

Conjoint Analysis

Warren F. Kuhfeld

Abstract

Conjoint analysis is used to study consumers' product preferences and simulate consumer choice. This chapter describes conjoint analysis and provides examples using SAS. Topics include metric and non-metric conjoint analysis, efficient experimental design, data collection and manipulation, holdouts, brand by price interactions, maximum utility and logit simulators, and change in market share.[†]

Introduction

Conjoint analysis is used to study the factors that influence consumers' purchasing decisions. Products possess attributes such as price, color, ingredients, guarantee, environmental impact, predicted reliability, and so on. Consumers typically do not have the option of buying the product that is best in every attribute, particularly when one of those attributes is price. Consumers are forced to make *trade-offs* as they decide which products to purchase. Consider the decision to purchase a car. Increased size generally means increased safety and comfort. The trade off is an increase in cost and environmental impact and a decrease in gas mileage and maneuverability. Conjoint analysis is used to study these trade-offs.

Conjoint analysis is a popular marketing research technique. It is used in designing new products, changing or repositioning existing products, evaluating the effects of price on purchase intent, and simulating market share. Refer to Green and Rao (1971) and Green and Wind (1975) for early introductions to conjoint analysis, Louviere (1988) for a more recent introduction, and Green and Srinivasan (1990) for a review article.

Conjoint Measurement

Conjoint analysis grew out of the area of *conjoint measurement* in mathematical psychology. Conjoint measurement is used to investigate the joint effect of a set of independent variables on an ordinal-scale-of-measurement dependent variable. The independent variables are typically nominal and sometimes interval-scaled variables. Conjoint measurement simultaneously finds a monotonic scoring of the dependent variable and numerical values for each level of each independent variable. The goal is to monotonically transform the ordinal values to equal the sum of their attribute level values. Hence, conjoint measurement is used to derive an interval variable from ordinal data. The conjoint measurement model is a mathematical model, not a statistical model, since it has no statistical error term.

[†]Copies of this chapter (TS-722H) and all of the macros are available on the web http://support.sas.com/techsup/tnote/tnote_stat.html#market.

Conjoint Analysis

Conjoint analysis is based on a main effects analysis-of-variance model. Subjects provide data about their preferences for hypothetical products defined by attribute combinations. Conjoint analysis decomposes the judgment data into components, based on qualitative attributes of the products. A numerical *part-worth utility* value is computed for each level of each attribute. Large part-worth utilities are assigned to the most preferred levels, and small part-worth utilities are assigned to the least preferred levels. The attributes with the largest part-worth utility range are considered the most important in predicting preference. Conjoint analysis is a statistical model with an error term and a loss function.

Metric conjoint analysis models the judgments directly. When all of the attributes are nominal, the metric conjoint analysis is a simple main-effects ANOVA with some specialized output. The attributes are the independent variables, the judgments comprise the dependent variable, and the part-worth utilities are the β 's, the parameter estimates from the ANOVA model. The following is a metric conjoint analysis model for three factors:

$$y_{ijk} = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

where

$$\sum \beta_{1i} = \sum \beta_{2j} = \sum \beta_{3k} = 0$$

This model could be used, for example, to investigate preferences for cars that differ on three attributes: mileage, expected reliability, and price. The y_{ijk} term is one subject's stated preference for a car with the i th level of mileage, the j th level of expected reliability, and the k th level of price. The grand mean is μ , and the error is ϵ_{ijk} . The predicted utility for the ijk product is:

$$\hat{y}_{ijk} = \hat{\mu} + \hat{\beta}_{1i} + \hat{\beta}_{2j} + \hat{\beta}_{3k}$$

Nonmetric conjoint analysis finds a monotonic transformation of the preference judgments. The model, which follows directly from conjoint measurement, iteratively fits the ANOVA model until the transformation stabilizes. The R^2 increases during every iteration until convergence, when the change in R^2 is essentially zero. The following is a nonmetric conjoint analysis model for three factors:

$$\Phi(y_{ijk}) = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

where $\Phi(y_{ijk})$ designates a monotonic transformation of the variable y .

The R^2 for a nonmetric conjoint analysis model will always be greater than or equal to the R^2 from a metric analysis of the same data. The smaller R^2 in metric conjoint analysis is not necessarily a disadvantage, since results should be more stable and reproducible with the metric model. Metric conjoint analysis was derived from nonmetric conjoint analysis as a special case. Today, metric conjoint analysis is probably used more often than nonmetric conjoint analysis.

In the SAS System, conjoint analysis is performed with the SAS/STAT procedure TRANSREG (transformation regression). Metric conjoint analysis models are fit using ordinary least squares, and nonmetric conjoint analysis models are fit using an alternating least squares algorithm (Young, 1981; Gifi,

1990). Conjoint analysis is explained more fully in the examples. The “PROC TRANSREG Specifications” section of this chapter starting on page 585 documents the PROC TRANSREG statements and options that are most relevant to conjoint analysis. The “Samples of PROC TRANSREG Usage” section starting on page 594 shows some typical conjoint analysis specifications. This chapter shows some of the SAS programming that is used for conjoint analysis. Alternatively, there is a marketing research GUI that performs conjoint analysis available from the main display manager PMENU by selecting: **Solutions** → **Analysis** → **Market Research**.

Choice-Based Conjoint

The meaning of the word “conjoint” has broadened over the years from conjoint measurement to conjoint analysis (which at first always meant what we now call nonmetric conjoint analysis) and later to metric conjoint analysis. Metric and nonmetric conjoint analysis are based on a linear ANOVA model. In contrast, a different technique, discrete choice, is based on the nonlinear multinomial logit model. Discrete choice is sometimes referred to as “choice-based conjoint.” This technique is not discussed in this chapter, however it is discussed in detail starting on page 141.

Experimental Design

Experimental design is a fundamental component of conjoint analysis. A conjoint study uses experimental design to create a list of products that vary on an assortment of attributes such as brand, price, size, and so on, and subjects rate or rank the products. There are many examples of making conjoint designs in this chapter. Before you read them, be sure to read the design chapters beginning on pages 47 and 99.

