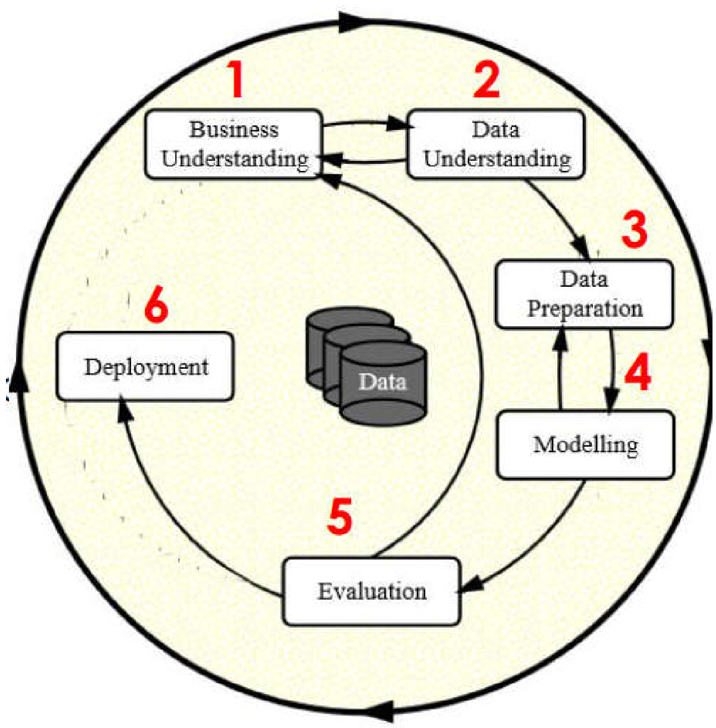
Data Mining Course

Feature Engineering

Prof. Chiraz Ben Abdelkader

October 03-04, 2019

Data Mining Pipeline



Step 4 – Data Preparation

- Data cleaning last week
- Feature engineering today
- Feature selection next week

Feature Engineering

- In machine learning, attributes are also called features.
- Feature engineering is the process of modifying or transforming existing attributes (columns).
- This is done for two main reasons:
  1) Put the data in the form required by the modeling techniques that we intend to use.
  2) To improve the quality of the attributes for building better/more accurate models.
- We will focus mostly on feature engineering for structured data.

What is Feature Engineering?

Feature engineering is about **creating new input features** from your existing ones.

In general, you can think of data cleaning as a process of subtraction and feature engineering as a process of addition.

This is often one of the most valuable tasks a data scientist can do to improve model performance, for 3 big reasons:

1. You can isolate and highlight key information, which helps your algorithms "focus" on what's important.
2. You can bring in your own **domain expertise**.
3. Most importantly, once you understand the "vocabulary" of feature engineering, you can bring in other people's domain expertise!

Getting classy.

Source: https://elitedatascience.com/feature-engineering

Common Feature Transformations

- **Scaling**: map values of a *numeric* attribute into a uniform range, such as [0,1] or [-1,1]
- **Log normalization**: apply log function on a numeric variable
- **Grouping sparse categories**: values of a categorical attribute with low frequency in data
- **One-hot encoding**: convert a *categorical* attribute to *numeric*
- **Date and time attributes**: (*unstructured*)
  - Time --> *hours*, *minutes*, *seconds*, ...
  - Date --> *day*, *month*, *year*

Scaling (or Scale Normalization)

- Also called *data standardization*
- Map values of a numeric attribute into a uniform range so that all attributes have similar range of values, for e.g. [0,1] or [-1,1]
- Scaling is necessary when using a modeling method that is sensitive to scale
  - This means the method gives more weight to larger attribute values in the constructed model.

Scaling (cont.)

- Modeling methods that are sensitive to scale:
  - Any method based on calculating differences between attribute values
  - Examples: linear regression, neural networks, KNN, Kmeans, SVM, ...
- Modeling methods NOT sensitive to scale:
  - Any method based on comparing values
  - Examples: decision trees, random forests, ...

