DATA 301: 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
Process Categorical data
Process Numerical data
Normalize Data (essential if data will be used by a machine learning algorithm)
Process dates (maybe later)

Did much of this when introducing project 1

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)

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 duplicate is missing 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 data?

Go to 31_cleaning_missing_and_duplicate_data.ipynb

Remove duplicates

First see if there are any

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

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 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

Handle missing data (np.Nan)

First the easy solution;
Use sklearns 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 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

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)

Handle missing data (np.Nan)

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

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']])
```

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

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

Handle missing data (np.Nan)

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

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']])

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

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

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

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’]

Handle missing data (np.Nan)

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

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']])

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

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

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

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 ...

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.

	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

Calculate average weight for each t-shirt size

```
avgs = df_better.groupby('t_shirt_size').mean()
avgs.weight
```

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

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 tothe 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
        min_val = min(lst_vals)
        min_index=lst_vals.index(min_val)

        #return t_shirt_size corresponding to this index
        return avgs.index[min_index]
    #its not missing, return what's there
    return row.t_shirt_size
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.

	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

Calculate average weight for each t-shirt size

```
avgs = df_better.groupby('t_shirt_size').mean()
avgs.weight
```

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

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 tothe 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
        min_val = min(lst_vals)
        min_index=lst_vals.index(min_val)

        #return t_shirt_size corresponding to this index
        return avgs.index[min_index]
    #its not missing, return what's there
    return row.t_shirt_size
df_better['impute_t_shirt_size'] = df.apply(func, axis=1)
```

Go to 31_cleaning_missing_and_duplicate_data.ipynb

Categorical data

Coming shortly