The Output Delivery System

The Output Delivery System (ODS) can be used to customize the output of SAS procedures including PROC TRANSREG, the procedure we use for conjoint analysis. PROC TRANSREG can produce a great deal of information for conjoint analysis, more than we often wish to see. We will use ODS primarily to exclude certain portions of the default conjoint output in which we are usually not interested. This will create a better, more parsimonious display for typical analyses. However, when we need it, we can revert back to getting the full array of information. See page 143 for other examples of customizing output using ODS. You can run the following step once to customize PROC TRANSREG conjoint analysis output.

```
proc template;
  edit Stat.Transreg.ParentUtilities;
    column Label Utility StdErr tValue Probt Importance Variable;
    header title;
    define title; text 'Part-Worth Utilities'; space=1; end;
    define Variable; print=off; end;
  end;
run;
```

Running this step edits the templates for the main conjoint analysis results table and stores a copy in `sasuser`. These changes will remain in effect until you delete them. These changes move the variable label to the first column, turn off printing the variable names, and set the table header to “Part-Worth Utilities”. These changes assume that each effect in the model has a variable label associated with it, so there is no need to print variable names. This will usually be the case. To return to the default output, run the following.

```
* Delete edited template, restore original template;
proc template;
  delete Stat.Transreg.ParentUtilities;
run;
```

By default, PROC TRANSREG prints an ANOVA table for metric conjoint analysis and both univariate and multivariate ANOVA tables for nonmetric conjoint analysis. With nonmetric conjoint analysis, PROC TRANSREG sometimes prints liberal and conservative ANOVA tables. All of the possible ANOVA tables, along with some header notes, can be suppressed by specifying the following statement before running PROC TRANSREG.

```
ods exclude notes anova liberalanova conservanova
      mvanova liberalmvanova conservmvanova;
```

For metric conjoint analysis, this statement can be abbreviated as follows.

```
ods exclude notes mvanova anova;
```

The rest of this section gives more details about what the PROC TEMPLATE step does and why. The rest of this section can be helpful if you wish to further customize the output from TRANSREG or some other procedure. Impatient readers may skip ahead to the candy example on page 489.

We are most interested in the part-worth utilities table in conjoint analysis, which contains the part-worth utilities, their standard errors, and the importance of each attribute. We can first use PROC TEMPLATE to identify the template for the utilities table and then edit the template. First, let’s have PROC TEMPLATE display the templates for PROC TRANSREG. The `source stat.transreg` statement specifies that we want to see PROC TEMPLATE source code for the STAT product and the TRANSREG procedure.

```
proc template;
  source stat.transreg;
run;
```

If we search the results for “Utilities”, we find the template for the table `Stat.Transreg.ParentUtilities`. Here is the template for the part-worth utilities table.

```
define table Stat.Transreg.ParentUtilities;
  notes "Parent Utilities Table for Proc Transreg";
  dynamic FootMessages TitleText;
  column Label Utility StdErr tValue Probt Importance Variable;
  header Title;
  footer Foot;
  define Title;
    text TitleText;
    space = 1;
    spill_margin;
    first_panel;
  end;
```

```

define Label;
    parent = Stat.Transreg.Label;
    style = RowHeader;
end;
define Utility;
    header = "Utility";
    format_width = 7;
    parent = Stat.Transreg.Coefficient;
end;
define StdErr;
    parent = Stat.Transreg.StdErr;
end;
define tValue;
    parent = Stat.Transreg.tValue;
    print = OFF;
end;
define Probt;
    parent = Stat.Transreg.Probt;
    print = OFF;
end;
define Importance;
    header = %nrstr(";Importance;%(%% Utility;Range%)");
    translate _val_=_ into " ";
    format = 7.3;
end;
define Variable;
    parent = Stat.Transreg.Variable;
end;
define Foot;
    text FootMessages;
    just = 1;
    maximize;
end;
control = control;
required_space = 20;
end;

```

Recall that we ran the following step to customize the output.

```

proc template;
    edit Stat.Transreg.ParentUtilities;
        column Label Utility StdErr tValue Probt Importance Variable;
        header title;
        define title; text 'Part-Worth Utilities'; space=1; end;
        define Variable; print=off; end;
    end;
run;

```

We specified the `edit Stat.Transreg.ParentUtilities` statement to name the table that we wish to change. The `column` statement was copied from the PROC TEMPLATE source listing, and it names all of the columns in the table. Some, like `tValue` and `Probt` do not print by default. We will change the `Variable` column to also not print. We redefine `Variable` with the `print=off` option specified. We also redefine the table header to read “Part-Worth Utilities”. The names in the `column` and `header` statements must match the names in the original template.

Chocolate Candy Example

This example illustrates conjoint analysis with rating scale data and a single subject. The subject was asked to rate his preference for eight chocolate candies. The covering was either dark or milk chocolate, the center was either chewy or soft, and the candy did or did not contain nuts. The candies were rated on a 1 to 9 scale where 1 means low preference and 9 means high preference. Conjoint analysis is used to determine the importance of each attribute and the part-worth utility for each level of each attribute.

Metric Conjoint Analysis

After data collection, the attributes and the rating data are entered into a SAS data set.

```

title 'Preference for Chocolate Candies';

data choc;
  input Chocolate $ Center $ Nuts $& Rating;
  datalines;
Dark Chewy Nuts 7
Dark Chewy No Nuts 6
Dark Soft Nuts 6
Dark Soft No Nuts 4
Milk Chewy Nuts 9
Milk Chewy No Nuts 8
Milk Soft Nuts 9
Milk Soft No Nuts 7
;

```

Note that the “&” specification in the `input` statement is used to read character data with embedded blanks.

PROC TRANSREG is used to perform a metric conjoint analysis.

```

ods exclude notes mvanova anova;
proc transreg utilities separators=', ' short;
  title2 'Metric Conjoint Analysis';
  model identity(rating) = class(chocolate center nuts / zero=sum);
run;

```

Printed output from the metric conjoint analysis is requested by specifying the `utilities` option in the `proc` statement. The value specified in the `separators=` option, in this case a comma followed by a blank, is used in constructing the labels for the part-worth utilities in the printed output. With these options, the labels will consist of the `class` variable name, a comma, a blank and the values of the `class` variables. We specify the `short` option to suppress the iteration history. PROC TRANSREG will still print a convergence summary table so we will know if there are any convergence problems. Since this is a metric conjoint analysis, there should be only one iteration and there should not be any problems. We specified `ods exclude notes mvanova anova` to exclude ANOVA information (which we usually want to ignore) and provide more parsimonious output. The analysis variables, the transformation of each variable, and transformation specific options are specified in the `model` statement.

The `model` statement provides for general transformation regression models, so it has a markedly different syntax from other SAS/STAT procedure `model` statements. Variable lists are specified in parentheses after a transformation name. The specification `identity(rating)` requests an `identity` transformation of the dependent variable `Rating`. A transformation name must be specified for all variable lists, even for the dependent variable in metric conjoint analysis, when no transformation is desired. The `identity` transformation of `Rating` will not change the original scoring. An equal sign follows the dependent variable specification, then the attribute variables are specified along with their transformation. The specification

```
class(chocolate center nuts / zero=sum)
```

designates the attributes as `class` variables with the restriction that the part-worth utilities sum to zero within each attribute. A slash must be specified to separate the variables from the transformation option `zero=sum`. The `class` specification creates a main-effects design matrix from the specified variables. This example produces only printed output; later examples will show how to store results in output SAS data sets.

The output is shown next. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG.

Preference for Chocolate Candies
Metric Conjoint Analysis

The TRANSREG Procedure

Dependent Variable Identity(Rating)

Class Level Information

Class	Levels	Values	
Chocolate	2	Dark Milk	
Center	2	Chewy Soft	
Nuts	2	No Nuts Nuts	
Number of Observations Read			8
Number of Observations Used			8

Identity(Rating)

Algorithm converged.

