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
Scale Numerical Data
Process dates (if needed)
Reduce dimensionality

General steps

Remove duplicates
Handle missing data
Process strings

Much of this for project 1

Process Categorical data
Scale Numerical Data
Process dates (if needed)
Reduce dimensionality

General steps

Remove duplicates
Handle missing data

Todays topics

Process strings
Process Categorical data
Scale Numerical Data
Process dates (if needed)
Reduce dimensionality

General steps

Remove duplicates
Handle missing data

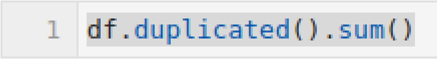
Todays topics

Process strings
Process Categorical data
Scale Numerical Data
Process dates (if needed)
Reduce dimensionality

This is not a complete list of steps

Remove duplicates

First see if there are any

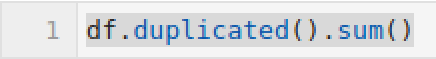


```
df.duplicated().sum()
```

from 31_cleaning_missing_and_duplicate_data.ipynb

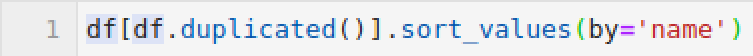
Remove duplicates

First see if there are any



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

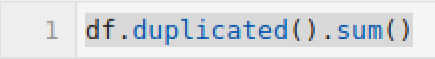
If so then verify them visually



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

Remove duplicates

First see if there are any



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

If so then verify them visually

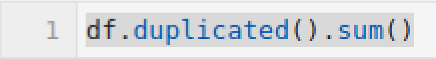
```
df[df.duplicated()].sort_values(by='name')
```

If everything looks fine, get rid of them

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

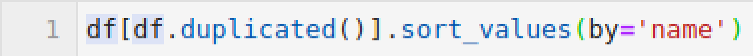
Remove duplicates

First see if there are any



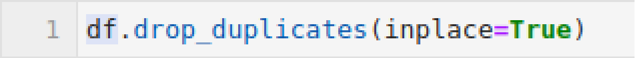
```
df.duplicated().sum()
```

If so then verify them visually



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

If everything looks fine, get rid of them

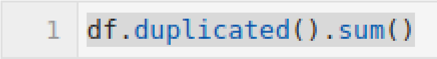


```
df.drop_duplicates(inplace=True)
```

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

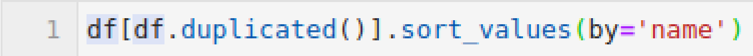
Remove duplicates

First see if there are any



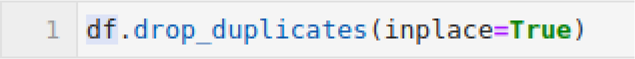
```
df.duplicated().sum()
```

If so then verify them visually



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

If everything looks fine, get rid of them

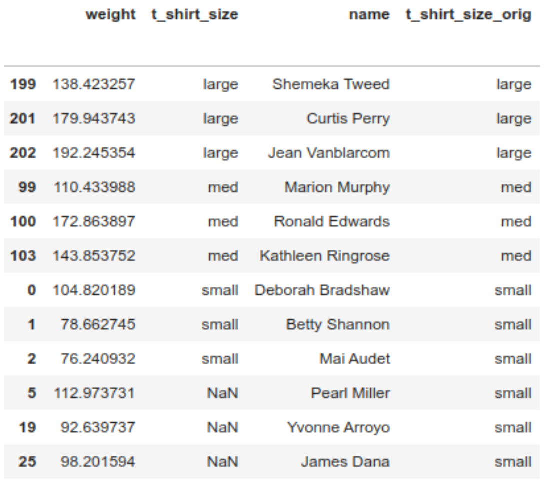


```
df.drop_duplicates(inplace=True)
```

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

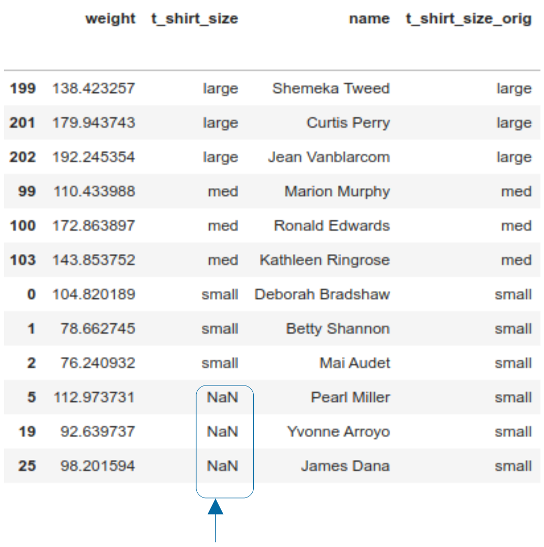
Go to 31_cleaning_missing_and_duplicate_data.ipynb

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)

