Module 6: Data types and tidy data

Tidy data

Let's do some tidy exercise first. This is one of the non-tidy dataset assembled by Hadley Wickham (check out here for more datasets, explanation, and R code).

Let's take a look at this small dataset:
https://raw.githubusercontent.com/tidyverse/tidyr/4c0a8d0fdb9372302fcc57ad995d57a43d9e4337/vignettes/pew.c

In [ ]:

```
import pandas as pd
```

In [ ]:

```
pew_df = pd.read_csv('https://raw.githubusercontent.com/tidyverse/tidyr/4c0a8d0fdb9372302fcc57ad995d57a43d9e4337/vignettes/pew.csv')
pew_df
```

Out[ ]:

	religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
0	Agnostic	27	34	60	81	76	137	122	109	84	96
1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	Don’t know/refused	15	14	15	11	10	35	21	17	18	116
5	Evangelical Prot	575	869	1064	982	881	1486	949	723	414	1529
6	Hindu	1	9	7	9	11	34	47	48	54	37
7	Historically Black Prot	228	244	236	238	197	223	131	81	78	339
8	Jehovah's Witness	20	27	24	24	21	30	15	11	6	37
9	Jewish	19	19	25	25	30	95	69	87	151	162
10	Mainline Prot	289	495	619	655	651	1107	939	753	634	1328
11	Mormon	29	40	48	51	56	112	85	49	42	69
12	Muslim	6	7	9	10	9	23	16	8	6	22
13	Orthodox	13	17	23	32	32	47	38	42	46	73
14	Other Christian	9	7	11	13	13	14	18	14	12	18
15	Other Faiths	20	33	40	46	49	63	46	40	41	71
16	Other World Religions	5	2	3	4	2	7	3	4	4	8
17	Unaffiliated	217	299	374	365	341	528	407	321	258	597

This dataset is about the relationships between income and religion, assembled from a research by the Pew Research Center. You can read more details here. Is this dataset tidy or not? Why?

Yes, many of the columns are values, not variable names. How should we fix it?

Pandas provides a convenient function called melt. You specify the `id_vars` that are variable columns, and `value_vars` that are value columns, and provide the name for the variable as well as the name for the values

`value_vars` that are value columns, and provide the name for the variable as well as the name for the values.

Q: so please go ahead and tidy it up! I'd suggest to use the variable name "income" and value name "frequency"

In [17]:

```
# TODO: put your code here

pew_tidy_df = pd.melt(pew_df, id_vars=['religion'], value_vars=['<$10k', '$10-20k', '$20
-30k', '$30-40k','$40-50k', '$50-75k','$75-100k', '$100-150k','>150k', 'Don\'t know/refu
sed' ],var_name='income', value_name='frequency')
```

In [22]:

```
pew_tidy_df.sample(10)
```

Out[22]:

	religion	income	frequency
138	Muslim	$100-150k	8
87	Other Faiths	$40-50k	49
47	Mormon	$20-30k	48
55	Atheist	$30-40k	52
38	Buddhist	$20-30k	30
56	Buddhist	$30-40k	34
131	Evangelical Prot	$100-150k	723
107	Unaffiliated	$50-75k	528
35	Unaffiliated	$10-20k	299
88	Other World Religions	$40-50k	2

If you were successful, you'll have something like this:

In [ ]:

```
pew_tidy_df.sample(10)
```

Out[ ]:

	religion	income	frequency
58	Don’t know/refused	$30-40k	11
9	Jewish	<$10k	19
111	Catholic	$75-100k	949
165	Catholic	Don't know/refused	1489
66	Muslim	$30-40k	10
84	Muslim	$40-50k	9
99	Jewish	$50-75k	95
139	Orthodox	$100-150k	42
7	Historically Black Prot	<$10k	228
163	Atheist	Don't know/refused	76

Data types

Let's talk about data types briefly. Understanding data types is not only important for choosing the right visualizations, but also important for efficient computing and storage of data. You may not have thought about how pandas represent data in memory. A Pandas `Dataframe` is essentially a bunch of `Series`, and those `Series` are essentially `numpy` arrays. An array may contain a fixed-length items such as integers or variable

`Series` are essentially `numpy` arrays. An array may contain a fixed-length items such as integers or variable length items such as strings. Putting some efforts to think about the correct data type can potentially save a lot of memory as well as time.

A nice example would be the categorical data type. If you have a variable that only has several possible values, it's essentially a categorical data. Take a look at the `income` variable.

In [ ]:

```
pew_tidy_df.income.value_counts()
```

Out[ ]:

```
$30-40k               18
Don't know/refused    18
$100-150k             18
$20-30k               18
$50-75k               18
<$10k                 18
$10-20k               18
$75-100k              18
>150k                 18
$40-50k               18
Name: income, dtype: int64
```

These were the column names in the original non-tidy data. The value can take only one of these income ranges and thus it is a categorical data. What is the data type that pandas use to store this column?

In [ ]:

```
pew_tidy_df.income.dtype
```

Out[ ]:

```
dtype('O')
```

The `O` means that it is an object data type, which does not have a fixed size like integer or float. The series contains a sort of pointer to the actual text objects. You can actually inspect the amount of memory used by the dataset.

