**Data Preprocessing:**

**Transforming Skewed Continuous Features:** A dataset may sometimes contain at least one feature whose values tend to lie near a single number, but will also have a non-trivial number of vastly larger or smaller values than that single number. Algorithms can be sensitive to such distributions of values and can underperform if the range is not properly normalized.

It is common practice to apply a [logarithmic transformation](https://en.wikipedia.org/wiki/Data_transformation_(statistics)) on the data so that the very large and very small values do not negatively affect the performance of a learning algorithm. Using a logarithmic transformation significantly reduces the range of values caused by outliers. Care must be taken when applying this transformation however: The logarithm of 0 is undefined, so we must translate the values by a small amount above 0 to apply the logarithm successfully.

### Normalizing Numerical Features: In addition to performing transformations on features that are highly skewed, it is often good practice to perform some type of scaling on numerical features. Normalization ensures that each feature is treated equally when applying supervised learners. Note that once scaling is applied, observing the data in its raw form will no longer have the same original meaning.

### Examples: MinMaxScaler, StandardScaler

**Encoding Categorical Values:** Learning algorithms expect input to be numeric, which requires that non-numeric features (called categorical variables) be converted. One popular way to convert categorical variables is by using the one-hot encoding scheme. One-hot encoding creates a "dummy" variable for each possible category of each non-numeric feature. For example, assume someFeature has three possible entries: A, B, or C. We then encode this feature into someFeature\_A, someFeature\_B and someFeature\_C.

|  | **someFeature** |  | **someFeature\_A** | **someFeature\_B** | **someFeature\_C** |
| --- | --- | --- | --- | --- | --- |
| 0 | B |  | 0 | 1 | 0 |
| 1 | C | ----> one-hot encode ----> | 0 | 0 | 1 |
| 2 | A |  | 1 | 0 | 0 |

### Use [pandas.get\_dummies()](http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html?highlight=get_dummies" \l "pandas.get_dummies" \t "_blank) to perform one-hot encoding.

Additionally, as with the non-numeric features, we need to convert the non-numeric target label to numerical values for the learning algorithm to work. For example, if we have two possible categories for this label ("<=50K" and ">50K"), we can avoid using one-hot encoding and simply encode these two categories as 0 and 1.

income = income\_raw.replace({'<=50K':0 , '>50K':1})

**Shuffle and Split Data:** Once all the *categorical variables* have been converted into numerical features, and all numerical features have been normalized. We split the data (both features and their labels) into training and test sets. 80% of the data will be used for training and 20% for testing.

# Import train\_test\_splitfrom sklearn.model\_selection import train\_test\_split# Split the 'features' and 'income' data into training and testing setsX\_train, X\_test, y\_train, y\_test = train\_test\_split(features\_final, income, test\_size = 0.2, random\_state = 0)

**Evaluation Metrics:**

**Confusion Matrix:**

|  |  |  |
| --- | --- | --- |
|  | **Correct** | **Not Correct** |
| **Selected** | True Positive(tp) | False Positive(fp) |
| **Not Selected** | False Negative(fn) | True Negative(tn) |

**Accuracy** = (tp+tn) / (tp+fp+fn+tn)

**Precision**: The percentage of selected items that are correct.

Precision = tp / (tp+fp)

**Recall**: The percentage of correct items that are selected.

Recall = tp / (tp+fn)

**F-measure:** A combined measure that assesses the Precision – Recall tradeoff is F measure (weighted harmonic mean)

F = (β^2 + 1) \* PR / (β^2 \* P) + R

For β=1, F = 2PR/(P+R)