Rebecca Vickery September 7, 2018

Feature Transformation for Machine Learning, a Beginners Guide

When first starting to learn how to optimise machine learning models I would often find, after getting to the model building stage, that I would have to keep going back to revisit the data to better handle the types of features present in the dataset. Over time I have found that one of the first steps to take before building the models is to carefully review the variable types present in the data, and to try to determine up front the best transformation process to take to achieve the optimal model performance.

In the following post I am going to describe the process I take to identify, and transform four common variable types. I am going to be using a dataset taken from the "machine learning with a heart" warm up competition hosted on the https://www.drivendata.org/ website. The full dataset can be downloaded here https://www.drivendata.org/competitions/54/machine-learning-with-a-heart/data/. DrivenData host regular online challenges that are based on solving social problems. I have recently started to engage in some of these competitions in an effort to use some of my skills for a good cause, and also to gain experience with data sets and problems that I don't usually encounter in my day to day work.

Identifying Variable Types

In statistics numerical variables can be characterised into four main types. When starting a machine learning project it is important to determine the type of data that is in each of your features as this can have a significant impact on how the models perform. I have tried to give a simple description of the four types below.

- **Continuous** variables are variables that can have an infinite number of possible values, as opposed to discrete variables which can only have a specified range of values. An example of a continuous variable would be the number of miles that a car has driven in its lifetime.
- **Nominal** variables are categorical values that have 2 or more possible values, but in which the order of those values have no meaning. For example we might use a numerical representation to interpret types of cars say compact has a value of 1, MPV has a value of 2 and the convertible has a value of 3. However, the fact that the compact car has a value of 1 and the convertible has a value of 2 does not mean that mathematically the convertible group is in someway larger than the MPV. It is simply a numerical representation of the category.
- **Dichotomous** variables are again categorical but only have 2 possible values usually 0 and 1. For example we might categorise car ownership as 1 (meaning yes) or 0 (meaning no). When we convert variables into dummy columns (which we will do later in this post) the new features produced also become dichotomous.

- **Ordinal** variables are similar to nominal in that they have 2 or more possible values, the primary difference is that these values have a meaningful order or rank. So in our car example this might be something like engine size where these categories could be ordered in terms of power, 1.2, 1.6, 1.8.