Root MSE	0.50000	R-Square	0.9500
Dependent Mean	7.00000	Adj R-Sq	0.9125
Coeff Var	7.14286		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	7.0000	0.17678	
Chocolate, Dark	-1.2500	0.17678	50.000
Chocolate, Milk	1.2500	0.17678	
Center, Chewy	0.5000	0.17678	20.000
Center, Soft	-0.5000	0.17678	
Nuts, No Nuts	-0.7500	0.17678	30.000
Nuts, Nuts	0.7500	0.17678	

We see **Algorithm converged** in the output indicating no problems with the iterations. We also see $R^2 = 0.95$. The last table displays the part-worth utilities. The part-worth utilities show the most and least preferred levels of the attributes. Levels with positive utility are preferred over those with negative utility. Milk chocolate (part-worth utility = 1.25) was preferred over dark (-1.25), chewy center (0.5) over soft (-0.5), and nuts (0.75) over no nuts (-0.75).

Conjoint analysis provides an approximate decomposition of the original ratings. The predicted utility for a candy is the sum of the intercept and the part-worth utilities. The conjoint analysis model for the preference for chocolate type i , center j , and nut content k is

$$y_{ijk} = \mu + \beta_{1i} + \beta_{2j} + \beta_{3k} + \epsilon_{ijk}$$

for $i = 1, 2$; $j = 1, 2$; $k = 1, 2$; where

$$\beta_{11} + \beta_{12} = \beta_{21} + \beta_{22} = \beta_{31} + \beta_{32} = 0$$

The part-worth utilities for the attribute levels are the parameter estimates $\hat{\beta}_{11}$, $\hat{\beta}_{12}$, $\hat{\beta}_{21}$, $\hat{\beta}_{22}$, $\hat{\beta}_{31}$, and $\hat{\beta}_{32}$ from this main-effects ANOVA model. The estimate of the intercept is $\hat{\mu}$, and the error term is ϵ_{ijk} .

The predicted utility for the ijk combination is

$$\hat{y}_{ijk} = \hat{\mu} + \hat{\beta}_{1i} + \hat{\beta}_{2j} + \hat{\beta}_{3k}$$

For the most preferred milk/chewy/nuts combination, the predicted utility and actual preference values are

$$7.0 + 1.25 + 0.5 + 0.75 = 9.5 = \hat{y} \approx y = 9.0$$

For the least preferred dark/soft/no nuts combination, the predicted utility and actual preference values are

$$7.0 + -1.25 + -0.5 + -0.75 = 4.5 = \hat{y} \approx y = 4.0$$

The predicted utilities are regression predicted values; the squared correlation between the predicted utilities for each combination and the actual preference ratings is the R^2 .

The *importance* value is computed from the part-worth utility range for each factor (attribute). Each range is divided by the sum of all ranges and multiplied by 100. The factors with the largest part-worth utility ranges are the most important in determining preference. Note that when the attributes have a varying number of levels, attributes with the most levels sometimes have inflated importances (Wittink, Krishnamurthi, and Reibstein; 1989).

The importance values show that type of chocolate, with an importance of 50%, was the most important attribute in determining preference.

$$\frac{100 \times (1.25 - -1.25)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 50\%$$

The second most important attribute was whether the candy contained nuts, with an importance of 30%.

$$\frac{100 \times (0.75 - -0.75)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 30\%$$

Type of center was least important at 20%.

$$\frac{100 \times (0.50 - -0.50)}{(1.25 - -1.25) + (0.50 - -0.50) + (0.75 - -0.75)} = 20\%$$

Nonmetric Conjoint Analysis

In the next part of this example, PROC TRANSREG is used to perform a nonmetric conjoint analysis of the candy data set. The difference between requesting a nonmetric and metric conjoint analysis is the dependent variable transformation; a `monotone` transformation of `Rating` variable is requested instead of an `identity` transformation. Also, we did not specify the `short` option this time so that we could see the iteration history table. The `output` statement is used to put the transformed rating into the `out=` output data set.

```
ods exclude notes anova liberalanova conservanova
      mvanova liberalmvanova conservmvanova;
proc transreg utilities separators=', ';
  title2 'Nonmetric Conjoint Analysis';
  model monotone(rating) = class(chocolate center nuts / zero=sum);
  output;
run;
```

Nonmetric conjoint analysis iteratively derives the monotonic transformation of the ratings. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG.

Preference for Chocolate Candies
Nonmetric Conjoint Analysis

The TRANSREG Procedure

Dependent Variable Monotone(Rating)

Class Level Information

Class	Levels	Values
Chocolate	2	Dark Milk
Center	2	Chewy Soft
Nuts	2	No Nuts Nuts
Number of Observations Read		8
Number of Observations Used		8

TRANSREG Univariate Algorithm Iteration History for Monotone(Rating)

Iteration Number	Average Change	Maximum Change	R-Square	Criterion Change	Note
1	0.08995	0.23179	0.95000		
2	0.01263	0.03113	0.96939	0.01939	
3	0.00345	0.00955	0.96981	0.00042	
4	0.00123	0.00423	0.96984	0.00003	
5	0.00050	0.00182	0.96985	0.00000	
6	0.00021	0.00078	0.96985	0.00000	
7	0.00009	0.00033	0.96985	0.00000	
8	0.00004	0.00014	0.96985	0.00000	
9	0.00002	0.00006	0.96985	0.00000	
10	0.00001	0.00003	0.96985	0.00000	Converged

Algorithm converged.

Root MSE	0.38829	R-Square	0.9698
Dependent Mean	7.00000	Adj R-Sq	0.9472
Coeff Var	5.54699		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	7.0000	0.13728	
Chocolate, Dark	-1.3143	0.13728	53.209
Chocolate, Milk	1.3143	0.13728	
Center, Chewy	0.4564	0.13728	18.479
Center, Soft	-0.4564	0.13728	
Nuts, No Nuts	-0.6993	0.13728	28.312
Nuts, Nuts	0.6993	0.13728	

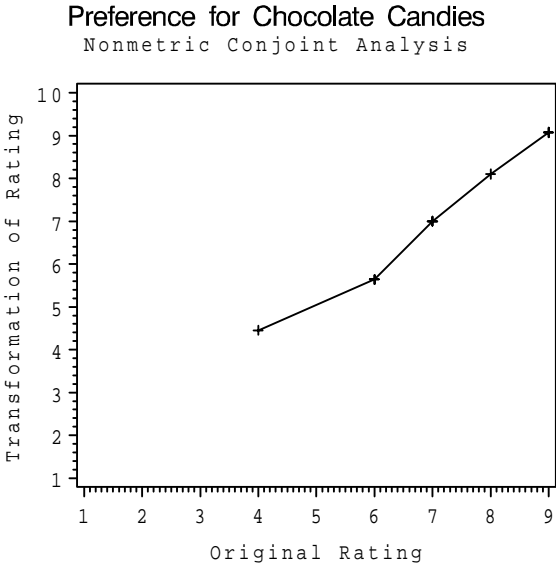
The standard errors are not adjusted for the fact that the dependent variable was transformed and so are generally liberal (too small).

The R^2 increases from 0.95 for the metric case to 0.96985 for the nonmetric case. The importances and part-worth utilities are slightly different from the metric analysis, but the overall pattern of results is the same.