Scaling Methods

- **min-max** normalization

$$X_{norm} = \frac{X - X_{min}}{X_{max} - X_{min}}$$

- **Z-score** normalization
  - Also called *standardization*

$$X_{norm} = \frac{x - \mu}{\sigma}$$

- **unit-norm** normalization

$$\vec{X}_{norm} = \frac{\vec{X}}{||\vec{X}||}$$

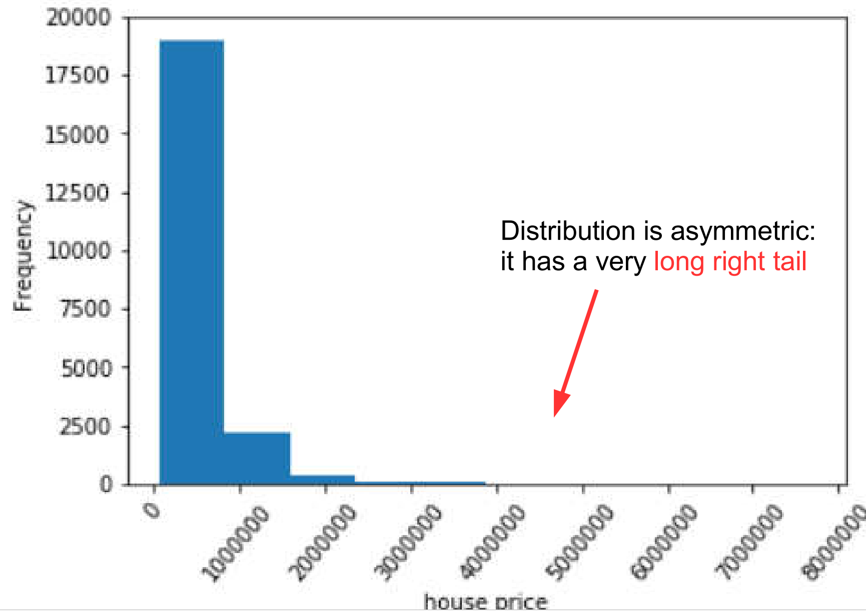
  - $\vec{X}$ is a vector containing all numeric attributes of a particular instance

Log Normalization

- This transformation is typically used when a numeric variable has very skewed distribution (long tail), which is problematic for models
- The distribution of the new variable is more symmetric
- Example: the *price* variable in TP1

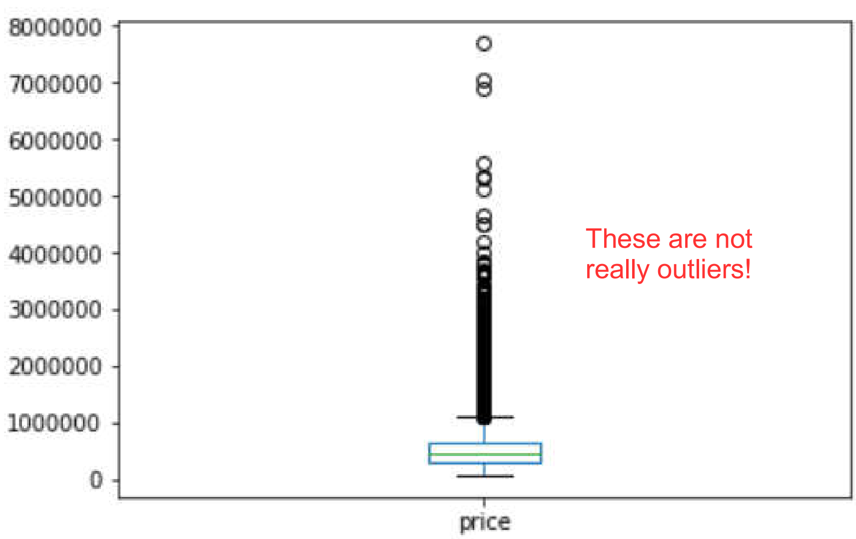
Example

```
In [21]: fig=house_sales_df.price.plot.hist(rot=50)
         fig=plt.xlabel('house price')
```



Example (cont.)

