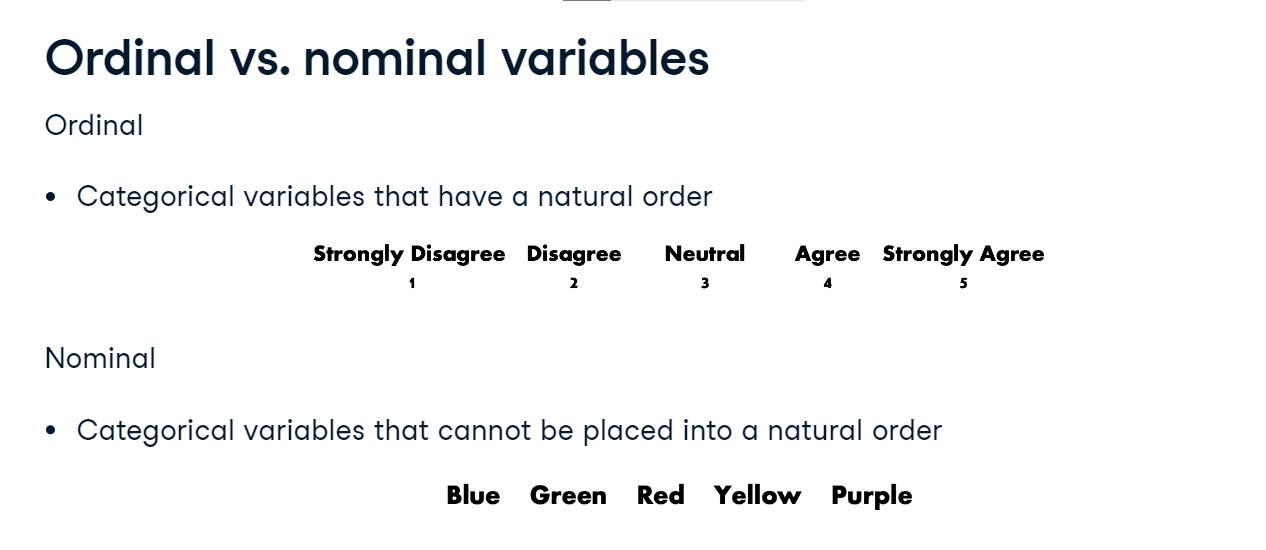
**What does it mean to be "categorical"?**

A variable is usually considered categorical if it contains a finite number of distinct groups - or categories. Generally, the number of categories and the corresponding names are already known. In research fields, this type of data is also known as qualitative data. On the other hand, numerical data, also know as quantitative data, is expressed using a value and is usually in the form of a measurement. This course is completely focused on working with categorical data.