The GPLOT procedure is used to plot the transformation of the ratings.

```
proc sort; by rating; run;

proc gplot;
  title h=1.5 'Preference for Chocolate Candies';
  title2 h=1 'Nonmetric Conjoint Analysis';
  plot trating * rating = 1 / frame haxis=axis2 vaxis=axis1;
  symbol1 v=plus i=join;
  axis1 order=(1 to 10)
    label=(angle=90 'Transformation of Rating');
  axis2 order=(1 to 9) label=('Original Rating');
run; quit;
```



In this case, the transformation is nearly linear. In practice, the R^2 may increase much more than it did in this example, and the transformation may be markedly nonlinear.

Frozen Diet Entrées Example (Basic)

This example uses PROC TRANSREG to perform a conjoint analysis to study preferences for frozen diet entrées. The entrées have four attributes: three with three levels and one with two levels. The attributes are shown in the table.

Factor	Levels		
Main Ingredient	Chicken	Beef	Turkey
Fat Claim Per Serving	8 Grams	5 Grams	2 Grams
Price	\$2.59	\$2.29	\$1.99
Calories	350	250	

Choosing the Number of Stimuli

Ideally, for this design, we would like the number of *runs* in the experimental design to be divisible by 2 (because of the two-level factor), 3 (because of the three-level factors), $2 \times 3 = 6$ (to have equal numbers of products in each two-level and three-level factor combinations), and $3 \times 3 = 9$ (to have equal numbers of products in each pair of three-level factor combinations). If we fit a main-effects model, we need at least $1 + 3 \times (3 - 1) + (2 - 1) = 8$ runs. We can avoid doing this math ourselves and instead use the %MktRuns autocall macro to help us choose the number of products. See page 597 for macro documentation and information on installing and using SAS autocall macros. To use this macro, you specify the number of levels for each of the factors. For this example, specify three 3's and one 2.

```
title 'Frozen Diet Entrees';
```

```
%mktruns( 3 3 3 2 )
```

Frozen Diet Entrees

Design Summary

Number of Levels	Frequency
2	1
3	3

Frozen Diet Entrees

```
Saturated      = 8
Full Factorial = 54
```

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
18 *	0	
36 *	0	
12	3	9
24	3	9
30	3	9
9	4	2 6
27	4	2 6
15	7	2 6 9
21	7	2 6 9
33	7	2 6 9

* - 100% Efficient Design can be made with the MktEx Macro.

n	Design	Reference
18	2 ** 1 3 ** 7	Orthogonal Array
36	2 ** 16 3 ** 4	Orthogonal Array
36	2 ** 11 3 ** 12	Orthogonal Array
36	2 ** 10 3 ** 8 6 ** 1	Orthogonal Array
36	2 ** 9 3 ** 4 6 ** 2	Orthogonal Array
36	2 ** 4 3 ** 13	Orthogonal Array
36	2 ** 3 3 ** 9 6 ** 1	Orthogonal Array
36	2 ** 2 3 ** 12 6 ** 1	Orthogonal Array
36	2 ** 2 3 ** 5 6 ** 2	Orthogonal Array
36	2 ** 1 3 ** 8 6 ** 2	Orthogonal Array
36	2 ** 1 3 ** 3 6 ** 3	Orthogonal Array

The output tells us that we need at least eight products, shown by the “Saturated = 8”. The sizes 18 and 36 would be optimal. Twelve is a good size but three times it cannot be divided by $9 = 3 \times 3$. The “three times” comes from the $3(3 - 1)/2 = 3$ pairs of three-level factors. Similarly, the size 9 has four violations because it cannot be divided once by 2 and three times by $6 = 2 \times 3$ (once for each three-level factor and two-level factor pair). We could use a size smaller than 18 and not have equal frequencies everywhere, but 18 is a manageable number so we will use 18.

When an orthogonal and balanced design is available from the %MktEx macro, the %MktRuns macro tells us about it. For example, the macro tells us that our design, which can be designated $2^1 3^3$, is available in 18 runs, and can be constructed from a design with 1 two-level factor (2 ** 1 or 2^1) and 7 three-level factors (3 ** 7 or 3^7). Both the %MktRuns and %MktEx macros accept this ‘ $n ** m$ ’ exponential syntax as input, which means m factors each at n levels. Hence, 2 3 ** 7 or 2 ** 1 3 ** 7 or 2 3 3 3 3 3 3 3 are all equivalent level-list specifications for the experimental design $2^1 3^7$, which has 1 two-level factor and 7 three-level factors.

Generating the Design

We can use the `%MktEx` autocall macro to find a design. When you invoke the `%MktEx` macro for a simple problem, you only need to specify the numbers of levels and number of runs. The macro does the rest. The `%MktEx` macro can create designs in a number of ways. For this problem, it simply looks up an orthogonal design. Here is the `%MktEx` macro call for this example:

```
%mktex(3 3 3 2, n=18)
```

The first argument to the `%MktEx` macro is a list of factor levels, and the second is the number of runs (`n=18`). These are all the options that are needed for a simple problem such as this one. However, throughout this book, random number seeds are explicitly specified with the `seed=` option so that you can reproduce these results.[‡] Here is the code for creating our design with the random number seed and the actual factor names specified:

```
%mktex(3 3 3 2, n=18, seed=151)
%mktlab(vars=Ingredient Fat Price Calories)
```

The `%MktEx` macro always creates factors named `x1`, `x2`, and so on. The `%MktLab` autocall macro is used to change the names when you want to provide actual factor names. This example has four factors, `Ingredient`, `Fat`, and `Price`, each with three levels and `Calories` with two levels.

Here is the output:

```

                                Frozen Diet Entrees

                                Algorithm Search History

                                Current          Best
Design   Row,Col  D-Efficiency  D-Efficiency  Notes
-----
      1     Start    100.0000    100.0000  Tab
      1     End     100.0000

                                Frozen Diet Entrees

                                The OPTEX Procedure

                                Class Level Information

                                Class  Levels   -Values-

                                x1      3      1 2 3
                                x2      3      1 2 3
                                x3      3      1 2 3
                                x4      2      1 2

```

[‡]By specifying a random number seed, results should be reproducible within a SAS release for a particular operating system. However, due to machine differences, some results may not be exactly reproducible on other machines. For most orthogonal and balanced designs, the results should be reproducible. When computerized searches are done, it is likely that you will not get the same design as the one in the book, although you would expect the efficiency differences to be slight.

Frozen Diet Entrees

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.6667

We see that the macro had no trouble finding an optimal, 100% efficient experimental design through table look up. The value `Tab` in the `Notes` column of the algorithm search history tells us the macro was able to use table look up. See pages 597, 667, and the discrete choice examples starting on page 141 for more information on how the `%MktEx` macro works.

The `%MktEx` macro creates two output data sets with the experimental design, `Design` and `Randomized`. The `Design` data set is sorted and for a number of the tabled designs, often has a first row consisting entirely of ones. For these reasons, you should typically use the *randomized* design. In the randomized design, the profiles are presented in a random order and the levels have been randomly reassigned. Neither of these operations affects the design efficiency, balance, or orthogonality. When there are restrictions on the design (see for example page 551), the profiles are sorted into a random order, but the levels are *not* randomly reassigned. The randomized design is the default input to the `%MktLab` macro.