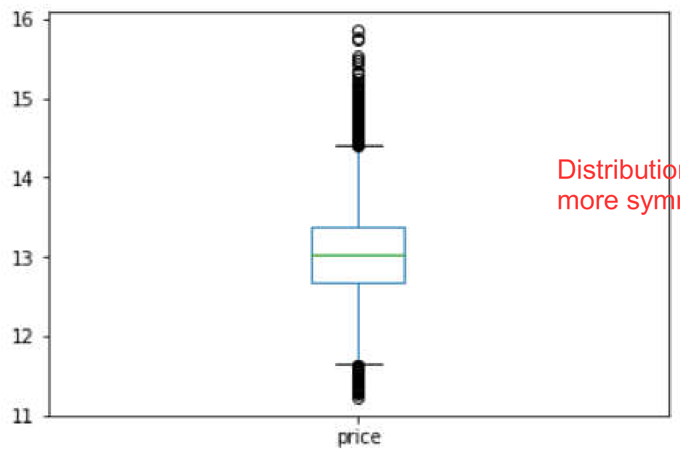
```
In [13]: fig=house_sales_df.price.plot.box()
```



Example (cont.)

```
In [24]:  house_sales_df.price_log = house_sales_df.price.map(np.log)
```

```
In [25]:  fig=house_sales_df.price_log.plot.box()
```



Distribution is now much more symmetric

One-hot Encoding

- Replaces a categorical attribute with K numeric binary attributes, where K = number of categories in the variable.
- The new attributes are called dummy variables
- Necessary for many modeling methods that only work with numeric attributes.
    - Such as linear regression, NNs, SVM, KNN, Kmeans

Example

Categorical variable

fav_color
blue
green
orange
green

3 dummy variables

fav_color_enc
[1, 0, 0]
[0, 1, 0]
[0, 0, 1]
[0, 1, 0]

Example (cont.)

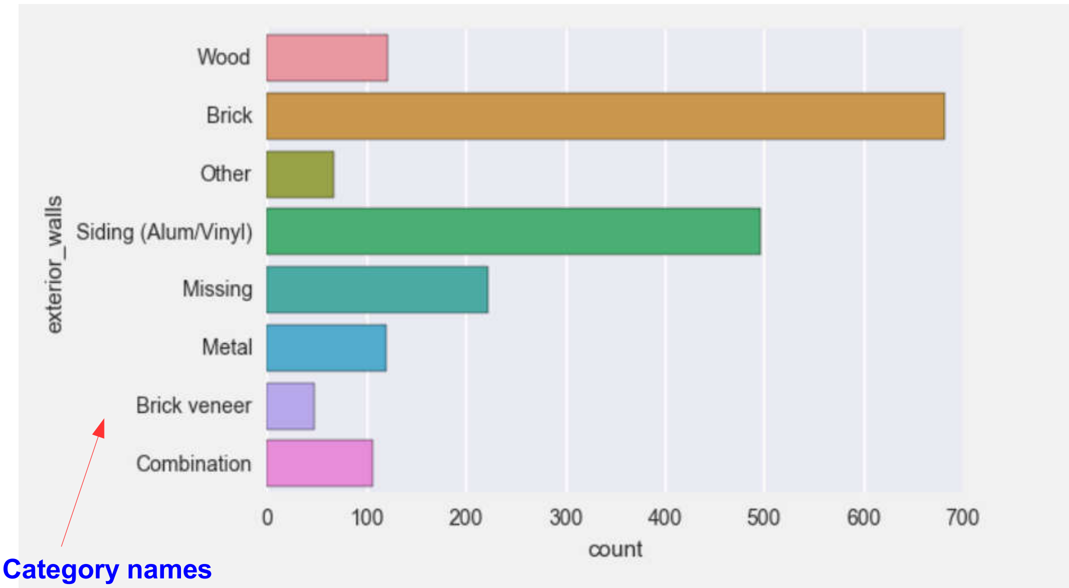
Python Code

```
In [8]: print(users["fav_color"])

0      blue
1     green
2    orange
3     green
Name: fav_color, dtype: object
In [9]: print(pd.get_dummies(users["fav_color"]))

   blue  green  orange
0     1      0       0
1     0      1       0
2     0      0       1
3     0      1       0
```

users is a data frame
fav_color is a categorical column

Example 2

Barplot of a categorical variable called ***exterior_walls***



Example 2 (cont.)