Preparing the data

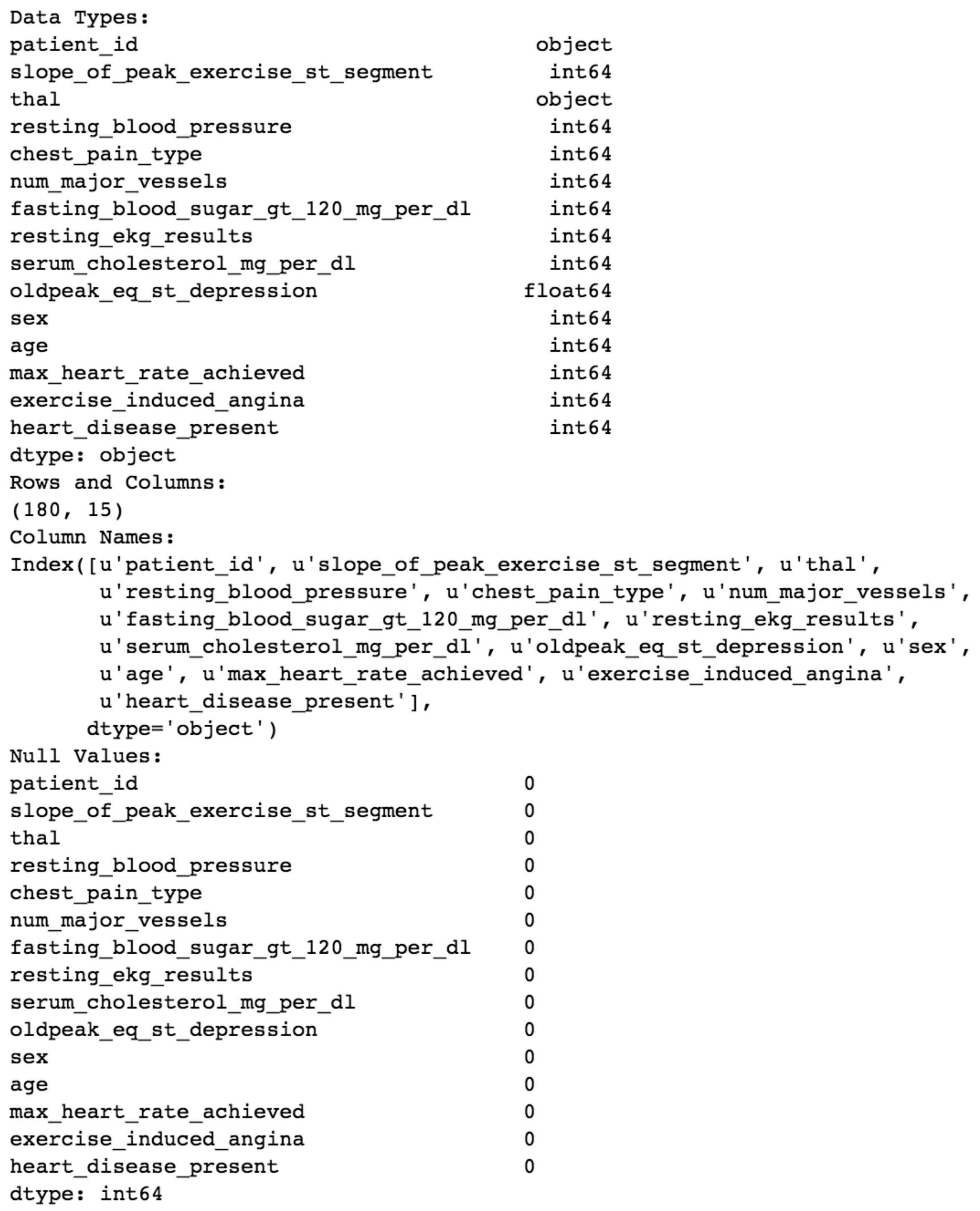
I am going to use our machine learning with a heart dataset to walk through the process of identifying and transforming the variable types. I have downloaded and read the csv files into a Jupyter Notebook. Next I run the following function to get a snapshot of the composition of the data.

```
import pandas as pd

def quick_analysis(df):
 print("Data Types:")
 print(df.dtypes)
 print("Rows and Columns:")
 print(df.shape)
 print("Column Names:")
 print(df.columns)
 print("Null Values:")
 print(df.apply(lambda x: sum(x.isnull()) / len(df)))

quick_analysis(train)
```

This produces the following output.

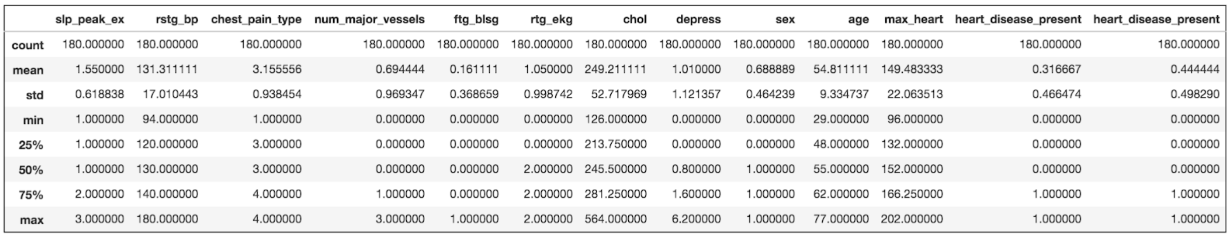


```
Data Types:
patient_id                              object
slope_of_peak_exercise_st_segment        int64
thal                                    object
resting_blood_pressure                   int64
chest_pain_type                          int64
num_major_vessels                        int64
fasting_blood_sugar_gt_120_mg_per_dl     int64
resting_ekg_results                      int64
serum_cholesterol_mg_per_dl              int64
oldpeak_eq_st_depression               float64
sex                                      int64
age                                      int64
max_heart_rate_achieved                  int64
exercise_induced_angina                  int64
heart_disease_present                    int64
dtype: object
Rows and Columns:
(180, 15)
Column Names:
Index([u'patient_id', u'slope_of_peak_exercise_st_segment', u'thal',
       u'resting_blood_pressure', u'chest_pain_type', u'num_major_vessels',
       u'fasting_blood_sugar_gt_120_mg_per_dl', u'resting_ekg_results',
       u'serum_cholesterol_mg_per_dl', u'oldpeak_eq_st_depression', u'sex',
       u'age', u'max_heart_rate_achieved', u'exercise_induced_angina',
       u'heart_disease_present'],
      dtype='object')
Null Values:
patient_id                              0
slope_of_peak_exercise_st_segment       0
thal                                    0
resting_blood_pressure                  0
chest_pain_type                         0
num_major_vessels                       0
fasting_blood_sugar_gt_120_mg_per_dl    0
resting_ekg_results                     0
serum_cholesterol_mg_per_dl             0
oldpeak_eq_st_depression                0
sex                                     0
age                                     0
max_heart_rate_achieved                 0
exercise_induced_angina                 0
heart_disease_present                   0
dtype: int64
```