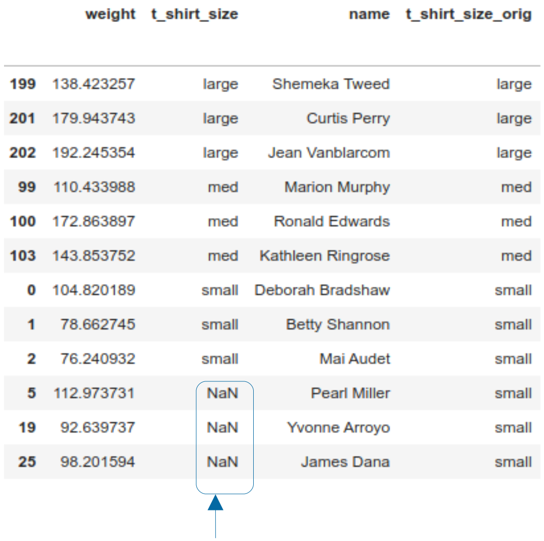


	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small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

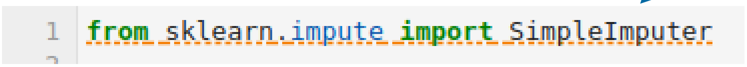


	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small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

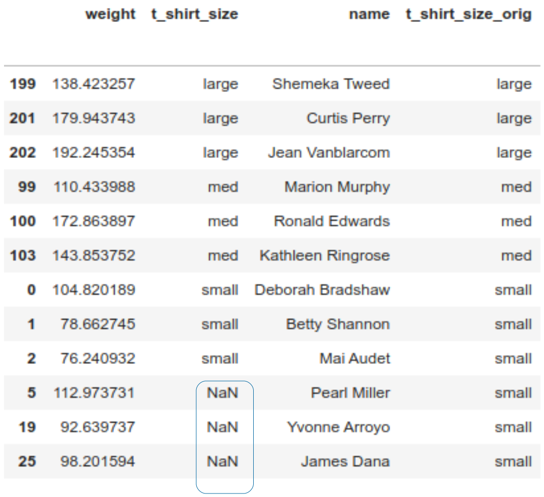
Handle missing data (np.Nan)

First the easy solution;
Use sklearns SimpleImputer

Installed with Anaconda

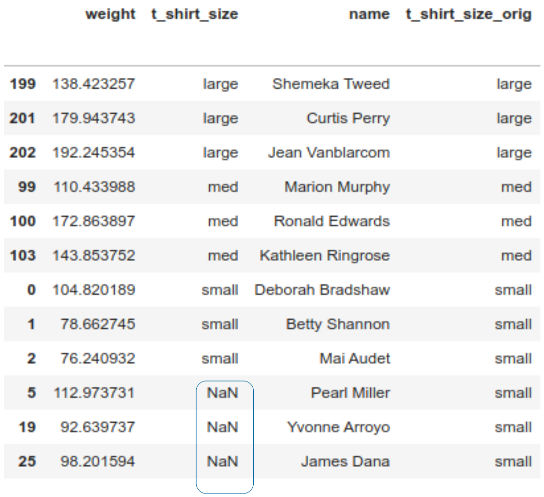


```python
from sklearn.impute import SimpleImputer
```



	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

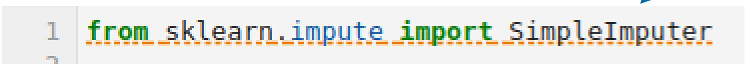
Handle missing data (np.Nan)



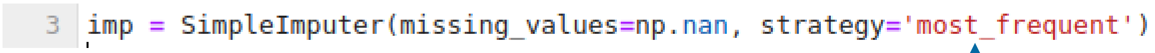
	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small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

First the easy solution;
Use sklearns SimpleImputer

Installed with Anaconda



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

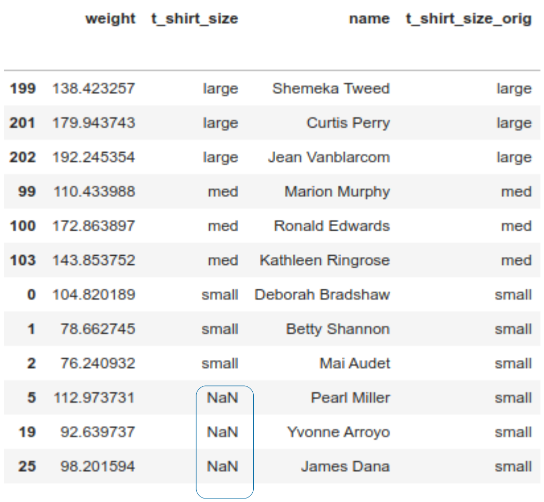


```
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

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

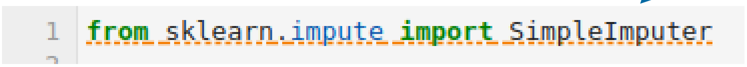
Handle missing data (np.Nan)

First the easy solution;
Use sklearns SimpleImputer

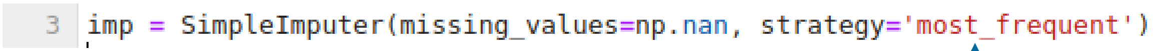


	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small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

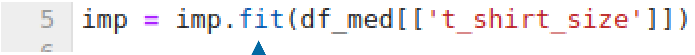