Category name	Dummy variable name	E.g.
Wood	exterior_walls_Wood	= 0
Brick	exterior_walls_Brick	**= 1**
Other	exterior_walls_Other	= 0
Siding (Alum/Vinyl)	exterior_walls_Siding (Alum/Vinyl)	= 0
Missing	exterior_walls_Missing	= 0
Metal	exterior_walls_Metal	= 0
Brick veneer	exterior_walls_Brick veneer	= 0
Combination	exterior_walls_Combination	= 0

Example observation
Where exterior_walls=Brick

Grouping sparse categories

- Combine 2 or more categories into one category

- Two major use cases:
  1) Some categories occur in too few observations (rows)
     - under-represented categories are useless and/or confusing for model construction
     - *Rule of thumb*: at least ~50 observations per category

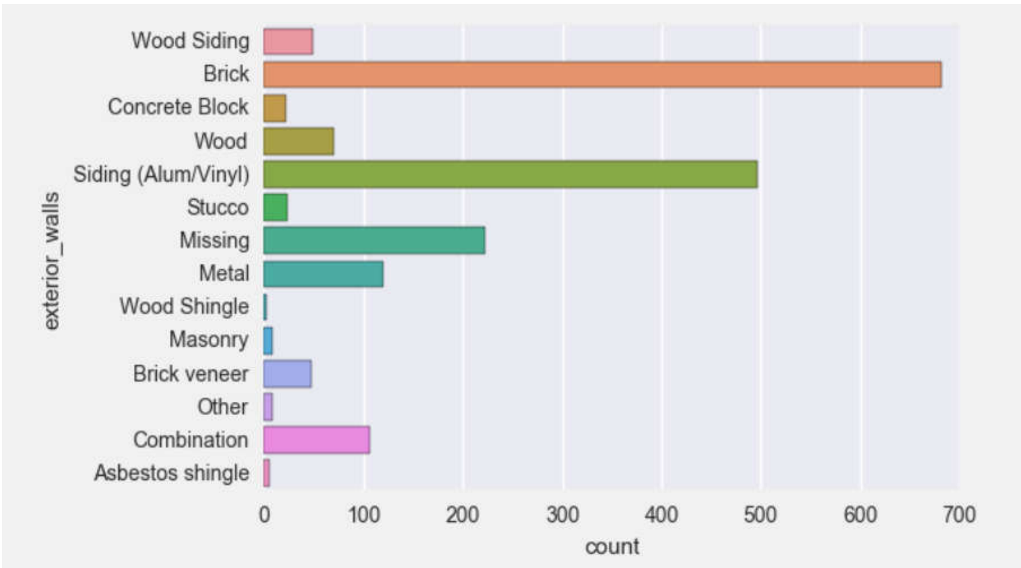
  2) Redundant categories (different category names but represent the same entity in the real-world); actually this can also be considered as data cleaning.

Remark

- When the categorical attribute is ordered, only contiguous categories can be combined.

