MET MA 603: SAS Programming and Applications

Arrays

The Need for Arrays

Sometimes we need to perform the same operation on many different variables. For example, consider the City_temps_by_month_fahr dataset below. If we need to convert each temperature from Fahrenheit to Celsius, we would need to write a statement for each month.

VIEWTABLE: Work.City_temps_by_month_fahr

	City	Month_1	Month_2	Month_3	Month_4	Month_5	Month_6	Month_7	Month_8
1	Washingon, D.C.	42	44	52	64	75	83	87	84
2	Boston	36	39	45	56	66	76	81	80
3	Miami	76	78	80	83	87	89	91	91
4	San Francisco	57	60	62	63	64	66	67	68
5	Minneapolis	24	29	41	58	69	79	83	80

```
Month_1 = (Month_1 – 32) / 9 * 5;
Month_2 = (Month_2 – 32) / 9 * 5;
Month_3 = (Month_3 – 32) / 9 * 5;
etc.
etc.
```

The Need for Arrays (cont.)

There are two problems with this. First, it is tedious to enter and review so much repetitive code. Second, it increases the chance of typos.

A better way of getting the result needed in the previous slide would be if we could instruct SAS to do the same conversion calculation on each of the Month_i variables.

```
Month(i) = (Month(i) – 32) / 9 * 5;
```

This can be achieved with Arrays.

An **Array** is a convenient way of temporarily identifying a group of variables for processing within a Data step.

Defining Arrays

Arrays are defined within the Data step. We must supply SAS with the name of the array, number of items, data type, and name of each variable separated by spaces.

```
ARRAY name (n) $ q1 q2 q3;
ARRAY name (n)    num1 num2 num3;
```


The variables within in array must be all of one data type.

Arrays are temporary structures created and used within a Data step. At the end of the Data step the array is gone.

Array names follow the usual SAS naming convention.

Shorthand for Listing Variables

SAS can recognize two shorthand ways of listing variables.

The first method relies on there being an obvious pattern in the variable names. Listing the first and last variables separated by a dash allows for the variable names that would fall in between to be omitted.


```
ARRAY months (12)    Month_1 - Month_12;
```


The second method relies on the order in which the variables are organized in the input dataset. The first and last variables are listed, with a double dash in between. This is equivalent to listing all of the variables in the dataset that fall in between first and last.

```
ARRAY months (12)    Month_1 -- Month_12;
```

Arrays and Do Loops

Arrays almost always are found in combination with a Do Loop, which makes it possible to iterate through each variable in the array. This is what gives arrays their efficiency.


```
data   city_temps_by_month_cels;
set              city_temps_by_month_fahr;
array  temps (12) month1 - month12;
do i = 1 to 12;
       temps (i) = (temps (i) - 32) / 1.8 ;
end;
run;
```

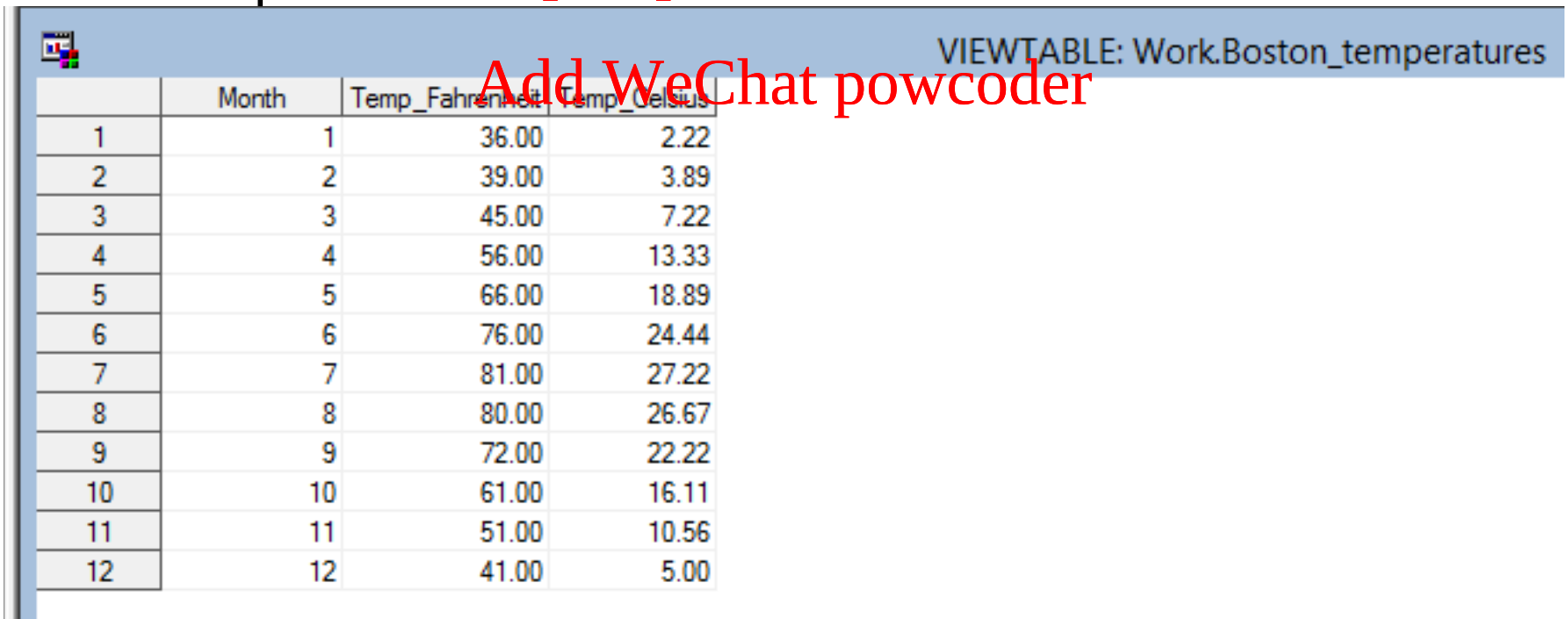

Think of the array as a short-hand way of referring to each variable, without needing to actually type in each different name.

Practice

Using city_temps_by_month_fahr dataset, calculate the average of the 12 monthly temperatures for each city. Calculate the maximum and minimum monthly temperature for each city. Use the Fahrenheit scale.

Practice

Using city_temps_by_month_fahr dataset, create a new dataset that has the monthly temperatures for the city of Boston, with the month and temperature as variables, as shown below. To achieve this result you will need to combine several other tools in addition to arrays. Include both temperature scales.



VIEWTABLE: Work.Boston_temperatures

	Month	Temp_Fahrenheit	Temp_Celsius
1	1	36.00	2.22
2	2	39.00	3.89
3	3	45.00	7.22
4	4	56.00	13.33
5	5	66.00	18.89
6	6	76.00	24.44
7	7	81.00	27.22
8	8	80.00	26.67
9	9	72.00	22.22
10	10	61.00	16.11
11	11	51.00	10.56
12	12	41.00	5.00

Readings

- Textbook section 3.11