smith
...	...	...	...	...
295	149.039786	3	green	Irene Glover
296	189.241702	3	orange	Theresa Tomlin
297	173.061783	3	red	Rebekah Millar
298	178.617007	3	red	Melinda Bonner
299	193.698527	3	blue	Frank Gonzalez

300 rows × 4 columns

# Ordinal data

## Advantages

- Establishes a numerical order
- Does not add new columns to DataFrame
- Works with tree based models (Random Forest, Boosted Trees).

## Disadvantages

- You usually have to hand code the numbering to ensure the ordering is correct (so you do not get small=3, large=2, medium=1)

# Nominal data

Does not have an order so cannot convert a nominal categorical variable to a number in the same way that you do a Ordinal one because this implies an order.

T-shirt color is nominal `ts_colors = ['green', 'blue', 'orange', 'red', 'black']`

How to convert t-shirt color to a number without implying an order?

# Nominal data

Does not have an order so cannot convert a nominal categorical variable to a number in the same way that you do a Ordinal one because you would then imply an order.

T-shirt color is nominal `ts_colors = ['green', 'blue', 'orange', 'red', 'black']`

How to convert t-shirt color to a number without implying an order?

Use something called One Hot Encoding. You create 1 column for each unique nominal value.

# Nominal data – One Hot Encode t\_shirt\_color

	weight	t_shirt_size	t_shirt_color	name
0	87.478379	1	black	Timothy Bunch
1	101.982078	1	black	Miguel Williams
2	114.504086	1	orange	Tommy Jennings
3	95.567857	1	red	Willie Ledet
4	109.106926	1	orange	David Smith
...	...	...	...	...
295	149.039786	3	green	Irene Glover
296	189.241702	3	orange	Theresa Tomlin
297	173.061783	3	red	Rebekah Millar
298	178.617007	3	red	Melinda Bonner
299	193.698527	3	blue	Frank Gonzalez

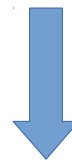
300 rows × 4 columns

call



```
df=pd.get_dummies(df,columns=['t_shirt_color'])
```

transform



Notice that there is now 1 column per color. Only 1 of those columns will ever be 1 at a time, the rest will be 0's

	weight	t_shirt_size	name	t_shirt_color_black	t_shirt_color_blue	t_shirt_color_green	t_shirt_color_orange	t_shirt_color_red
0	87.478379	1	Timothy Bunch	1	0	0	0	0
1	101.982078	1	Miguel Williams	1	0	0	0	0
2	114.504086	1	Tommy Jennings	0	0	0	1	0
3	95.567857	1	Willie Ledet	0	0	0	0	1
4	109.106926	1	David Smith	0	0	0	1	0
...	...	...	...	...	...	...	...	...
295	149.039786	3	Irene Glover	0	0	1	0	0
296	189.241702	3	Theresa Tomlin	0	0	0	1	0
297	173.061783	3	Rebekah Millar	0	0	0	0	1
298	178.617007	3	Melinda Bonner	0	0	0	0	1
299	193.698527	3	Frank Gonzalez	0	1	0	0	0

300 rows × 8 columns



# Nominal data

## Advantages

- One Hot Encoding ensures that a machine learning algorithm will not deduce an order to column members.

## Disadvantages

- Expands the feature space (adds  $n-1$  columns if the nominal variable has  $n$  unique values). So high cardinality columns can dramatically expand feature space.
- Does not work as well with tree based models (Random Forest, Boosted Trees)

# Normalize Data

Coming soon