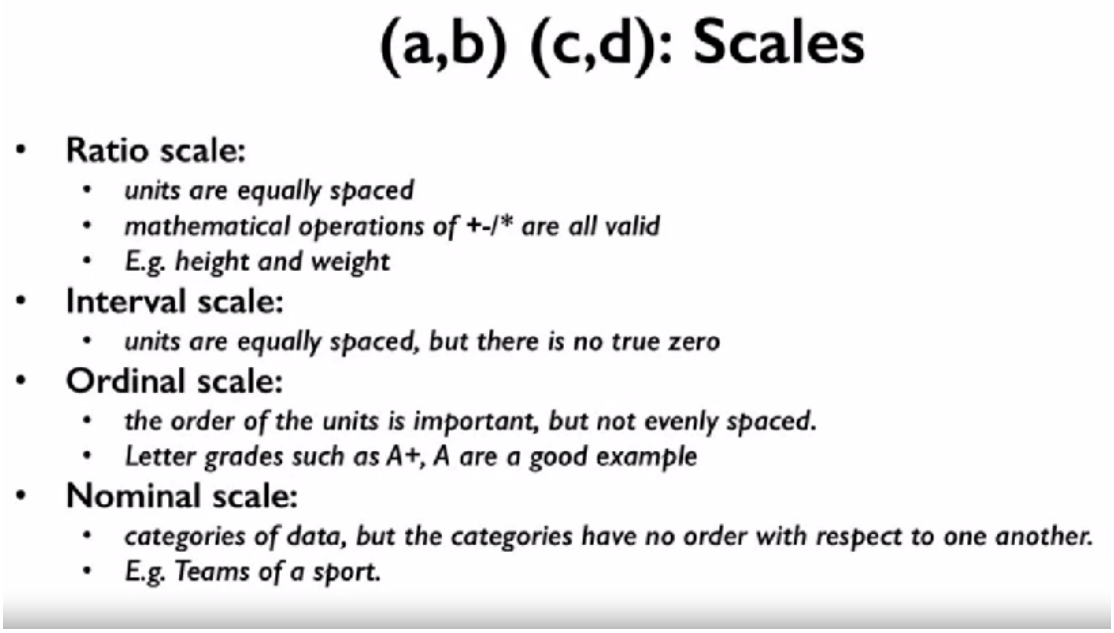
Scales

Now that we've covered many of the mechanics of Pandas, I want to stop and talk for a moment about data types and scales.
We've already seen that Pandas supports a number of different computational data types such as strings, integers and floating point numbers.
What this doesn't capture is what we call the scale of the data.

Let's say that we have a dataframe of students and their academic levels such as being in grade 1 or grade 2 and grade 3.
There's a difference between a student in grade 1 and a student in grade 2, the same as the difference between a student in grade 8 and a student in grade 9.
Well let's think about the final exam scores these students might get on assignments.
Is the difference between an A and an A minus the same as the difference between an A minus and a B plus?
At the University of Michigan at least, the answer is usually no.

We've intuitively seen some different scales, and as we move through data cleaning and statistical analysis and machine learning, it's important to clarify our knowledge and terminology.

As a data scientist there are four scales that it's worth knowing:



The first is a **ratio scale.
In the ratio scale the measurements units are equally spaced and mathematical operations, such as subtract, division, and multiplication are all valid.
Good examples of ratio scale measurements might be the height and weight.

The next scale is the **interval scale.
In the interval scale the measurement units are equally spaced like the ratio scale. But there's no clear absence of value.
That is there isn't a true zero, and so operation such as multiplication and division are not valid.
An example of the interval scale might be the temperatures measured in Celsius or Fahrenheit.
Since there's never an absence of temperature and 0 degrees is a meaningful value of temperature.
The direction on a campus might be another good example where 0 degrees on the campus doesn't indicate a lack of direction.
For most of the work you do at data mining, the differences between the ratio and interval scales might not be clearly apparent or important than the algorithm you're applying.
But it's important to have this clear in your mind when you're applying advanced statistical tests.

The next scale is the **ordinal scale, and this is also important. In the ordinal scale the order of values is important but the differences between the values are not equally spaced.
Here is a grading method that is used in many classes at the University of Michigan as a great example, where letter grades are given with pluses and minuses.
But when you compare this to percentage values you see that a letter by itself covers 4% of the available grades and that a letter with a plus or minus is usually just 4% for the available grades.
Based on this, it would be odd if there were as many students who received an A plus or A minus as there were who received a straight A.
Ordinal data is very common in machine learning and can sometimes be a challenge to work with.

The last scale I'll mention is the **nominal scale which is often just called categorical data.
Here the names of teams in a sport might be good example.
There are a limited number of teams but changing their order or playing mathematical function to them is meaningless.

Categorical values are very common and we generally refer to categories where there are only two possible values as binary.

So why did I stop talking about Pandas and jump into this discussion of scale.
Well, given how important they are in statistics and machine learning, Pandas has a number of interesting functions to deal with converting between measurement scales.

Let's start first with nominal data, which in Pandas is called categorical data.
Panda is actually has a built in type for categorical data and you could set a column of your data to categorical data by using the as type method.
As type tries to change the underlying type of your data, in this case to category data.