**Ordinal vs. nominal variables**

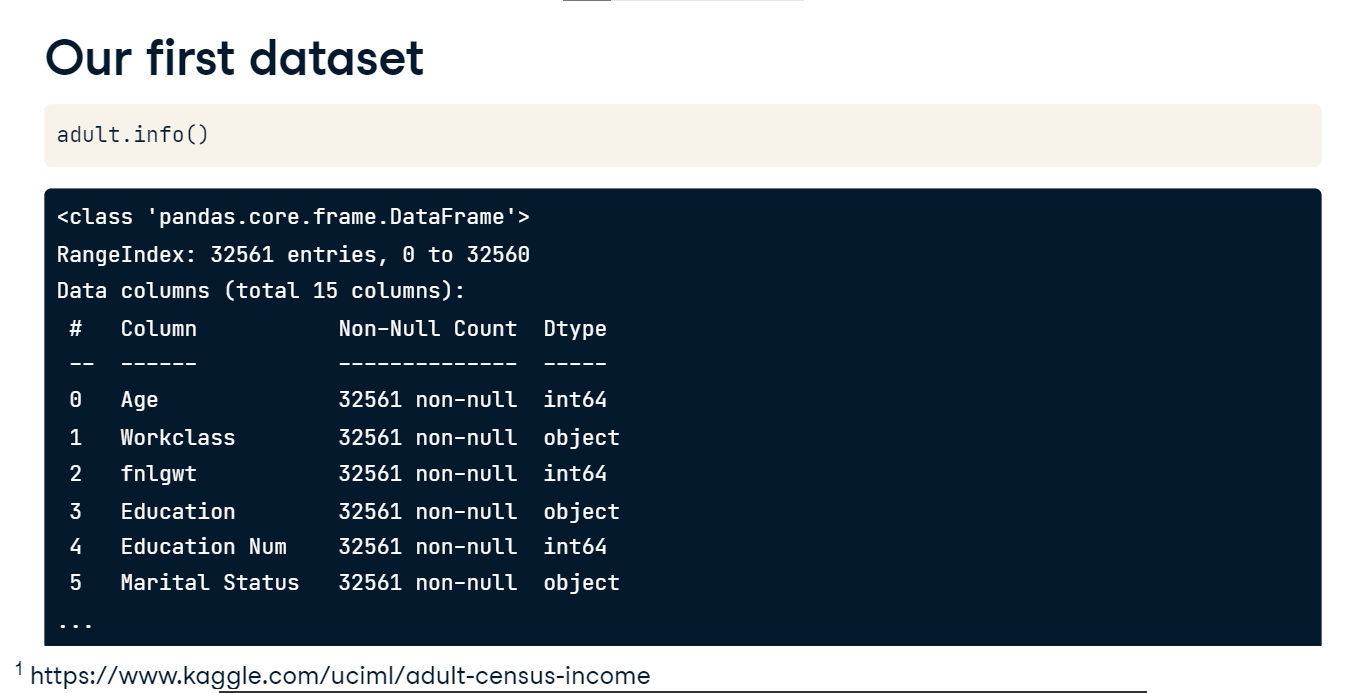
Categorical data can be further broken down into two different types, ordinal and nominal. You can think of ordinal variables as having an order. When categories within a variable have a natural rank order, the variable is considered ordinal. We have all filled out surveys that have ordinal options, such as choices ranging from strongly disagree up through strongly agree. On the flip side, nominal variables are those that cannot be placed into a natural order. A good example here is when a survey asks what your favorite color is from a list of options.



**Our first dataset**

To begin working with categorical variables, let's take a look at our first dataset. The adult census income dataset contains information on US adults and whether or not an adult makes over $50,000 annually. We have used the pandas method info to take a look at the dataset's variables and their data types, or dtypes for short. We quickly see that there are over 32,000 entries and 15 total columns. Take a look at the marital status column, which has the dtype - object. A dtype of object is how pandas stores strings and is a good indicator that a variable might be categorical.