```
from sklearn.impute import SimpleImputer
```



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

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

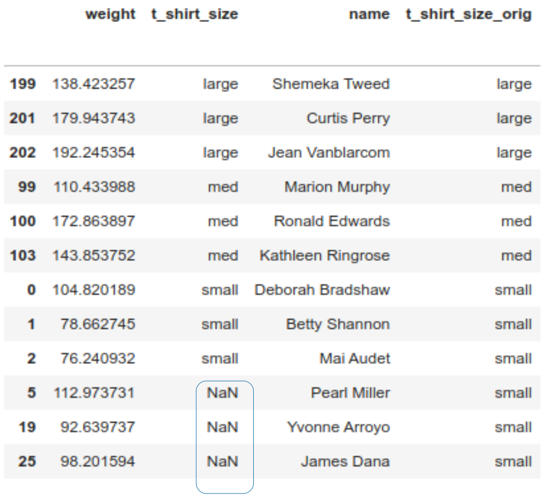


```
imp = imp.fit(df_med[['t_shirt_size']])
```

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

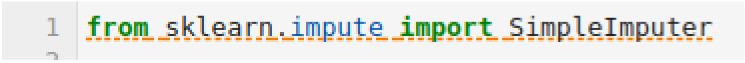
Handle missing data (np.Nan)

First the easy solution;
Use sklearns SimpleImputer

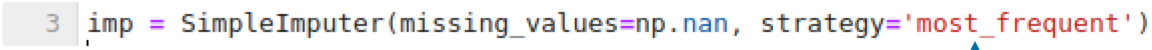


	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small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

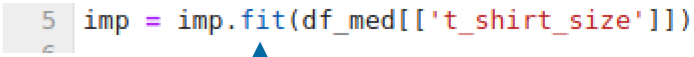


```
from sklearn.impute import SimpleImputer
```



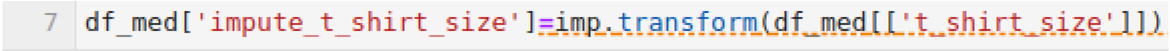
```
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

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



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

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

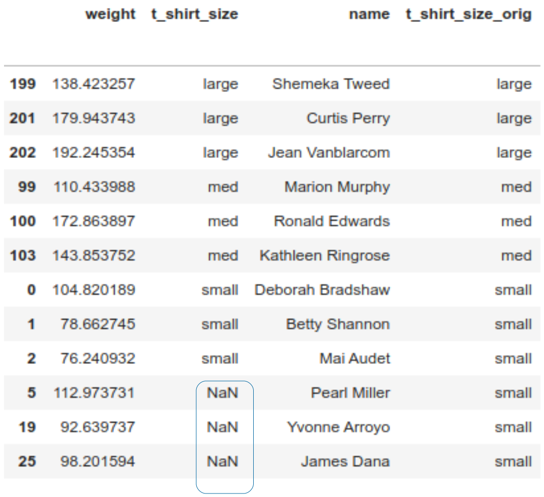


```
df_med['impute_t_shirt_size']=imp.transform(df_med[['t_shirt_size']])
```

Transform the data using the imputer, in this case calculate the most Frequent value seen and replace missing values in df_med[‘impute_t_shirt_size’] with it.

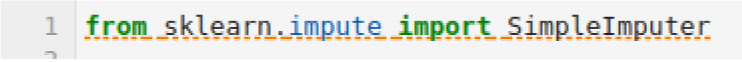
Handle missing data (np.Nan)

First the easy solution;
Use sklearns SimpleImputer

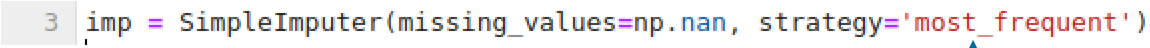


	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small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

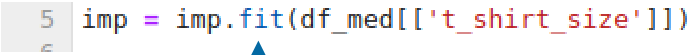


```python
from sklearn.impute import SimpleImputer
```



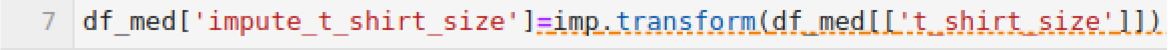
```python
imp = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
```

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



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

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



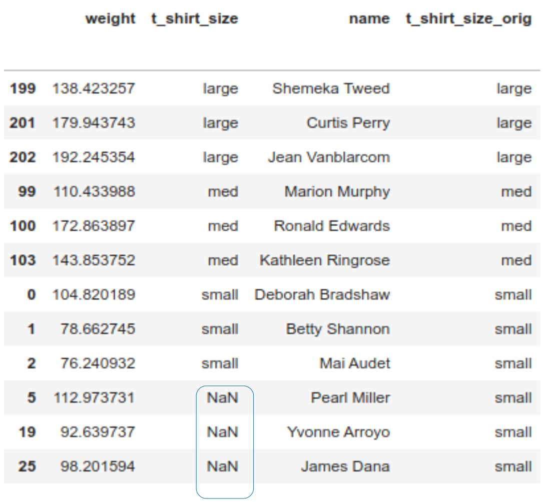
```python
df_med['impute_t_shirt_size']=imp.transform(df_med[['t_shirt_size']])
```

Transform the data using the imputer, in this case calculate the most Frequent value seen and replace missing values in df_med['impute_t_shirt_size'] with it.

But you can usually do better than this ...

Handle missing data (np.Nan)

What if you calculate missing values
Based on weight.