You can further change this to ordinal data by passing in an ordered flags set to true and passing in the categories in an ordered fashion.
Here's an example.
Let's create a dataframe of letter grades in the descending order.
We can also set an index value of some more course grain measure.

```
df = pd.DataFrame(['A+', 'A', 'A-', 'B+', 'B', 'B-', 'C+', 'C', 'C-', 'D+', 'D'],
                  index=['excellent', 'excellent', 'excellent', 'good', 'good', 'good', 'ok', 'ok', 'ok', 'poor', 'poor'])
df.rename(columns={0: 'Grades'}, inplace=True)
df
```

	Grades
excellent	A+
excellent	A
excellent	A-
good	B+
good	B
good	B-
ok	C+
ok	C
ok	C-
poor	D+
poor	D

Now when we instruct Pandas to render this as categorical data, we see that the D type has been set as category and that there are 11 different categories.

```
df['Grades'].astype('category').head()
```

```
excellent    A+
excellent     A
excellent    A-
good         B+
good          B
Name: Grades, dtype: category
Categories (11, object): [A, A+, A-, B, ..., C+, C-, D, D+]
```

If we want to indicate to Pandas that this data is in a logical order, we pass the ordered equals true flag and we see those reflected in the category D type using the less than sign.

```
grades = df['Grades'].astype('category',
                             categories=['D', 'D+', 'C-', 'C', 'C+', 'B-', 'B', 'B+', 'A-', 'A', 'A+'],
                             ordered=True)
grades.head()
```

```
excellent    A+
excellent     A
excellent    A-
good         B+
good          B
Name: Grades, dtype: category
Categories (11, object): [D < D+ < C- < C ... B+ < A- < A < A+]
```

What can you do with this?
Well, ordinal data has ordering so it can help you with the Boolean masking.
For instance, if we have our list of grades and we compared them with a C.
If we did this lexographically, we would that a C+ and a C- are both actually greater than a C.

```
grades > 'C'
```

```
excellent     True
excellent     True
excellent     True
good          True
good          True
good          True
ok            True
ok           False
ok           False
poor         False
poor         False
Name: Grades, dtype: bool
```

Instead of coding each of these to something which is lexographical, like a number, we can indicate that there's a clear order to the data. And then broadcasting will work as we expect.
We can then use a certain set of mathematical operators like minimum, maximum and others on the ordinal data.

Sometimes it's useful to represent categorical values as each being a column with a true or a false as to whether that category applies.
This is especially common in feature extraction, which is a topic in the third course in this specialization.

Variables with a Boolean value are typically called dummy variables.
And pandas has a built-in function called get dummies, which will convert the values of a single column into multiple columns of 0's and 1's, indicating the presence of a dummy variable.

There's one more function on scales that I'd like to talk about.
And that's on reducing a value which is on the interval or ratio scale, like a number grade, into one that is categorical like a letter grade.
Now, this might seem a bit counter intuitive to you since you're losing information about the value.
But it's useful on a couple of places.
First, if you're visualizing the frequencies of categories, and this can be an extremely useful approach and histograms are regularly used with converted interval or racial data.
And we'll take a look at histograms in the second course in this specialization.

Second, if you're using a machine learning classification approach on data, then you need to be using categorical data.
So reducing dimensionality is useful there too.
panda says a function called cut, which takes in argument which is some real like structure of a column or a data frame or a series.
It also takes a number of bins to be used and all bins are kept at equal spacing.

Let's go back to our census data for an example. We saw that we could group by state, then aggregate to get a list of the average county size by state.
If we further apply cut to this with say 10 bins, we can see that the states listed as categoricals using the average county size.
Here we see the states like Alabama and Alaska fall into the same category, while California and the District of Colombia fall into a very different category.
Now, cutting is just one way to build categories from your data, and there's many other methods.

```
df = pd.read_csv('census.csv')
df = df[df['SUMLEV']==50]
df = df.set_index('STNAME').groupby(level=0)['CENSUS2010POP'].agg({'avg': np.average})
pd.cut(df['avg'],10)
```

```
STNAME
Alabama                 (11706.0871, 75333.413]
Alaska                  (11706.0871, 75333.413]
Arizona                (390320.176, 453317.529]
Arkansas                (11706.0871, 75333.413]
California             (579312.234, 642309.586]
Colorado                (75333.413, 138330.766]
Connecticut            (390320.176, 453317.529]
Delaware               (264325.471, 327322.823]
District of Columbia   (579312.234, 642309.586]
Florida                (264325.471, 327322.823]
Georgia                 (11706.0871, 75333.413]
Hawaii                 (264325.471, 327322.823]
Idaho                   (11706.0871, 75333.413]
Illinois                (75333.413, 138330.766]
Indiana                 (11706.0871, 75333.413]
Iowa                    (11706.0871, 75333.413]
Kansas                  (11706.0871, 75333.413]
Kentucky                (11706.0871, 75333.413]
Louisiana               (11706.0871, 75333.413]
Maine                   (75333.413, 138330.766]
Maryland               (201328.118, 264325.471]
Name: avg, dtype: category
Categories (10, object): [(11706.0871, 75333.413] < (75333.413, 138330.766] < (138330.766, 201328.118] < (201328.118, 264325.47
1] ... (390320.176, 453317.529] < (453317.529, 516314.881] < (516314.881, 579312.234] < (579312.234, 642309.586]]
```

For instance, cut gives you interval data, where the spacing between each category is equal sized.
But sometimes you want to form categories based on frequency.
You want the number of items in each bin to be the same, instead of the spacing between bins.
It really depends on the shape of your data and what you're planning to do with it.