Evaluating and Preparing the Design

We will use the `FORMAT` procedure to create descriptive labels for the levels of the attributes. By default, the values of the factors are positive integers. For example for `ingredient`, we create a format `if` (for Ingredient Format) that assigns the descriptive value label “Chicken” for level 1, “Beef” for level 2, and “Turkey” for level 3. A permanent SAS data set is created with the formats assigned (although, as we will see in the next example, we could have done this previously in the `%MktLab` step). Finally, the design is printed.

```
proc format;
  value if 1='Chicken' 2='Beef' 3='Turkey';
  value ff 1='8 Grams' 2='5 Grams' 3='2 Grams';
  value pf 1='$2.59' 2='$2.29' 3='$1.99';
  value cf 1='350' 2='250';
run;

data sasuser.dietdes;
  set final;
  format ingredient if. fat ff. price pf. calories cf.;
run;

proc print; run;
```

Here is the design.

 Frozen Diet Entrees

Obs	Ingredient	Fat	Price	Calories
1	Turkey	5 Grams	\$1.99	350
2	Turkey	8 Grams	\$2.29	350
3	Chicken	8 Grams	\$1.99	350
4	Turkey	2 Grams	\$2.59	250
5	Beef	8 Grams	\$2.59	350
6	Beef	2 Grams	\$1.99	350
7	Beef	5 Grams	\$2.29	350
8	Beef	5 Grams	\$2.29	250
9	Chicken	2 Grams	\$2.29	350
10	Beef	8 Grams	\$2.59	250
11	Turkey	8 Grams	\$2.29	250
12	Chicken	5 Grams	\$2.59	350
13	Chicken	5 Grams	\$2.59	250
14	Chicken	2 Grams	\$2.29	250
15	Turkey	5 Grams	\$1.99	250
16	Turkey	2 Grams	\$2.59	350
17	Beef	2 Grams	\$1.99	250
18	Chicken	8 Grams	\$1.99	250

Even when you know the design is 100% D -efficient, orthogonal, and balanced, it is good to run basic checks on your designs. You can use the %MktEval autocall macro to display information about the design.

```
%mkteval;
```

The macro first prints a matrix of canonical correlations between the factors. We hope to see an identity matrix (a matrix of ones on the diagonal and zeros everywhere else), which would mean that all of the factors are uncorrelated. Next, the macro prints all one-way frequencies for all attributes, all two-way frequencies, and all n -way frequencies (in this case four-way frequencies). We hope to see equal or at least nearly equal one-way and two-way frequencies, and we want to see that each combination occurs only once.

Frozen Diet Entrees

Canonical Correlations Between the Factors

There are 0 Canonical Correlations Greater Than 0.316

	Ingredient	Fat	Price	Calories
Ingredient	1	0	0	0
Fat	0	1	0	0
Price	0	0	1	0
Calories	0	0	0	1

Frozen Diet Entrees
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

	Frequencies
Ingredient	6 6 6
Fat	6 6 6
Price	6 6 6
Calories	9 9
Ingredient Fat	2 2 2 2 2 2 2 2 2
Ingredient Price	2 2 2 2 2 2 2 2 2
Ingredient Calories	3 3 3 3 3 3
Fat Price	2 2 2 2 2 2 2 2 2
Fat Calories	3 3 3 3 3 3
Price Calories	3 3 3 3 3 3
N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

A *canonical correlation* is the maximum correlation between linear combinations of the coded factors (see page 70). All zeros off the diagonal show that this design is orthogonal for main effects. If any off-diagonal canonical correlations had been greater than 0.316 ($r^2 > 0.1$), the macro would have listed them in a separate table. The last title line tells you that none of them were this large. For nonorthogonal designs and designs with interactions, the canonical-correlation matrix is not a substitute for looking at the variance matrix (with `examine=v`) in the `%MktEx` macro. The `%MktEx` macro just provides a quick and more-compact picture of the correlations between the factors. The variance matrix is sensitive to the actual model specified and the coding. The canonical-correlation matrix just tells you if there is some correlation between the main effects. In this case, there are no correlations.

The equal one-way frequencies show you that this design is balanced. The equal two-way frequencies show you that this design is orthogonal. Equal one-way and two-way frequencies together show you that this design is 100% *D*-efficient. The *n*-way frequencies, all equal to one, show you that there are no duplicate profiles. This is a perfect design for a main effects model. However, there are other 100% efficient designs for this problem with duplicate observations. In the last part of the output, the *n*-Way frequencies may contain some 2's for those designs. You can specify `options=nodups` in the `%MktEx` macro to ensure that there are no duplicate profiles.

The `%MktEval` macro produces a very compact summary of the design, hence some information, for example the levels to which the frequencies correspond, is not shown. You can use the `print=freqs` option in the `%MktEval` macro to get a less compact and more detailed display.

Printing the Stimuli and Data Collection

Next, we generate the stimuli. The `data _null_` step uses the `file` statement to set the print destination to the printed output destination. The design data set is read with the `set` statement. A `put` statement prints the attributes along with some constant text and the combination number. The `put` statement option `+3` skips 3 spaces, `@50` starts printing in column 50, `+(-1)` skips one space *backwards* getting rid of the blank that would by default appear after the stimulus number, and `/` skips to a new line. Text enclosed in quotes is literally copied to the output. For our attribute variables, the formatted

values are printed. The variable `_n_` is the number of the current pass through the DATA step, which in this case is the stimulus number. The `if` statement causes six descriptions to be printed on a page.

```

title;
data _null_;
  file print;
  set sasuser.dietdes;
  put ///
    +3 ingredient 'Entree' @50 '(' _n_ +(-1) ')' /
    +3 'With ' fat 'of Fat and ' calories 'Calories' /
    +3 'Now for Only ' Price +(-1) '.'///;
  if mod(_n_, 6) = 0 then put _page_;
run;

```

Turkey Entree With 5 Grams of Fat and 350 Calories Now for Only \$1.99.	(1)
Turkey Entree With 8 Grams of Fat and 350 Calories Now for Only \$2.29.	(2)
Chicken Entree With 8 Grams of Fat and 350 Calories Now for Only \$1.99.	(3)
Turkey Entree With 2 Grams of Fat and 250 Calories Now for Only \$2.59.	(4)
Beef Entree With 8 Grams of Fat and 350 Calories Now for Only \$2.59.	(5)
Beef Entree With 2 Grams of Fat and 350 Calories Now for Only \$1.99.	(6)
Beef Entree With 5 Grams of Fat and 350 Calories Now for Only \$2.29.	(7)
Beef Entree With 5 Grams of Fat and 250 Calories Now for Only \$2.29.	(8)
Chicken Entree With 2 Grams of Fat and 350 Calories Now for Only \$2.29.	(9)

Beef Entree	(10)
With 8 Grams of Fat and 250 Calories	
Now for Only \$2.59.	
Turkey Entree	(11)
With 8 Grams of Fat and 250 Calories	
Now for Only \$2.29.	
Chicken Entree	(12)
With 5 Grams of Fat and 350 Calories	
Now for Only \$2.59.	
Chicken Entree	(13)
With 5 Grams of Fat and 250 Calories	
Now for Only \$2.59.	
Chicken Entree	(14)
With 2 Grams of Fat and 250 Calories	
Now for Only \$2.29.	
Turkey Entree	(15)
With 5 Grams of Fat and 250 Calories	
Now for Only \$1.99.	
Turkey Entree	(16)
With 2 Grams of Fat and 350 Calories	
Now for Only \$2.59.	
Beef Entree	(17)
With 2 Grams of Fat and 250 Calories	
Now for Only \$1.99.	
Chicken Entree	(18)
With 8 Grams of Fat and 250 Calories	
Now for Only \$1.99.	