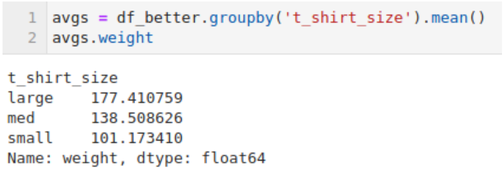


	weight	t_shirt_size	name	t_shirt_size_orig
199	138.423257	large	Shemeka Tweed	large
201	179.943743	large	Curtis Perry	large
202	192.245354	large	Jean Vanblarcom	large
99	110.433988	med	Marion Murphy	med
100	172.863897	med	Ronald Edwards	med
103	143.853752	med	Kathleen Ringrose	med
0	104.820189	small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

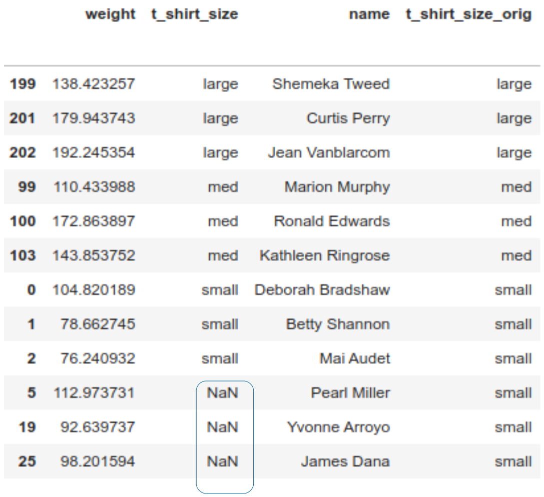
What if you calculate missing values Based on weight.

Calculate average weight for each t-shirt size



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

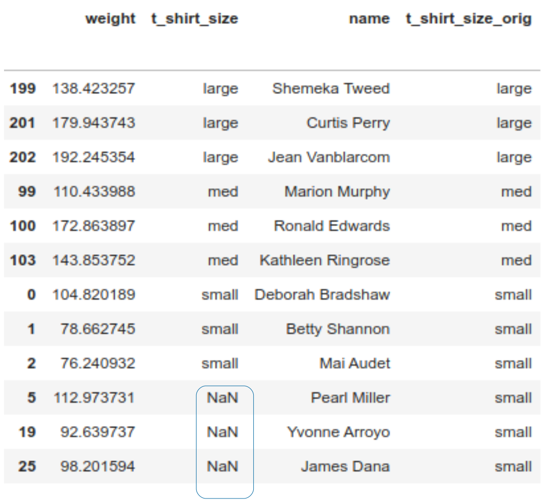
```
t_shirt_size
large    177.410759
med      138.508626
small    101.173410
Name: weight, dtype: float64
```



	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

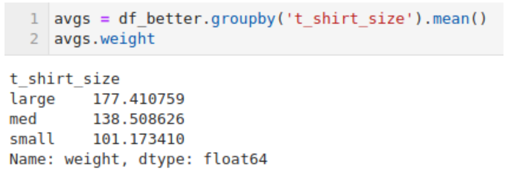
Handle missing data (np.Nan)

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

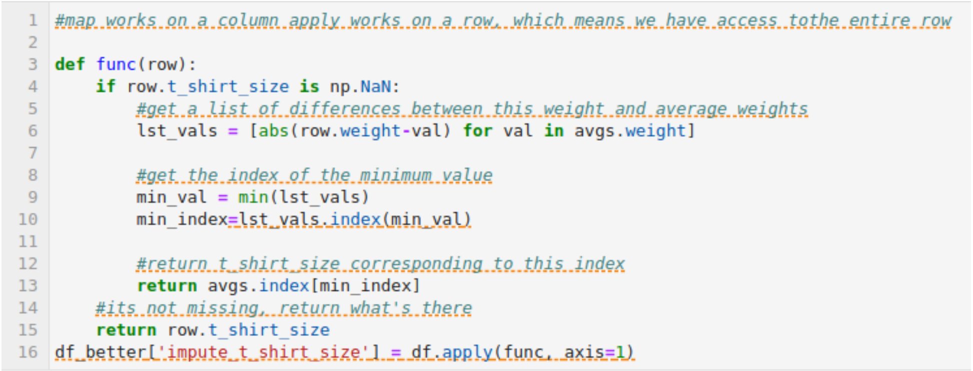
Calculate average weight for each t-shirt size



```
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
```

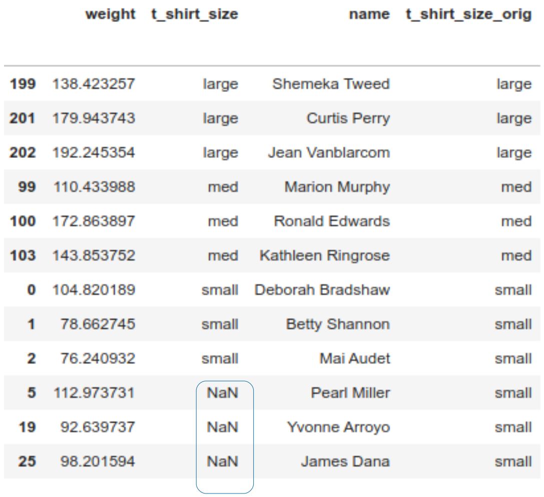
Use that info to impute missing values based on user weight



```
#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)
```