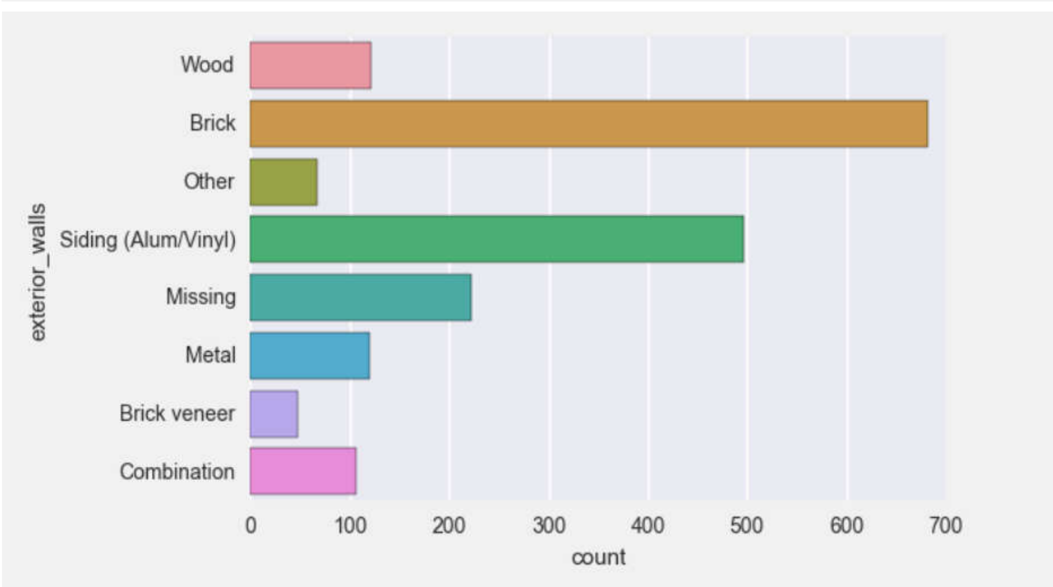
- **Example**
  - Consider an attribute *shirt_size* that has 6 possible values(categories): XS, S, M, L, XL, XXL
  - We can aggregate (combine) XL and XXL into one category
  - But we cannot aggregate XS and XXL

Example



Example (cont.)

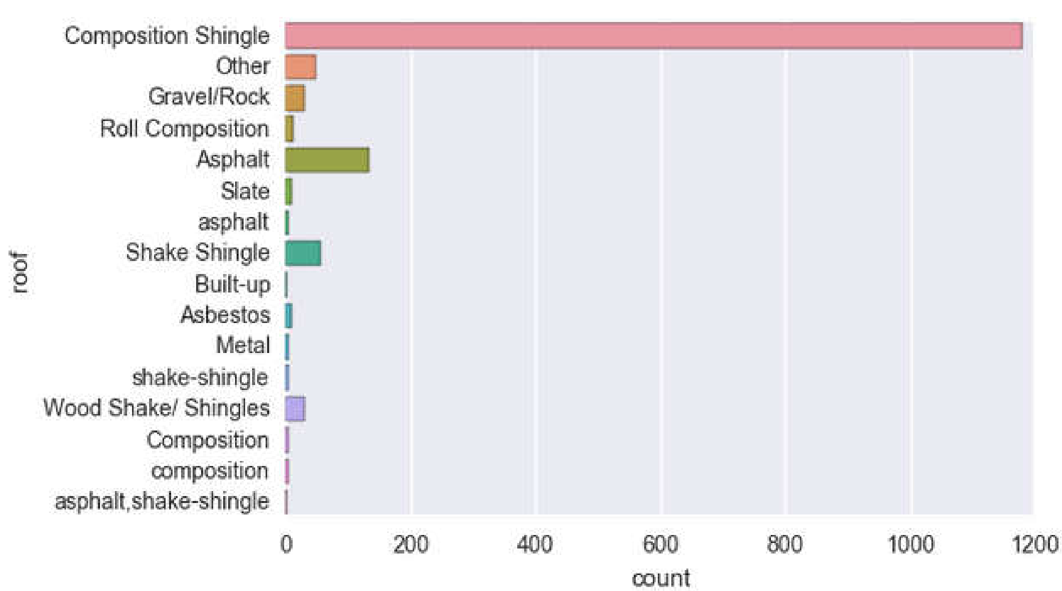
Here's how the class distributions look after combining similar and other classes:



Example (cont.)

- 'Wood Siding', 'Wood Shingle', and 'Wood' were grouped into a single category called 'Wood'
- 'Concrete Block', 'Stucco', 'Masonry', 'Other', and 'Asbestos shingle' were grouped into a new category called 'Other'

Exercise



- Are there categories that occur too few times? **Yes, for example ...**
- Are there redundant categories? **Yes, for example ...**