Next, we print the stimuli, produce the cards, and ask a subject to sort the cards from most preferred to least preferred. The combination numbers (most preferred to least preferred) are entered as data. For example, this subject's most preferred combination is 17, which is the "Beef Entree, With 2 Grams of Fat and 250 Calories, Now for Only \$1.99", and her least preferred combination is 18, "Chicken Entree, With 8 Grams of Fat and 250 Calories, Now for Only \$1.99".

Data Processing

The data are transposed, going from one observation and 18 variables to 18 observations and one variable named `Combo`. The next `DATA` step creates the variable `Rank`: 1 for the first and most preferred combination, ..., and 18 for the last and least preferred combination. The data are sorted by combination number and merged with the design.

```

title 'Frozen Diet Entrees';

data results;
  input combo1-combo18;
  datalines;
17 6 8 7 10 5 4 16 15 1 11 2 9 14 12 13 3 18
;
proc transpose out=results(rename=(col1=combo)); run;
data results; set results; Rank = _n_; drop _name_; run;
proc sort; by combo; run;
data results(drop=combo);
  merge sasuser.dietdes results;
  run;
proc print; run;

```

Frozen Diet Entrees					
Obs	Ingredient	Fat	Price	Calories	Rank
1	Turkey	5 Grams	\$1.99	350	10
2	Turkey	8 Grams	\$2.29	350	12
3	Chicken	8 Grams	\$1.99	350	17
4	Turkey	2 Grams	\$2.59	250	7
5	Beef	8 Grams	\$2.59	350	6
6	Beef	2 Grams	\$1.99	350	2
7	Beef	5 Grams	\$2.29	350	4
8	Beef	5 Grams	\$2.29	250	3
9	Chicken	2 Grams	\$2.29	350	13
10	Beef	8 Grams	\$2.59	250	5
11	Turkey	8 Grams	\$2.29	250	11
12	Chicken	5 Grams	\$2.59	350	15
13	Chicken	5 Grams	\$2.59	250	16
14	Chicken	2 Grams	\$2.29	250	14
15	Turkey	5 Grams	\$1.99	250	9
16	Turkey	2 Grams	\$2.59	350	8
17	Beef	2 Grams	\$1.99	250	1
18	Chicken	8 Grams	\$1.99	250	18

Recall that the seventeenth combination was most preferred, and it has a rank of 1. The eighteenth combination was least preferred and it has a rank of 18.

Nonmetric Conjoint Analysis

PROC TRANSREG is used to perform the nonmetric conjoint analysis of the ranks.

```
ods exclude notes anova liberalanova conservanova
      mvanova liberalmvanova conservmvanova;
proc transreg utilities order=formatted separators=', ';
  model monotone(rank / reflect) =
    class(Ingredient Fat Price Calories / zero=sum);
  output out=utils p ireplace;
run;
```

The `utilities` option prints the part-worth utilities and importance table. The `order=formatted` option sorts the levels of the attributes by the formatted values. By default, levels are sorted by their internal unformatted values (in this case the integers 1, 2, 3). The `model` statement names the variable `Rank` as the dependent variable and specifies a `monotone` transformation for the nonmetric conjoint analysis. The `reflect` transformation option is specified with rank data. With rank data, small values mean high preference and large values mean low preference. The `reflect` transformation option reflects the ranks around their mean $-(\text{rank} - \text{mean rank}) + \text{mean rank}$ so that in the results, large part-worth utilities will mean high preference. With ranks ranging from 1 to 18, `reflect` transforms 1 to 18, 2 to 17, ..., r to $(19 - r)$, ..., and 18 to 1. (Note that the mean rank is the midpoint, in this case $(18+1)/2 = 9.5$, and $-(r - \bar{r}) + \bar{r} = 2\bar{r} - r = 2(\max(r) + \min(r))/2 - r = 19 - r$.) The `class` specification names the attributes and scales the part-worth utilities to sum to zero within each attribute.

The `output` statement creates the `out=` data set, which contains the original variables, transformed variables, and indicator variables. The predicted utilities for all combinations are written to this data set by the `p` option (for predicted values). The `ireplace` option specifies that the transformed independent variables replace the original independent variables, since both are the same.

Here are the results of the conjoint analysis. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG.

Frozen Diet Entrees

The TRANSREG Procedure

Dependent Variable Monotone(Rank)

Class Level Information

Class	Levels	Values
Ingredient	3	Beef Chicken Turkey
Fat	3	2 Grams 5 Grams 8 Grams
Price	3	\$1.99 \$2.29 \$2.59
Calories	2	250 350

Number of Observations Read 18
 Number of Observations Used 18

TRANSREG Univariate Algorithm Iteration History for Monotone(Rank)

Iteration Number	Average Change	Maximum Change	R-Square	Criterion Change	Note
1	0.07276	0.10014	0.99174		
2	0.00704	0.01074	0.99977	0.00802	
3	0.00468	0.00710	0.99990	0.00013	
4	0.00311	0.00470	0.99995	0.00006	
5	0.00207	0.00312	0.99998	0.00003	
6	0.00138	0.00208	0.99999	0.00001	
7	0.00092	0.00138	1.00000	0.00001	
8	0.00061	0.00092	1.00000	0.00000	
9	0.00041	0.00061	1.00000	0.00000	
10	0.00027	0.00041	1.00000	0.00000	
11	0.00018	0.00027	1.00000	0.00000	
12	0.00012	0.00018	1.00000	0.00000	
13	0.00008	0.00012	1.00000	0.00000	
14	0.00005	0.00008	1.00000	0.00000	
15	0.00004	0.00005	1.00000	0.00000	
16	0.00002	0.00004	1.00000	0.00000	
17	0.00002	0.00002	1.00000	0.00000	
18	0.00001	0.00002	1.00000	0.00000	
19	0.00001	0.00001	1.00000	0.00000	Converged

Algorithm converged.