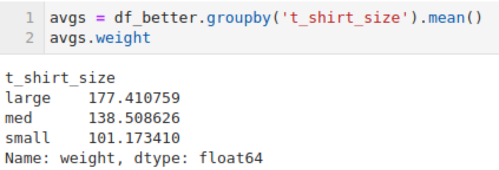
Handle missing data (np.Nan)

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

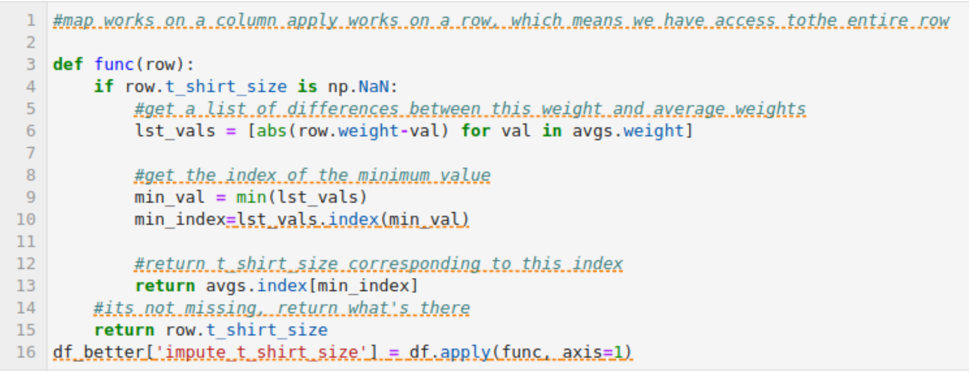
Calculate average weight for each t-shirt size



```
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

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

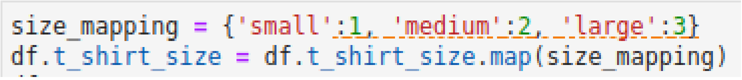
Both types need to be encoded numerically in order to be used by many ML models. But their encoding techniques differ depending on the type of model used.

Ordinal data

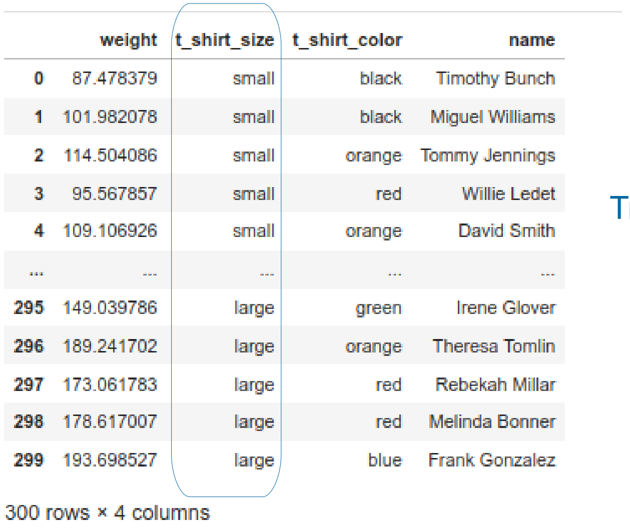
Ordinal data– data that has an order, can be sorted

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

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



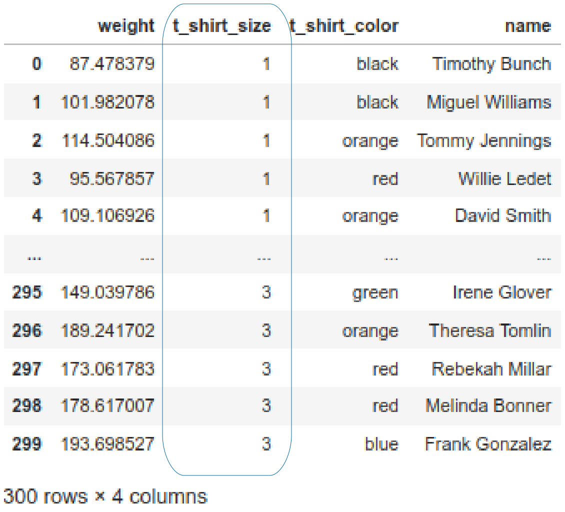
```
size_mapping = {'small':1, 'medium':2, 'large':3}
df.t_shirt_size = df.t_shirt_size.map(size_mapping)
```



	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

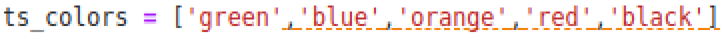
- Establishes a numerical order
- Does not add new columns to DataFrame
- Works with tree based models (Random Forest, Boosted Trees).

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.

T-shirt color is nominal 

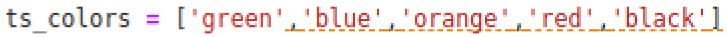
```
ts_colors = ['green','blue','orange','red','black']
```

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.

T-shirt color is nominal

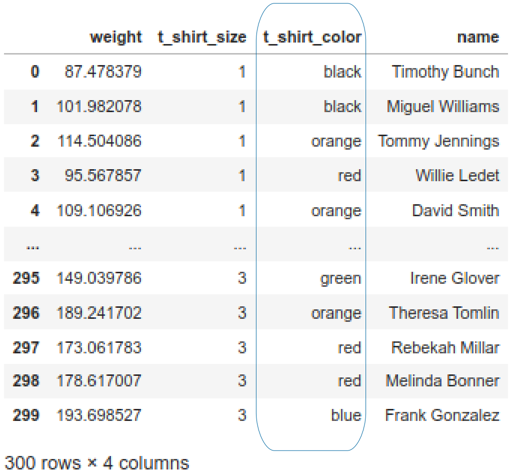