1. 1 <https://www.kaggle.com/uciml/adult-census-income>

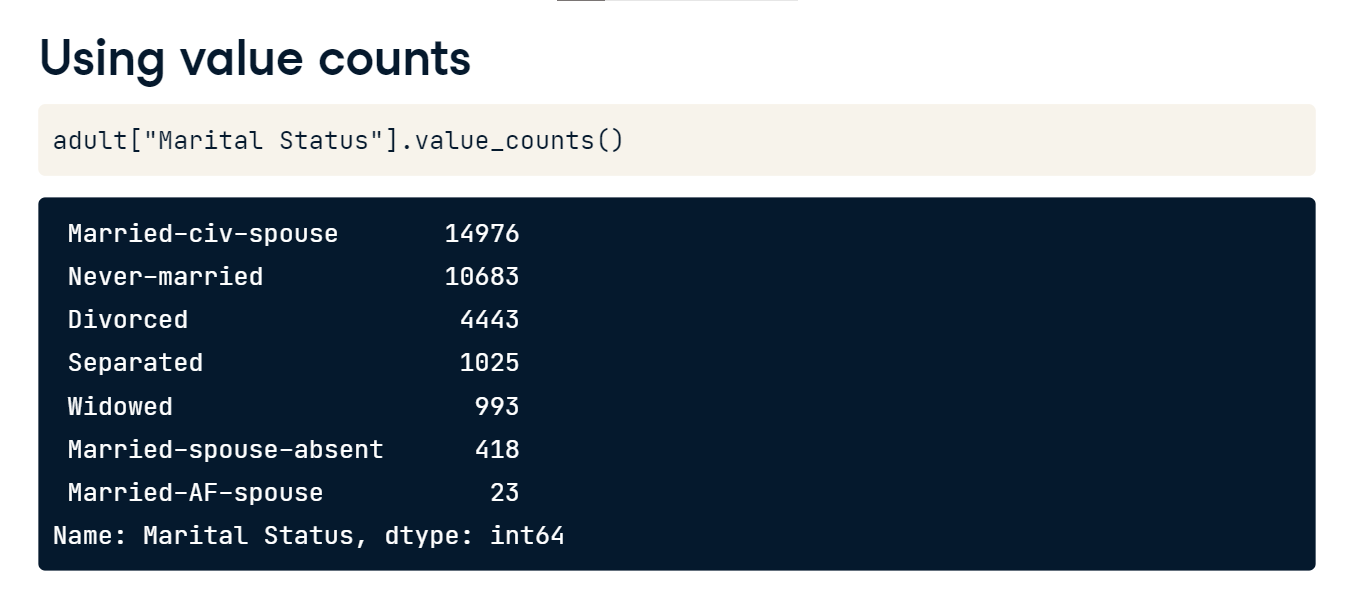


**Using describe**

We can explore the marital status column in more detail by using the describe method on the pandas Series Marital Status. We see that there are seven unique values, with the married-civ-spouse option being the most common entry with almost 15,000 occurrences.

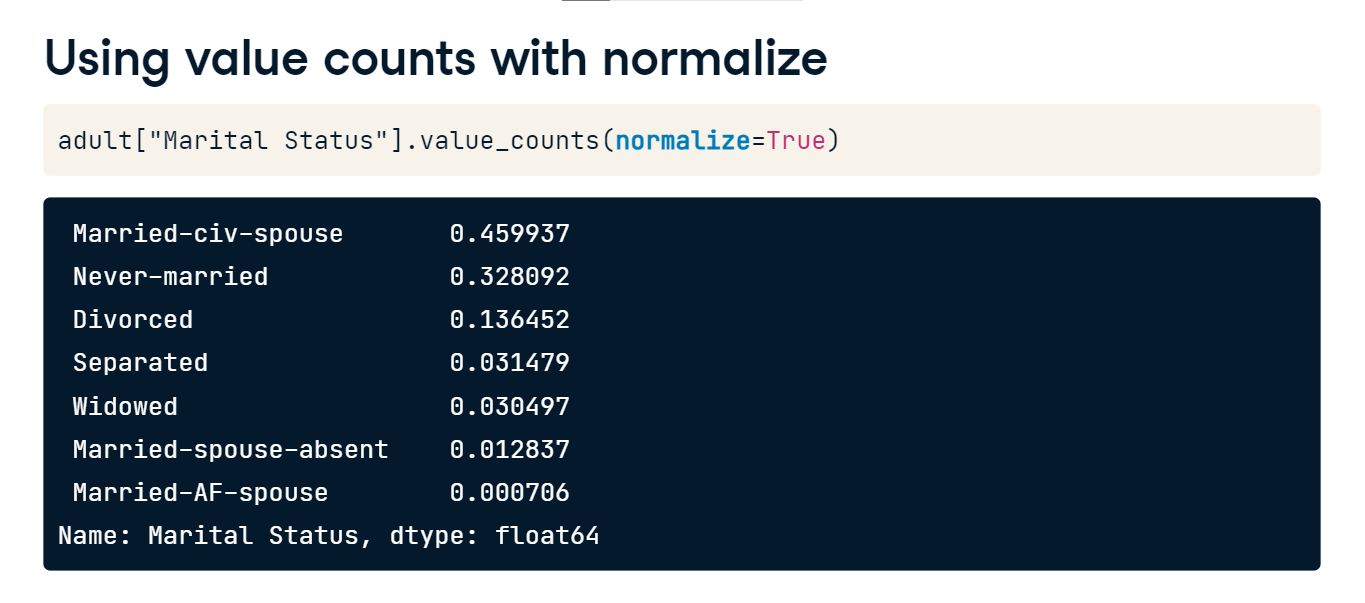
**Using value counts**

Another way to explore the marital status column is to use the value counts method. This method prints out a frequency table of the values found in a pandas Series. We still see the 15,000 entries for married-civ-spouse, but we also see the remaining counts for each unique value in the marital status column.



**Using value counts with normalize**

Although the value counts method has a few parameters, the most commonly used parameter is definitely normalize. By setting normalize to equal True, the output will contain relative frequency values instead of the counts of the unique values. The values shown are the proportions of all responses in a pandas Series that equal to a specific value. In this example, married-civ-spouse makes up 46% of all responses.