This tells me that I have a small dataset of only 180 rows and that there are 15 columns. One of the features is non-numeric, and will therefore need to be transformed prior to applying most machine learning libraries. There are no null values so I don’t need to worry about treating those. Before processing the dataset I also drop the “patient_id” column for now as that is non-numeric and will not be used in any of the training or prediction stages.

I then run the pandas describe function to produce some quick descriptive statistics.

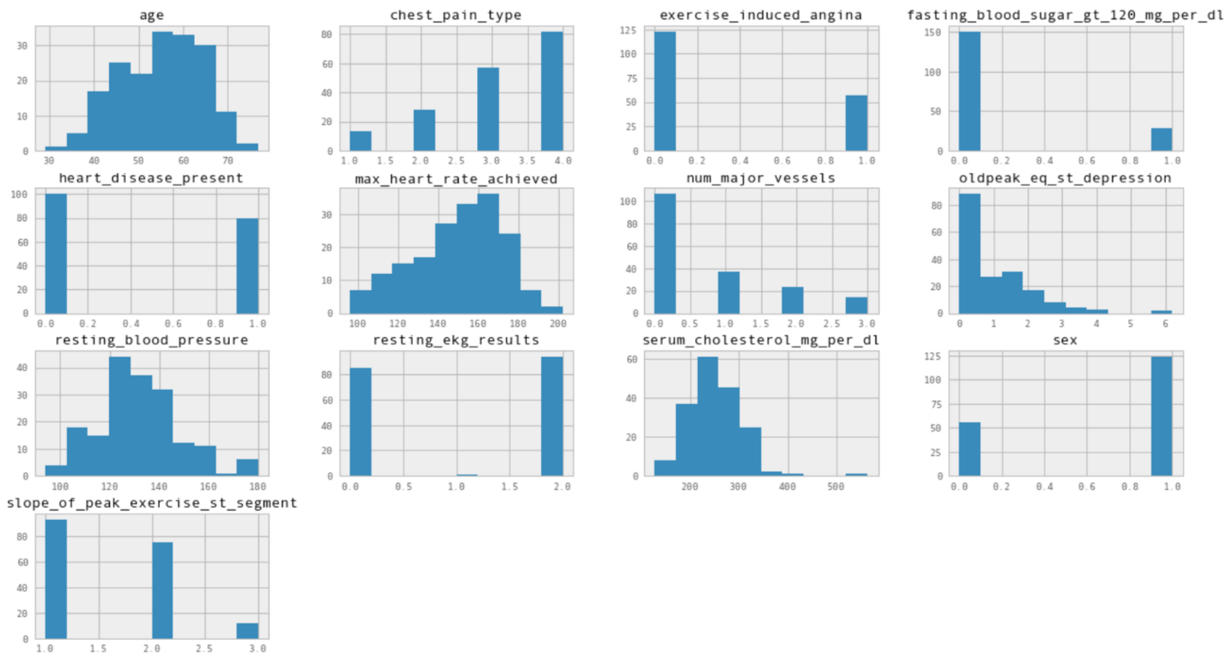
```
train.describe()
```



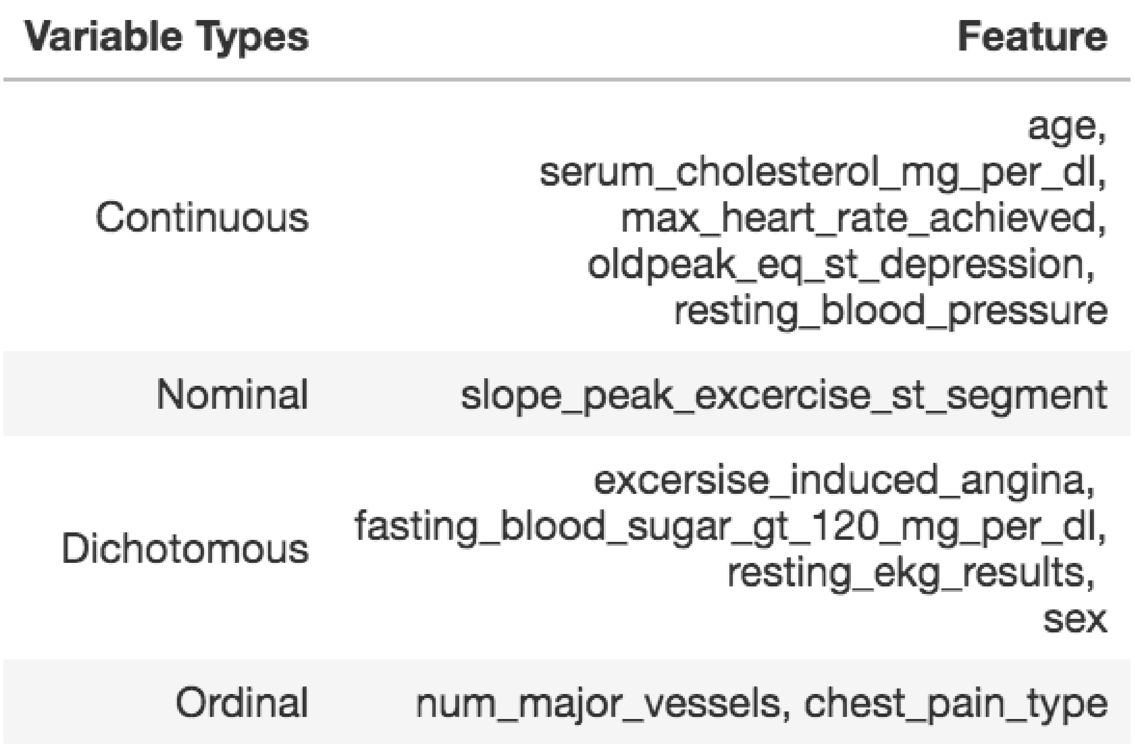
	slp_peak_ex	rstg_bp	chest_pain_type	num_major_vessels	ftg_blsg	rtg_ekg	chol	depress	sex	age	max_heart	heart_disease_present	heart_disease_present
count	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000	180.000000
mean	1.550000	131.311111	3.155556	0.694444	0.161111	1.050000	249.211111	1.010000	0.688889	54.811111	149.483333	0.316667	0.444444
std	0.618838	17.010443	0.938454	0.969347	0.368659	0.998742	52.717969	1.121357	0.464239	9.334737	22.063513	0.466474	0.498290
min	1.000000	94.000000	1.000000	0.000000	0.000000	0.000000	126.000000	0.000000	0.000000	29.000000	96.000000	0.000000	0.000000
25%	1.000000	120.000000	3.000000	0.000000	0.000000	0.000000	213.750000	0.000000	0.000000	48.000000	132.000000	0.000000	0.000000
50%	1.000000	130.000000	3.000000	0.000000	0.000000	2.000000	245.500000	0.800000	1.000000	55.000000	152.000000	0.000000	0.000000
75%	2.000000	140.000000	4.000000	1.000000	0.000000	2.000000	281.250000	1.600000	1.000000	62.000000	166.250000	1.000000	1.000000
max	3.000000	180.000000	4.000000	3.000000	1.000000	2.000000	564.000000	6.200000	1.000000	77.000000	202.000000	1.000000	1.000000