Summary of Transformations

Transformation name	Type of input & output variables	Why/when should be used?	Python functions and classes
Scale normalization	Numeric to Numeric	All numeric variables must have similar ranges; for scale-sensitive modeling methods.	sklearn.preprocessing.MinMaxScaler sklearn.preprocessing.StandardScaler sklearn.preprocessing.Normalizer
Grouping sparse categories	Categorical to categorical	categories with few observations are useless/noisy for modeling.	Regular Pandas methods
One-hot encoding	Categorical to numeric	You want to later use a model that only accepts numeric features.	<u>For two categories:</u> sklearn.preprocessing.LabelEncoder <u>For multiple categories:</u> **pd.get_dummies**(df.categorical_var)
Log normalization	Numeric to numeric	variable has an asymmetric distribution (long tail)	df['new_var'] = **np.log**(df.old_var) OR df['new_var'] = df.old_var.**map**(**np.log**)
Date/time	unstructured	map unstructured variable to structured variables	Pandas and numpy

Examples with Python Code

Scaling Example

```
from sklearn.preprocessing import StandardScaler

# Create instance of the StandardScaler class
ss = StandardScaler()

# Apply scaler to numeric columns of DataFrame
df_numeric = df[numeric_variables]
df_scaled = ss.fit_transform(df_numeric)
```

MinMaxScaler class is used the same way as ***StandardScaler***

OneHot Encoding Example

```
from sklearn.preprocessing import LabelEncoder

# Create instance of the LabelEncoder class
le = LabelEncoder()

# Apply transformation to a specific categorical column
df['gender_binary'] = le.fit_transform(df.gender)
```

LabelEncoder class is used when categorical variable has 2 values.

OneHot Encoding Example 2

Example with 2 categories

```
In [4]: from sklearn.preprocessing import LabelEncoder

In [5]: le = LabelEncoder()
In [6]: users["sub_enc_le"] = le.fit_transform(users["subscribed"])

In [7]: print(users[["subscribed", "sub_enc_le"]])

  subscribed  sub_enc_le
0          y           1
1          n           0
2          n           0
3          y           1
```

old column   new column

OneHot Encoding Example 3

Example with multiple categories

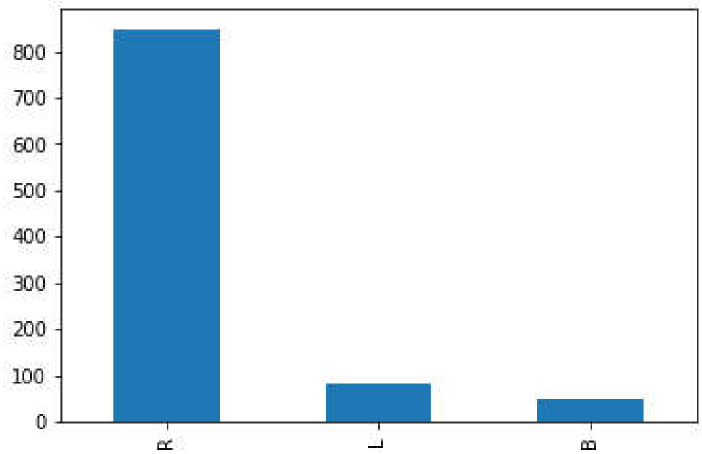
```
In [8]: print(users["fav_color"])

0      blue
1     green
2    orange
3     green
Name: fav_color, dtype: object

In [9]: print(pd.get_dummies(users["fav_color"]))

   blue  green  orange
0     1      0       0
1     0      1       0
2     0      0       1
3     0      1       0
```

users is a data frame
fav_color is a categorical column

Sparse Categories Example

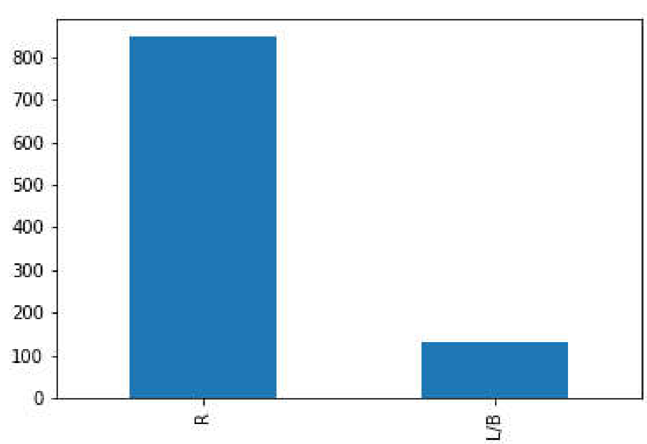
In [27]:
```python
# Bar plot of the categorical variable 'Handed'

school_data_df.Handed.value_counts().plot.bar()
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x208e33a5898>



Sparse Categories Example (cont.)

In [66]:
```python
idx = (school_data_df.Handed == 'L') | (school_data_df.Handed == 'B')