**Categorical data in pandas**

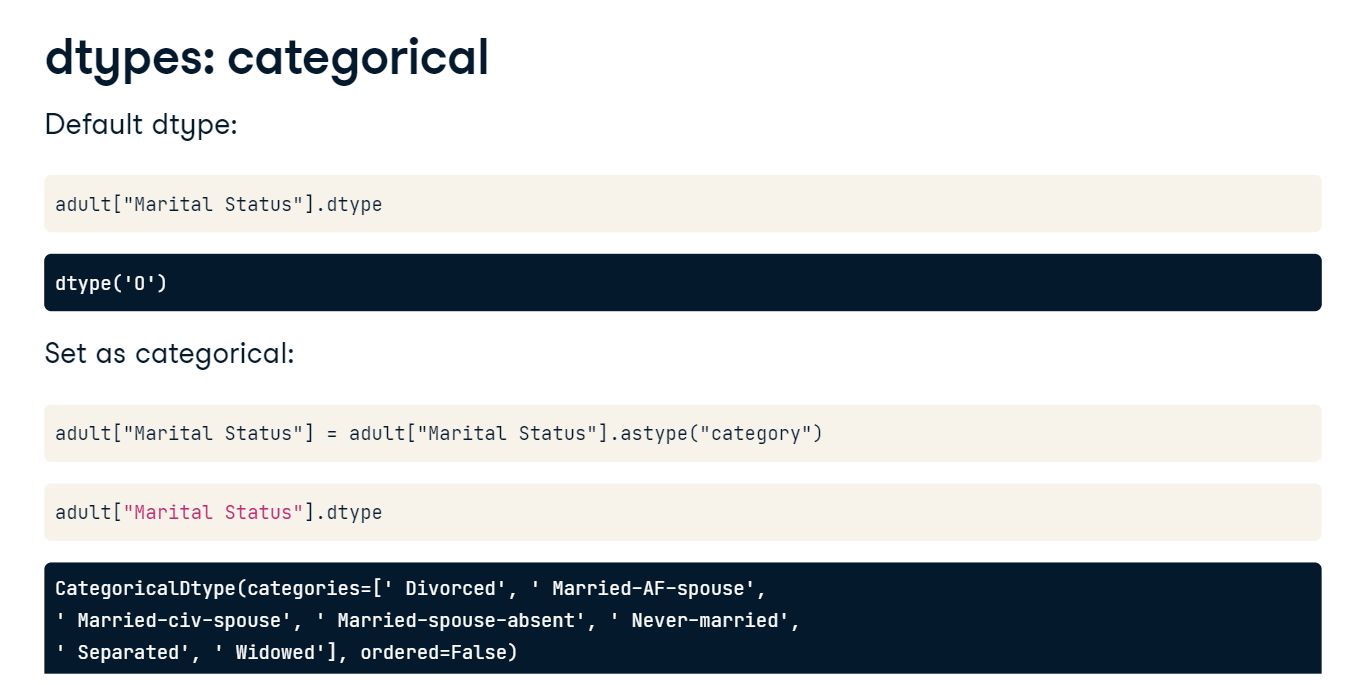
The most common way of working with categorical data in Python is through using pandas. Let's take a quick look at how pandas handles categorical data.

**dtypes: object**

After reading in the adult census income dataset, we can print out the dtypes of each column by using the dtypes property. We see dtypes of primarily int64, which is one way to save values as integers, and dtypes of object. By default, pandas tries to infer the data type of each column. In this dataset, the numerical values have been assigned a dtype of int64, while the columns containing strings are stored with the object dtype.