Frozen Diet Entrees

The TRANSREG Procedure

The TRANSREG Procedure Hypothesis Tests for Monotone(Rank)

Root MSE	0.00007166	R-Square	1.0000
Dependent Mean	9.50000	Adj R-Sq	1.0000
Coeff Var	0.00075429		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	9.5000	0.00002	
Ingredient, Beef	6.0281	0.00002	74.999
Ingredient, Chicken	-6.0281	0.00002	
Ingredient, Turkey	-0.0000	0.00002	
Fat, 2 Grams	2.0094	0.00002	25.000
Fat, 5 Grams	0.0000	0.00002	
Fat, 8 Grams	-2.0094	0.00002	
Price, \$1.99	0.0000	0.00002	0.000
Price, \$2.29	0.0000	0.00002	
Price, \$2.59	-0.0000	0.00002	
Calories, 250	0.0001	0.00002	0.001
Calories, 350	-0.0001	0.00002	

The standard errors are not adjusted for the fact that the dependent variable was transformed and so are generally liberal (too small).

We see in the conjoint output that main ingredient was the most important attribute at almost 75% and that beef was preferred over turkey, which was preferred over chicken. We also see that fat content was the second most important attribute at 25% and lower fat is preferred over higher fat. Price and calories only account for essentially none of the preference.

Next, the products in the `out=` data set are sorted by their predicted utility and the combinations are printed along with their rank, transformed and reflected rank, and predicted values (predicted utility). The variable `Rank` is the original rank variable; `TRank` contains the transformation of rank, in this case the reflection and monotonic transformation; and `PRank` contains the predicted utilities or predicted values. The first letter of the variable name comes from the first letter of “Transformation” and “Predicted”.

```
proc sort; by descending prank; run;

proc print label;
  var ingredient fat price calories rank trank prank;
  label trank = 'Reflected Rank'
        prank = 'Utilities';
run;
```

Frozen Diet Entrees							
Obs	Ingredient	Fat	Price	Calories	Rank	Reflected Rank	Utilities
1	Beef	2 Grams	\$1.99	250	1	17.5375	17.5375
2	Beef	2 Grams	\$1.99	350	2	17.5373	17.5373
3	Beef	5 Grams	\$2.29	250	3	15.5282	15.5281
4	Beef	5 Grams	\$2.29	350	4	15.5279	15.5280
5	Beef	8 Grams	\$2.59	250	5	13.5188	13.5188
6	Beef	8 Grams	\$2.59	350	6	13.5186	13.5186
7	Turkey	2 Grams	\$2.59	250	7	11.5095	11.5094
8	Turkey	2 Grams	\$2.59	350	8	11.5092	11.5093
9	Turkey	5 Grams	\$1.99	250	9	9.5001	9.5001
10	Turkey	5 Grams	\$1.99	350	10	9.4999	9.4999
11	Turkey	8 Grams	\$2.29	250	11	7.4908	7.4907
12	Turkey	8 Grams	\$2.29	350	12	7.4905	7.4906
13	Chicken	2 Grams	\$2.29	250	14	5.4813	5.4814
14	Chicken	2 Grams	\$2.29	350	13	5.4813	5.4812
15	Chicken	5 Grams	\$2.59	250	16	3.4719	3.4720
16	Chicken	5 Grams	\$2.59	350	15	3.4719	3.4719
17	Chicken	8 Grams	\$1.99	250	18	1.4626	1.4627
18	Chicken	8 Grams	\$1.99	350	17	1.4626	1.4625

It is interesting to see that the sorted combinations support the information in the utilities table. The combinations are perfectly sorted on beef, turkey, and chicken. Furthermore, within ties in the main ingredient, the products are sorted by fat content.

Frozen Diet Entrées Example (Advanced)

This example is an advanced version of the previous example. It illustrates conjoint analysis with more than one subject. It has six parts.

- The %MktEx macro is used to generate an experimental design.
- Holdout observations are generated.
- The descriptions of the products are printed for data collection.
- The data are collected, entered, and processed.
- The metric conjoint analysis is performed.
- Results are summarized across subjects.

Creating a Design with the %MktEx Macro

The first thing you need to do in a conjoint study is decide on the product attributes and levels. Then you create the experimental design. We will use the same experimental design as we used in the previous example. The attributes and levels are shown in the table.

Factor	Levels		
Main Ingredient	Chicken	Beef	Turkey
Fat Claim Per Serving	8 Grams	5 Grams	2 Grams
Price	\$2.59	\$2.29	\$1.99
Calories	350	250	

We will create our designs in the same way as we did in the previous example, starting on page 498. Only the random number seed has changed. Like before, we use the %MktEval macro to check the one-way and two-way frequencies and to ensure that each combination only appears once. See page 597 for macro documentation and information on installing and using SAS autocall macros.

```

title 'Frozen Diet Entrees';

proc format;
  value if 1='Chicken' 2='Beef' 3='Turkey';
  value ff 1='8 Grams' 2='5 Grams' 3='2 Grams';
  value pf 1='$2.59' 2='$2.29' 3='$1.99';
  value cf 1='350' 2='250';
run;

%mktx(3 3 3 2, n=18, seed=205)
%mktxlab(vars=Ingredient Fat Price Calories)
%mktxeval;

```

Frozen Diet Entrees

Algorithm Search History

Design	Row,Col	Current	Best	Notes
		D-Efficiency	D-Efficiency	
1	Start	100.0000	100.0000	Tab
1	End	100.0000		

Frozen Diet Entrees

The OPTEX Procedure

Class Level Information

Class	Levels	-Values-
x1	3	1 2 3
x2	3	1 2 3
x3	3	1 2 3
x4	2	1 2

Frozen Diet Entrees

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	100.0000	100.0000	100.0000	0.6667

Frozen Diet Entrees

Canonical Correlations Between the Factors

There are 0 Canonical Correlations Greater Than 0.316

	Ingredient	Fat	Price	Calories
Ingredient	1	0	0	0
Fat	0	1	0	0
Price	0	0	1	0
Calories	0	0	0	1

Frozen Diet Entrees
Summary of Frequencies
There are 0 Canonical Correlations Greater Than 0.316

	Frequencies
Ingredient	6 6 6
Fat	6 6 6
Price	6 6 6
Calories	9 9
Ingredient Fat	2 2 2 2 2 2 2 2 2
Ingredient Price	2 2 2 2 2 2 2 2 2
Ingredient Calories	3 3 3 3 3 3
Fat Price	2 2 2 2 2 2 2 2 2
Fat Calories	3 3 3 3 3 3
Price Calories	3 3 3 3 3 3
N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

This design is 100% efficient, perfectly balanced and orthogonal, and each product occurs exactly once.

Designing Holdouts

The next steps add *holdout* observations to the design and display the results. Holdouts are ranked by the subjects but are analyzed with zero weight to exclude them from contributing to the utility computations. The correlation between the ranks for holdouts and their predicted utilities provide an indication of the validity of the results of the study.