school_data_df.loc[idx,'Handed'] = 'L/B'

fig=school_data_df.Handed.value_counts().plot.bar()
```

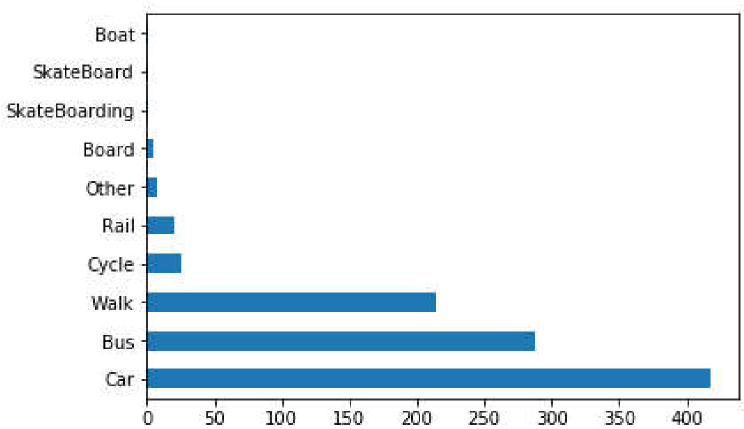


Sparse Categories Example 2

```
In [45]:  1  school_data_df.Travel_to_School.value_counts()

Out[45]: Car              418
         Bus              288
         Walk             214
         Cycle             25
         Rail              21
         Other              7
         Board              4
         SkateBoarding      1
         SkateBoard         1
         Boat               1
         Name: Travel_to_School, dtype: int64
```

Sparse Categories Example 2 (cont)

```
In [49]:  1  fig = school_data_df.Travel_to_School.value_counts().plot.barh()
```

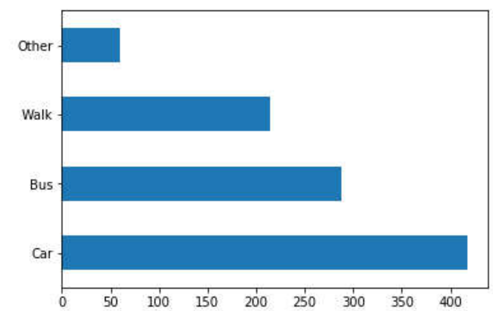


Sparse Categories Example 2 (cont)

```
idx = school_data_df.Travel_to_School.isin(['Board','SkateBoarding','SkateBoard','Boat','Rail','Cycle','Other'])
school_data_df.Travel_to_School[idx] = 'Other'
```

```
C:\Users\Admin\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-
-versus-a-copy
```

```
fig = school_data_df.Travel_to_School.value_counts().plot.barh()
```



Harmless warning; just ignore.

Final Remarks

- You should choose the right data transformations based on :
  1) <u>your</u> data
  2) Modeling methods you intend to use

- Data transformation is also a bit of an art
  - need to try different transformations until find the magic combination
  - experience and intuition help alot

Feature Engineering for Unstructured Data

- Feature engineering is necessary for unstructured attributes because most modeling methods can accept only structured data.
- Each type of unstructured data requires specialized feature engineering techniques
  - **text data**: feature engineering is done using regular expressions and natural language processing (NLP)
  - **image data**: feature engineering is done using computer vision and image processing techniques
  - **Time series data**
  - **...**

Deep Learning

- One of the major strengths of deep neural networks (deep learning) is that they perform feature engineering automatically as part of the modeling process.

Reference

https://elitedatascience.com/feature-engineering