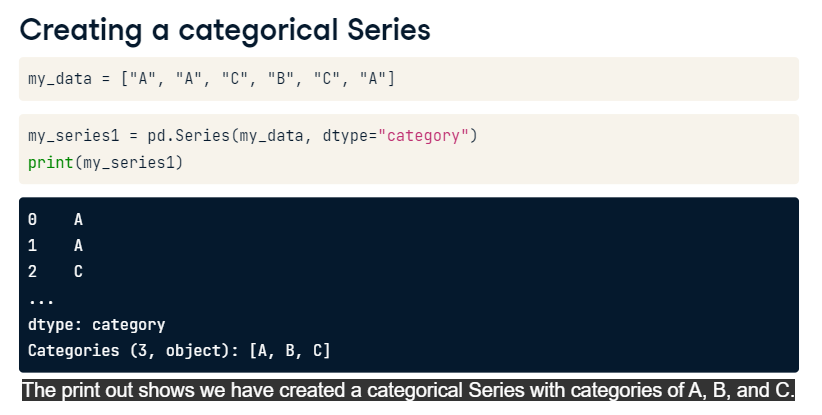
**dtypes: categorical**

By default, columns containing strings are not stored using pandas' category dtype, as not every column containing strings needs to be categorical. Let's look at the dtype for Marital Status. We use the dtype property, as opposed to the dtypes property, since we are working with a Series and not a DataFrame. Pandas uses a capital O to represent the object dtype. We can convert this to the categorical dtype using the astype method and specifying category. This time, the output is quite different. Pandas is telling us that the variable is now saved as categorical and is providing the list of the categories found in the Series. Finally, notice in the print out that ordered equals false - indicating there is currently no order for these categories.



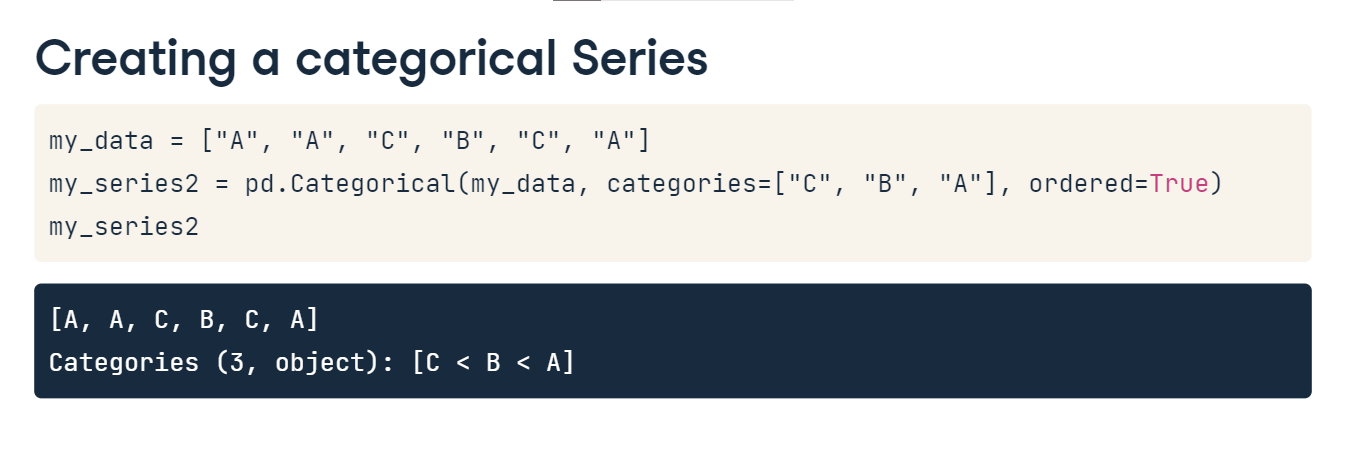
**Creating a categorical Series**

There are two ways to create a categorical pandas Series when your data is not already in a DataFrame format. First we can use pd-dot-series on a list or array of data and set the dtype argument to category. The print out shows we have created a categorical Series with categories of A, B, and C.



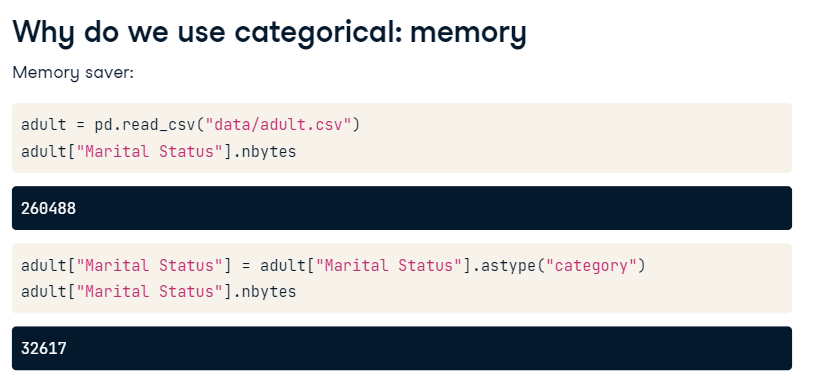
**Creating a categorical Series**

The second way is to use pd-dot-categorical. We are showing this alternative way because it allows us to tell pandas that the categories have a logical order by setting the ordered argument equal to true. The order is set by using the categories parameter. Whichever order you list the categories in will be the order of the categories going forward. Notice that the print out states that the order is C, then B, then A, which matches the order we used when creating the categorical Series.