```
ts_colors = ['green','blue','orange','red','black']
```

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.

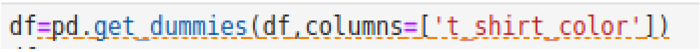
Nominal data – One Hot Encode t_shirt_ color



	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

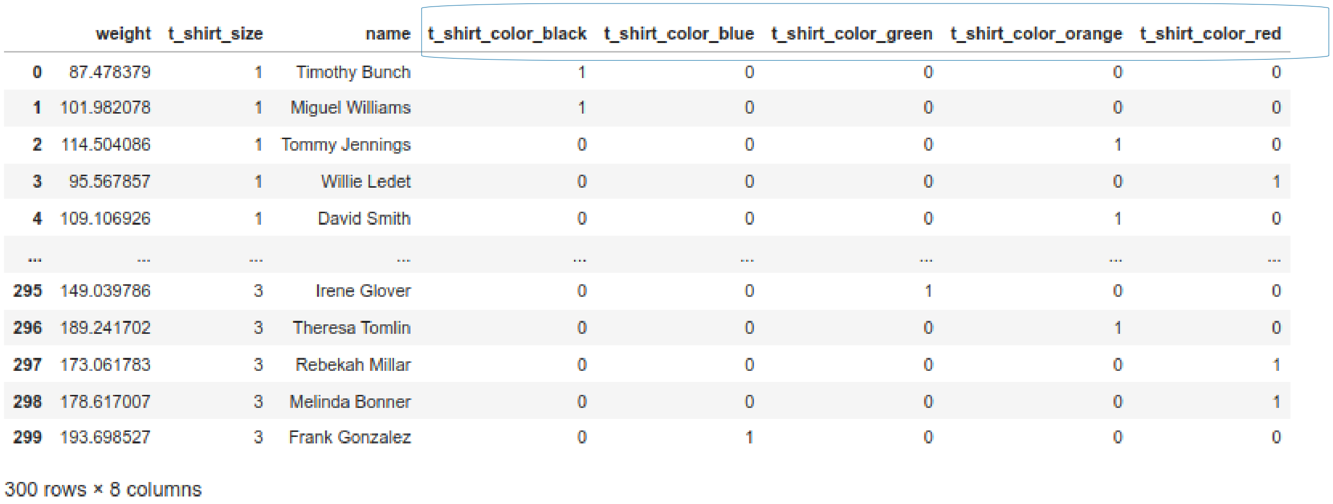
call



```
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 (OHE)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.
- OHE features have perfect multicolinearity (Dummy variable trap, model explainability)

NOTE:

- Don’t use for tree based models (Random Forest, Boosted Trees) treat all columns as ordinal.

Scale Numerical Data

ML algorithms based on Euclidian distance benefit from feature scaling, these include;

- K-means
- K-nearest neighbors
- DBScan (coming soon)
- Principal Component Analysis (PCA)
- Neural Networks

ML algorithms that do not require feature scaling (but it does not hurt);

- Naive Bayes
- Tree Based methods (Random Forest, Boosted Trees)

Scale Numerical Data

Min-Max encoding (normalization) – rescale features to fall between [0,1]

$$x_{\mathrm{norm}} = \frac{x - \min(x)}{\max(x) - \min(x)}$$

But if there are outliers, then they define min() and/or max()

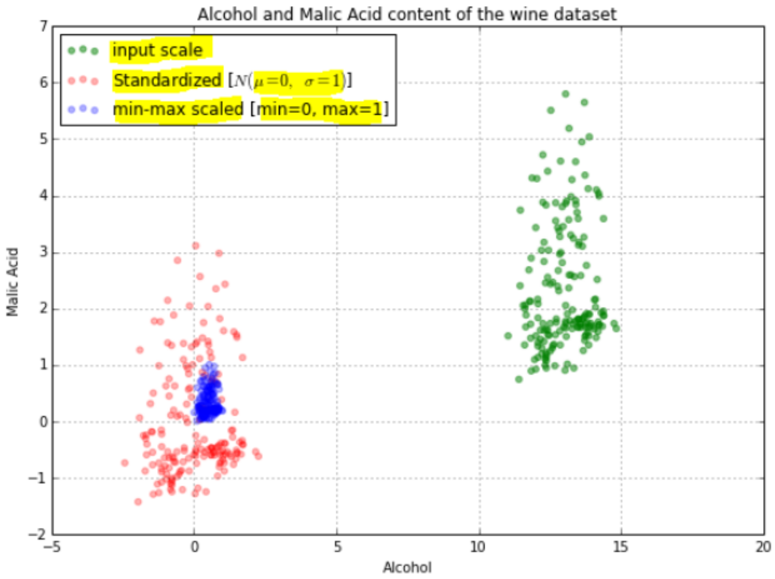
Scale Numerical Data

Standardization – rescale features to a mean of 0 and a standard deviation of 1

$$x_{\mathrm{stand}} = \frac{x - \mathrm{mean}(x)}{\mathrm{standard\ deviation}\ (x)}$$

Note that standardized data is centered at 0, and has both positive and negative values.