To categorise the variable types in the dataset I run the following code which produces histograms of all the numerical features. You can easily see from the resulting output which features are continuous and dichotomous. The continuous features display a continuous distribution pattern, whilst the dichotomous features have only two bars. The nominal and ordinal variables can sometimes be trickier to determine, and may require some further knowledge of the dataset or some specific domain knowledge. In the case of a machine learning competition such as this I would suggest referring to any data dictionary that may be supplied, if there isn't one (as is the case here) then a combination of intuition and trial and error may be needed.

```
import matplotlib.pyplot as plt
train[train.dtypes[(train.dtypes=="float64")|(train.dtypes=="int64")]
                        .index.values].hist(figsize=[11,11])
```



I have characterised the features into the four types in the table below. I can now make some decisions as to the transformation steps I will take in order to prepare the data for training and prediction.



Variable Types	Feature
Continuous	age, serum_cholesterol_mg_per_dl, max_heart_rate_achieved, oldpeak_eq_st_depression, resting_blood_pressure
Nominal	slope_peak_excercise_st_segment
Dichotomous	excersise_induced_angina, fasting_blood_sugar_gt_120_mg_per_dl, resting_ekg_results, sex
Ordinal	num_major_vessels, chest_pain_type

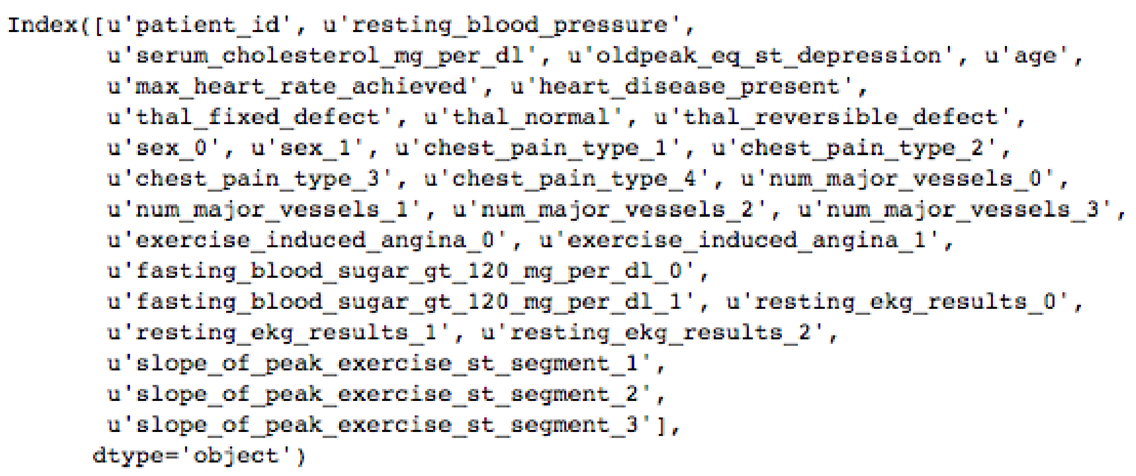
Dummy Variables

As mentioned earlier in this post any non-numerical values need to be converted to integers or floats in order to be utilised in most machine learning libraries. For low cardinality variables the best approach is usually to turn the feature into one column per unique value, with a 0 where the value is not present and a 1 where it is. These are referred to as dummy variables.

This technique is also usually best applied to any nominal variables. As these have no intrinsic order, if we don't apply this first, the machine learning algorithm may incorrectly look for a relationship in the order of these values.