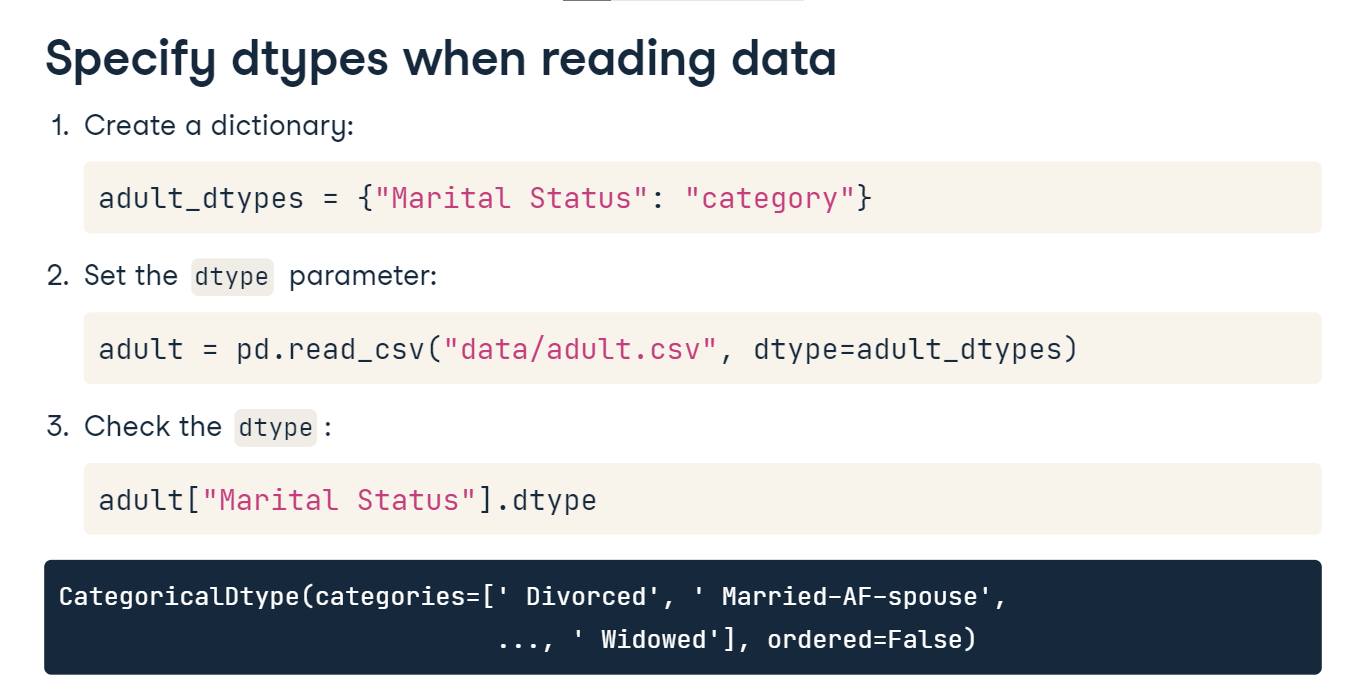
**Why do we use categorical: memory**

There are a few reasons why storing pandas Series with a dtype of categorical is useful. Let's look at the easiest one to quantify: it is a huge memory saver. Take a look at the number of bytes python uses to store the Marital Status column when saved as an object compared to when it is saved as a categorical column. We can do this using the nbytes attribute. In this example, using a categorical dtype reduced the memory footprint by a factor of eight. Since pandas will by default load all the data into your computers memory, reducing your memory footprint can be helpful when dealing with large datasets.