In [ ]:

```
pew_tidy_df.memory_usage()
```

Out[ ]:

```
Index          128
religion      1440
income        1440
frequency     1440
dtype: int64
```

In [ ]:

```
pew_tidy_df.memory_usage(deep=True)
```

Out[ ]:

```
Index           128
religion      12780
income        11700
frequency      1440
dtype: int64
```

What's going on with the `deep=True` option? When you don't specify `deep=True`, the memory usage method just tells you the amount of memory used by the numpy arrays in the pandas dataframe. When you pass `deep=True`, it tells you the total amount of memory by including the memory used by all the text objects. So, the `religion` and `income` columns occupies almost ten times of memory than the `frequency` column, which is simply an array of integers.

In [ ]:

```
pew_tidy_df.frequency.dtype
```

Out[ ]:

```
dtype('int64')
```

Is there any way to save up the memory? Note that there are only 10 categories in the income variable. That means we just need 10 numbers to represent the categories! Of course we need to store the names of each category, but that's just one-time cost. The simplest way to convert a column is using `astype` method.

In [ ]:

```
income_categorical_series = pew_tidy_df.income.astype('category')
# you can do pew_tidy_df.income = pew_tidy_df.income.astype('category')
```

Now, this series has the `CategoricalDtype` dtype.

In [ ]:

```
income_categorical_series.dtype
```

Out[ ]:

```
CategoricalDtype(categories=['$10-20k', '$100-150k', '$20-30k', '$30-40k', '$40-50k',
                  '$50-75k', '$75-100k', '<$10k', '>150k',
                  'Don't know/refused'],
                 ordered=False)
```

How much memory do we use?

In [ ]:

```
income_categorical_series.memory_usage(deep=True)
```

Out[ ]:

```
1278
```

In [ ]:

```
pew_tidy_df.income.memory_usage(deep=True)
```

Out[ ]:

```
11828
```

In [24]:

```
pew_df
```

Out[24]:

	religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
0	Agnostic	27	34	60	81	76	137	122	109	84	96
1	Atheist	12	27	37	52	35	70	73	59	74	76
2	Buddhist	27	21	30	34	33	58	62	39	53	54
3	Catholic	418	617	732	670	638	1116	949	792	633	1489
4	Don't know/refused	15	14	15	11	10	35	21	17	18	116
5	Evangelical Prot	575	869	1064	982	881	1486	949	723	414	1529
6	Hindu	1	9	7	9	11	34	47	48	54	37
7	Historically Black	228	244	236	238	197	223	131	81	78	339

	Prot religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
8	Jehovah's Witness	20	27	24	24	21	30	15	11	6	37
9	Jewish	19	19	25	25	30	95	69	87	151	162
10	Mainline Prot	289	495	619	655	651	1107	939	753	634	1328
11	Mormon	29	40	48	51	56	112	85	49	42	69
12	Muslim	6	7	9	10	9	23	16	8	6	22
13	Orthodox	13	17	23	32	32	47	38	42	46	73
14	Other Christian	9	7	11	13	13	14	18	14	12	18
15	Other Faiths	20	33	40	46	49	63	46	40	41	71
16	Other World Religions	5	2	3	4	2	7	3	4	4	8
17	Unaffiliated	217	299	374	365	341	528	407	321	258	597

We have reduced the memory usage by almost 10 fold! Not only that, because now the values are just numbers, it will be much faster to match, filter, manipulate. If your dataset is huge, this can save up a lot of space and time.

If the categories have ordering, you can specify the ordering too.

In [27]:

```
from pandas.api.types import CategoricalDtype
income_type = CategoricalDtype(categories=["Don't know/refused", '<$10k', '$10-20k', '$20-30k', '$30-40k',
                                           '$40-50k', '$50-75k', '$75-100k', '$100-150k', '>150k'], ordered=True)
income_type
```

Out[27]:

```
CategoricalDtype(categories=['Don't know/refused', '<$10k', '$10-20k', '$20-30k',
                  '$30-40k', '$40-50k', '$50-75k', '$75-100k', '$100-150k',
                  '>150k'],
                 ordered=True)
```

In [34]:

```
pew_tidy_df.income.astype(income_type).dtype
```

Out[34]:

```
CategoricalDtype(categories=['Don't know/refused', '<$10k', '$10-20k', '$20-30k',
                  '$30-40k', '$40-50k', '$50-75k', '$75-100k', '$100-150k',
                  '>150k'],
                 ordered=True)
```

In [43]:

```
income_categorical_series.memory_usage(deep = True)
```

Out[43]:

```
1278
```

This data type now allows you to compare and sort based on the ordering.

Q: ok, now convert both religion and income columns of `pew_tidy_df` as categorical dtype (in place) and show that `pew_tidy_df` now uses much less memory

In [58]:

```
from pandas.api.types import CategoricalDtype

# TODO: put your code here
#religion
```

```
pew_tidy_df['religion'] = pew_tidy_df.religion.astype('category')

#income
pew_tidy_df['income'] = pew_tidy_df.income.astype('category')

#memory usage
pew_tidy_df.memory_usage(deep=True)
```

Out[58]:

```
Index          128
religion      2119
income        1150
frequency     1440
dtype: int64
```

Index 128

religion 2119

income 1150

frequency 1440

dtype: int64

If you want to know more

- Jean-Nicholas Hould: Tidy Data in Python
- Stephen Simmons | Pandas from the Inside
- Data school: How do I make my pandas DataFrame smaller and faster?