Pandas has a nice function for this called get_dummies(). In the below code I have used this to convert all nominal and non-numeric features into new columns. You can see from the output that several new columns have been created and the original columns have been dropped.

```
dummy_cols = ['thal', 'chest_pain_type', 'num_major_vessels',
              'exercise_induced_angina', 'fasting_blood_sugar_gt_120_mg_per_dl',
              'resting_ekg_results', 'slope_of_peak_exercise_st_segment']
train = pd.get_dummies(train, columns = dummy_cols)
```



```
Index([u'patient_id', u'resting_blood_pressure',
       u'serum_cholesterol_mg_per_dl', u'oldpeak_eq_st_depression', u'age',
       u'max_heart_rate_achieved', u'heart_disease_present',
       u'thal_fixed_defect', u'thal_normal', u'thal_reversible_defect',
       u'sex_0', u'sex_1', u'chest_pain_type_1', u'chest_pain_type_2',
       u'chest_pain_type_3', u'chest_pain_type_4', u'num_major_vessels_0',
       u'num_major_vessels_1', u'num_major_vessels_2', u'num_major_vessels_3',
       u'exercise_induced_angina_0', u'exercise_induced_angina_1',
       u'fasting_blood_sugar_gt_120_mg_per_dl_0',
       u'fasting_blood_sugar_gt_120_mg_per_dl_1', u'resting_ekg_results_0',
       u'resting_ekg_results_1', u'resting_ekg_results_2',
       u'slope_of_peak_exercise_st_segment_1',
       u'slope_of_peak_exercise_st_segment_2',
       u'slope_of_peak_exercise_st_segment_3'],
      dtype='object')
```

Feature Scaling

The continuous variables in our dataset are at varying scales. For instance if you refer back to the histograms above you can see that the variable “oldpeak_eq_st_depression” ranges from 0 to 6, whilst “max_heart_rate_achieved” ranges from 100 to 200. This poses a problem for many popular machine learning algorithms which often use Euclidian distance between data points to make the final predictions. Standardising the scale for all continuous variables can often result in an increase in performance of machine learning models.

There are a number of methods for performing feature scaling in python. My preferred method is to use the Sci-Kit Learn MinMaxScaler function. Which transforms the scale so that all values in the features range from 0 to 1. I have included some code that does this below.

```
from sklearn import preprocessing

n_test = train[['serum_cholesterol_mg_per_dl','max_heart_rate_achieved',
                'oldpeak_eq_st_depression', 'resting_blood_pressure']]
cols_to_norm = ['serum_cholesterol_mg_per_dl','max_heart_rate_achieved',
                'oldpeak_eq_st_depression', 'resting_blood_pressure']

x = n_test.values
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
n_test = pd.DataFrame(x_scaled, columns=cols_to_norm)
l_test = train.drop(['serum_cholesterol_mg_per_dl','max_heart_rate_achieved',
                'oldpeak_eq_st_depression', 'resting_blood_pressure'], axis=1)
train = pd.concat([n_test, l_test], axis=1)
train.columns
```

Binning

You will notice from the code above that I did not include the continuous variable “age” in the feature scaling transformation. The reason for this is that age is an example of a feature type that might benefit from transformation into a discrete variable. In this example we can use bucketing or binning to transform the feature into a list of meaningful categories.

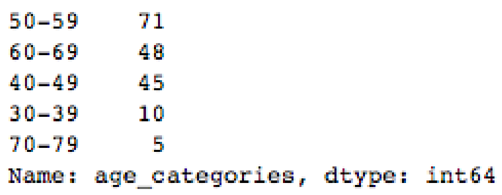
In the code below I have specified intuitive categories based on the distribution in the data. This uses the pandas cut function which takes in a list of bins, group_names and the data frame. This functions returns the original data frame with a new “age_categories” feature. This column can then be turned into a number of dummy columns using the method previously described.

```
bins = [30, 40, 50, 60, 70, 80]

group_names = ['30-39', '40-49', '50-59', '60-69', '70-79']

age_categories = pd.cut(train['age'], bins, labels=group_names)
train['age_categories'] = pd.cut(train['age'], bins, labels=group_names)
age_categories

pd.value_counts(train['age_categories'])
```



```
50-59    71
60-69    48
40-49    45
30-39    10
70-79     5
Name: age_categories, dtype: int64
```

What we now have is a dataset where all columns are non-numeric. We have created several new features, and transformed existing features into formats that should help to improve the performance of any machine learning models we may now use. Feature transformation is an important first step in the machine learning process and this can often have a significant impact on model performance. I have outlined here the first steps I would take in the process to logically think about how to treat the different variables I have. Once in the model building phase I will almost always go back and tweak the data using different methods to try to boost the accuracy of the models. However, I find that by following these steps at the beginning this often reduces the time I spend going back to the transformation stages.