**Specify dtypes when reading data**

If you know the data types of columns before reading in a dataset, it is good practice to specify at least some of the column dtypes. This can be done by creating a dictionary with column names as keys and data types as values. By setting the dtype parameter equal to this dictionary, pandas will set the dtypes of any columns that match keys found in the dictionary. We can quickly check the Marital Status dtype using the dtype method

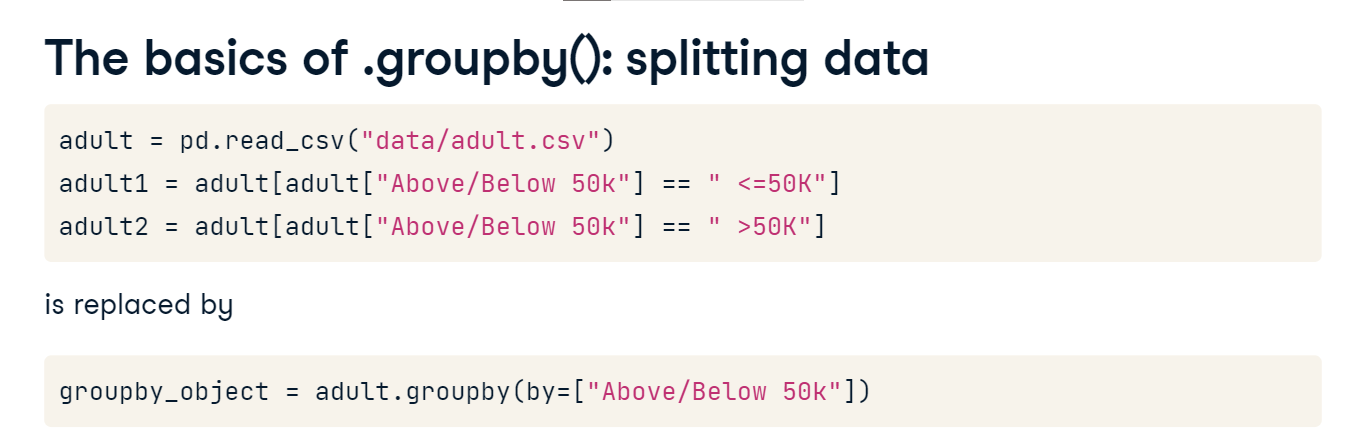


**Grouping data by category in pandas**

When using categorical variables in pandas, there are a few essential methods used for exploring data that every pandas user must master. We will cover one of these methods now.

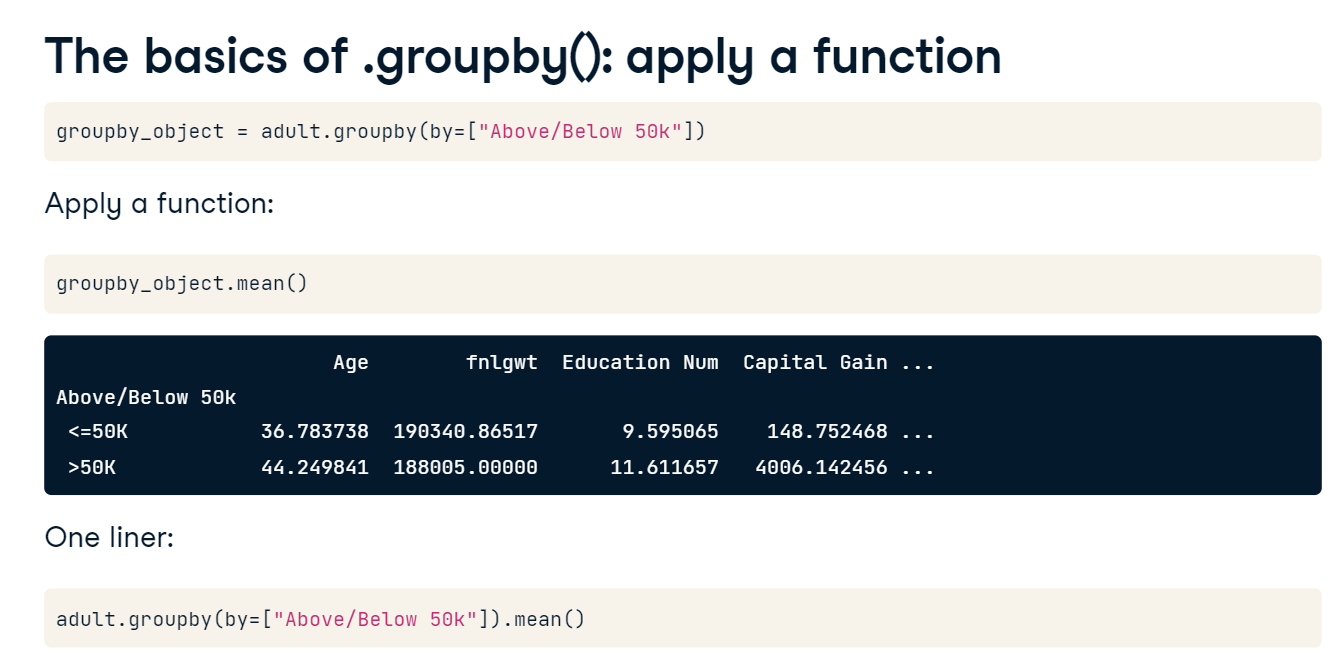
**The basics of .groupby(): splitting data**