Scale Numerical Data

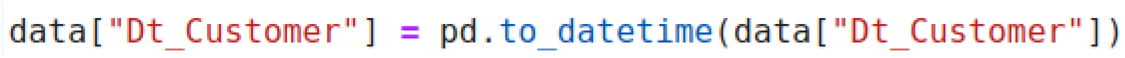


Note that standardized data is spread out more and preserves outlier information

In general prefer Standardization

Process Dates

Date/Times must be converted into a numerical format. The following call will convert many forms of date/time strings into a pandas datetime64 object



```
data["Dt_Customer"] = pd.to_datetime(data["Dt_Customer"])
```

We will use datetime fields a bit more later

Reduce Dimensionality

Columns for a Pandas DataFrame

The more features you have:

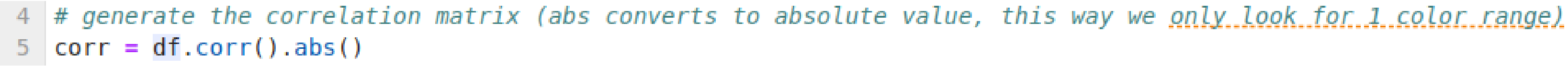
- the more data you need to train a ML model
- the harder it is to run cluster analysis
- the longer it takes for a ML algorithm to converge
- the higher the probability that your model will not generalize to new data
- the harder it is to visualize your data
- the higher the likelyhood that some features are redundant*

So reduce the number of features to only those that you need. Two ways presented here.

- Eliminate Highly Correlated Features
- Principle Components Analysis (PCA)
- We will look at some other ways later

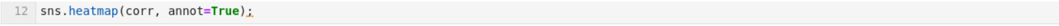
*Redundant features are features that are highly correlated, they also skew analyzing which features are the most important (feature importance - coming soon with Random Forest)

Eliminate Redundant features

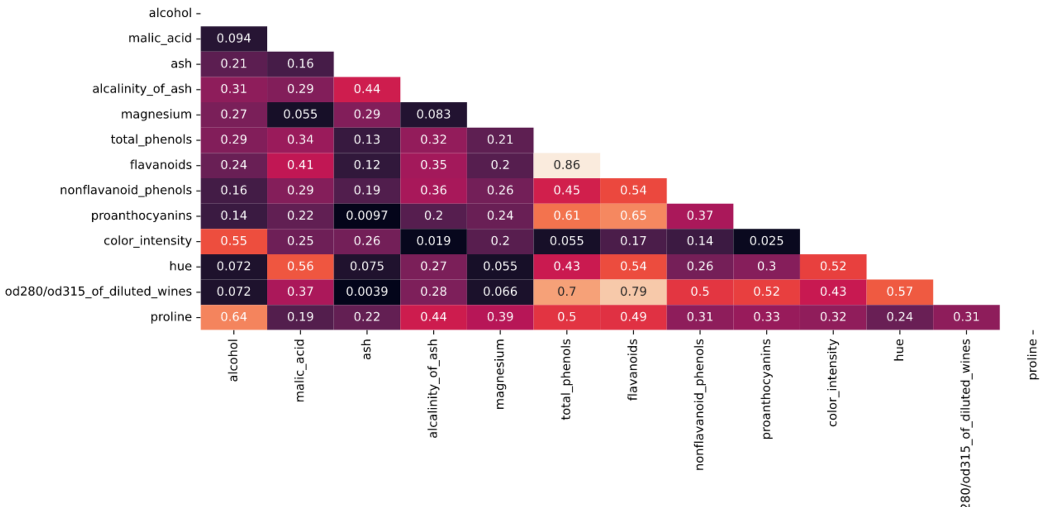
- Redundant features are columns that are highly correlated. They provide little to no additional information.
- Find them by correlation analysis, then drop them.
- Pandas DataFrame has a builtin correlation function that will calculate the correlation between every column
- 

```
# generate the correlation matrix (abs converts to absolute value, this way we only look for 1 color range)
corr = df.corr().abs()
```

- Use seaborn to display this matrix as a heatmap



```
sns.heatmap(corr, annot=True);
```



Eliminate Redundant features

Advantages:

- Faster model training with fewer features
- Your model may generalize better
- Eliminates source of error in Feature Importance analysis (later)

Disadvantages

- Eliminates few columns (what if you have hundreds?)
- You have to manually decide correlation threshold for elimination (typically 95%-99%)

Principle Components Analysis (PCA)

PCA is the process of computing the principal components and using them to perform a change of basis on the data, sometimes using only the first few principal components and ignoring the rest.   The principal components are eigenvectors of the data's covariance matrix.*

Bit of a mental handfull, how about:

*https://en.wikipedia.org/wiki/Principal_component_analysis

Principle Components Analysis (PCA)

PCA takes original features (columns) and recombines them into a list of new features (columns). Each new feature is:

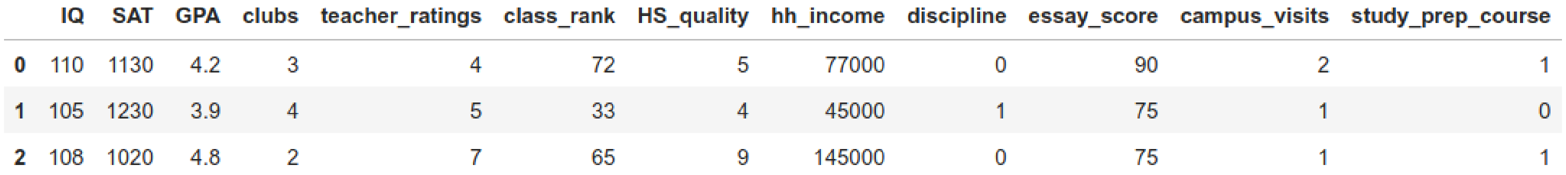
- a combination of the original features
- is not correlated with any other new PCA feature (they are all orthogonal to each other)

These features are sorted by the amount of information they capture (variance explained). The first captures the most, the next captures the second most and so on.

The problem is that these PCA features are hard to interpret.

Principle Components Analysis - ELI5

Suppose you have a list of 1000 students with the following features, and you want to predict which are going to do well in college

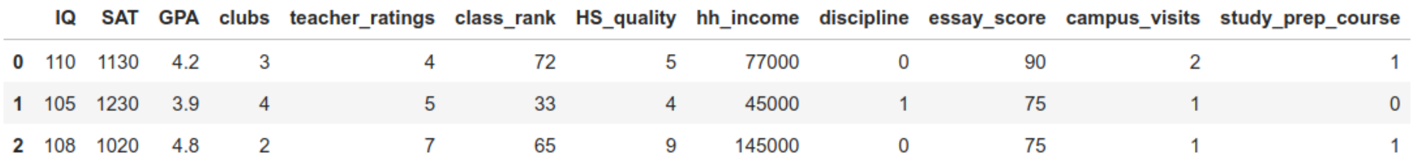


	IQ	SAT	GPA	clubs	teacher_ratings	class_rank	HS_quality	hh_income	discipline	essay_score	campus_visits	study_prep_course
0	110	1130	4.2	3	4	72	5	77000	0	90	2	1
1	105	1230	3.9	4	5	33	4	45000	1	75	1	0
2	108	1020	4.8	2	7	65	9	145000	0	75	1	1

https://www.reddit.com/r/statistics/comments/11d8ri/eli5_principal_component_analysis/

Principle Components Analysis - ELI5

Suppose you have a list of 1000 students with the following features, and you want to predict which are going to do well in college



	IQ	SAT	GPA	clubs	teacher_ratings	class_rank	HS_quality	hh_income	discipline	essay_score	campus_visits	study_prep_course
0	110	1130	4.2	3	4	72	5	77000	0	90	2	1
1	105	1230	3.9	4	5	33	4	45000	1	75	1	0
2	108	1020	4.8	2	7	65	9	145000	0	75	1	1

Standardize data then run PCA. Top 3 new PCA features that capture the most information may be;

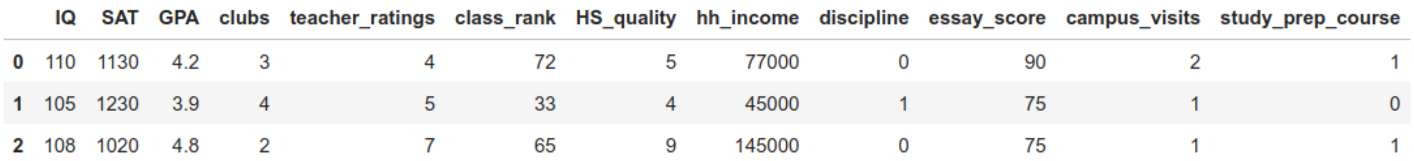
Does not have to be 3, but the more you choose, the better you represent the original dataset

X = B1*IQ + B2*SAT + B3*GPA
Y = B4*clubs + B5*teacher_rating + B6*discipline
Z = B6*income + B6*HS_quality + B7*class_rank

https://www.reddit.com/r/statistics/comments/11d8ri/eli5_principal_component_analysis/

Principle Components Analysis - ELI5

Suppose you have a list of 1000 students with the following features, and you want to predict which are going to do well in college



	IQ	SAT	GPA	clubs	teacher_ratings	class_rank	HS_quality	hh_income	discipline	essay_score	campus_visits	study_prep_course
0	110	1130	4.2	3	4	72	5	77000	0	90	2	1
1	105	1230	3.9	4	5	33	4	45000	1	75	1	0
2	108	1020	4.8	2	7	65	9	145000	0	75	1	1

Standardize data then run PCA. Top 3 new PCA features that capture the most information may be;

X = B1*IQ + B2*SAT + B3*GPA
Y = B4*clubs + B5*teacher_rating + B6*discipline
Z = B6*income + B6*HS_quality + B7*class_rank

The dataset is reduced from (1000,12) to (1000,3)
None of the new features are correlated
Much of the original information is still captured (but not all)
But it is usually difficult to interpret the new PCA features.

https://www.reddit.com/r/statistics/comments/11d8ri/eli5_principal_component_analysis/

Summary

- Handle duplicates
- Impute missing data (or drop it)
- Pre process strings
- Determine if string columns are ordinal or nominal categorical variables
- Transform categorical variables
- Scale data
- Consider dimensionality reduction