The groupby method splits data across the unique values of the column specified. One of the most common forms of data analysis is to split data across groups and to perform analysis on each group. The groupby process is essentially equal to creating multiple DataFrames, one for each value in the specified variable. In the adult income dataset, the Above/Below 50k variable has two categories. This data can be split into two DataFrames using two separate filters. However, this can be replaced by a one liner using groupby. The first parameter in the groupby method is by - which is used to specify the variable or variables to split the data by. This does not have to be a list, but we will use a list throughout for consistency.



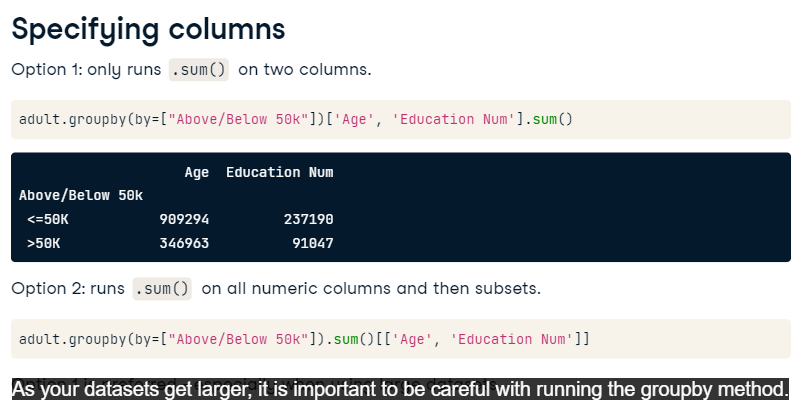
**The basics of .groupby(): apply a function**

After calling groupby, you can specify a function that should be applied to the split data. Common functions such as sum, count, and mean can be used, but custom functions can be applied as well. When using numerical functions, such as mean, the function will only be applied to the numerical columns. Notice that the mean of each numerical column in the adult dataset was calculated across the two groups of the Above/Below 50k column. Just as a quick note, you don't have to create a groupby object to run a function. You can chain the creation and the function call in a one liner.



**Specifying columns**

When using large datasets, it is not always possible to apply a groupby call to all columns. It is important to specify the columns of interest before calling the function to apply. Consider this group by call. We group by the above-below 50k column, subset to just the age and education number columns, and then calculate the sum of these two columns. Alternatively, we get the same result if we group by above-below 50k, call the sum function, and then subset the results to just the age and education number values. Option 1 tends to be much faster, as it will not perform the sum calculation on the other numerical columns. As your datasets get larger, it is important to be careful with running the groupby method.



**Groupby multiple columns**

The groupby method can be called on more than one variable. By specifying two columns with the by parameter, the groupby method will create subsets of the data for all combinations of the variables specified. Here we are using the size function to check how many rows of the data fall into each grouping. The variable Above/Below 50k has two categories, while Marital Status has seven. This creates 14 different groupings. When calling groupby on multiple variables, it is important to check the size of each combination to make sure there are enough rows per combination to do analysis. For example, the combination of more than 50k and Married-AF-spouse only has 10 rows of data, which is not very much. If no rows exist for a combination of the variables, it will not be added to the groupby object.

.