The first %MktEx step recreates the formats and the design (just so you can see all of the code for a design with holdouts in one step). The next %MktEx step adds four holdouts to the randomized design created from the previous step. The specification options=nodups (no duplicates) ensures that the holdouts do not match products already in the design. The first %MktEval step evaluates just the original design, excluding the holdouts. The second %MktEval step evaluates the entire design. Both %MktEval steps ensure that the variable w, which flags the active and holdout observations, is excluded and not treated as a factor. The %MktLab step gives the factors informative names and assigns formats. Unlike the previous examples, this time we directly assign the formats in the %MktLab macro using the statements= option, specifying a complete format statement.

```

title 'Frozen Diet Entrees';

proc format;
  value if 1='Chicken' 2='Beef' 3='Turkey';
  value ff 1='8 Grams' 2='5 Grams' 3='2 Grams';
  value pf 1='$2.59' 2='$2.29' 3='$1.99';
  value cf 1='350' 2='250';
run;

%mktx(3 3 3 2, n=18, seed=205)

%mktx(3 3 3 2, n=22, init=randomized, holdouts=4, options=nodups, seed=368)

```

```

proc print data=randomized; run;

%mkteval(data=randomized(where=(w=1)), factors=x:);
%mkteval(data=randomized(drop=w));

%mktlab(data=randomized, out=sasuser.dietdes,
        vars=Ingredient Fat Price Calories,
        statements=format Ingredient if. fat ff. price pf. calories cf.)

proc print; run;

```

Here is the last part of the output from the first %MktEx step, which shows that the macro found a 100% efficient design.

Frozen Diet Entrees				
The OPTEX Procedure				
Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error

1	100.0000	100.0000	100.0000	0.6667

Next, is some of the output from the %MktEx step that finds the holdouts. Notice that the macro immediately enters the design refinement step.

Frozen Diet Entrees				
Design Refinement History				
Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes

0	Initial	98.0764		Ini
1	Start	98.0764		Pre,Mut,Ann
1	22 1	98.2421	98.2421	Conforms
1	End	98.2421		
2	Start	98.2421		Pre,Mut,Ann
2	2 1	98.2421	98.2421	Conforms
2	End	98.2421		
3	Start	98.2421		Pre,Mut,Ann
3	2 1	98.2421	98.2421	Conforms
3	End	98.2421		

4	Start	98.2421		Pre,Mut,Ann
4	2 1	98.2421	98.2421	Conforms
4	End	98.2421		
5	Start	98.2421		Pre,Mut,Ann
5	2 1	98.2421	98.2421	Conforms
5	End	98.2421		

NOTE: Stopping since it appears that no improvement is possible.

Next, the raw design is printed. Observations with w equal to 1 comprise the original design. The observations with a missing w are the holdouts.

Frozen Diet Entrees

Obs	x1	x2	x3	x4	w
1	2	3	1	2	.
2	2	2	1	2	1
3	3	3	3	1	1
4	3	3	3	2	1
5	3	1	1	1	1
6	1	3	1	1	1
7	1	3	1	2	1
8	1	1	2	1	1
9	2	2	1	1	1
10	3	2	2	2	1
11	2	2	3	1	.
12	2	3	2	2	1
13	3	1	1	2	1
14	2	1	3	2	1
15	3	2	1	1	.
16	1	2	3	1	1
17	1	1	2	2	1
18	1	2	2	2	.
19	2	3	2	1	1
20	3	2	2	1	1
21	1	2	3	2	1
22	2	1	3	1	1

Here is the evaluation of the original design.

Frozen Diet Entrees
 Canonical Correlations Between the Factors
 There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4
x1	1	0	0	0
x2	0	1	0	0
x3	0	0	1	0
x4	0	0	0	1

Frozen Diet Entrees
 Summary of Frequencies
 There are 0 Canonical Correlations Greater Than 0.316

Frequencies

x1	6 6 6
x2	6 6 6
x3	6 6 6
x4	9 9
x1 x2	2 2 2 2 2 2 2 2 2
x1 x3	2 2 2 2 2 2 2 2 2
x1 x4	3 3 3 3 3 3
x2 x3	2 2 2 2 2 2 2 2 2
x2 x4	3 3 3 3 3 3
x3 x4	3 3 3 3 3 3
N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

Here is an evaluation of the design with the holdouts.

Frozen Diet Entrees
 Canonical Correlations Between the Factors
 There are 0 Canonical Correlations Greater Than 0.316

	x1	x2	x3	x4
x1	1	0.09	0.17	0.11
x2	0.09	1	0.09	0.11
x3	0.17	0.09	1	0.11
x4	0.11	0.11	0.11	1

Frozen Diet Entrees
 Summary of Frequencies
 There are 0 Canonical Correlations Greater Than 0.316
 * - Indicates Unequal Frequencies

Frequencies

```

*   x1      7 8 7
*   x2      6 9 7
*   x3      8 7 7
    x4      11 11
*   x1 x2   2 3 2 2 3 3 2 3 2
*   x1 x3   2 3 2 3 2 3 3 2 2
*   x1 x4   3 4 4 4 4 3
*   x2 x3   2 2 2 3 3 3 3 2 2
*   x2 x4   3 3 5 4 3 4
*   x3 x4   4 4 3 4 4 3
*   N-Way  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
            1 1 1
    
```

Here is the design, printed with descriptive factor names and formats.

Frozen Diet Entrees

Obs	Ingredient	Fat	Price	Calories	w
1	Beef	2 Grams	\$2.59	250	.
2	Beef	5 Grams	\$2.59	250	1
3	Turkey	2 Grams	\$1.99	350	1
4	Turkey	2 Grams	\$1.99	250	1
5	Turkey	8 Grams	\$2.59	350	1
6	Chicken	2 Grams	\$2.59	350	1
7	Chicken	2 Grams	\$2.59	250	1
8	Chicken	8 Grams	\$2.29	350	1
9	Beef	5 Grams	\$2.59	350	1
10	Turkey	5 Grams	\$2.29	250	1
11	Beef	5 Grams	\$1.99	350	.
12	Beef	2 Grams	\$2.29	250	1
13	Turkey	8 Grams	\$2.59	250	1
14	Beef	8 Grams	\$1.99	250	1
15	Turkey	5 Grams	\$2.59	350	.

16	Chicken	5 Grams	\$1.99	350	1
17	Chicken	8 Grams	\$2.29	250	1
18	Chicken	5 Grams	\$2.29	250	.
19	Beef	2 Grams	\$2.29	350	1
20	Turkey	5 Grams	\$2.29	350	1
21	Chicken	5 Grams	\$1.99	250	1
22	Beef	8 Grams	\$1.99	350	1

Print the Stimuli

Once the design is generated, the *stimuli* (descriptions of the combinations) must be generated for data collection. They are printed using the exact same step as we used on page 502.

```

title;
data _null_;
  file print;
  set sasuser.dietdes;
  put ///
    +3 ingredient 'Entree' @50 '(' _n_ +(-1) ')' /
    +3 'With ' fat 'of Fat and ' calories 'Calories' /
    +3 'Now for Only ' Price +(-1) '.'///;
  if mod(_n_, 6) = 0 then put _page_;
run;

```

In the interest of space, only the first three are shown.

Beef Entree	(1)
With 2 Grams of Fat and 250 Calories	
Now for Only \$2.59.	
Beef Entree	(2)
With 5 Grams of Fat and 250 Calories	
Now for Only \$2.59.	
Turkey Entree	(3)
With 2 Grams of Fat and 350 Calories	
Now for Only \$1.99.	

Data Collection, Entry, and Preprocessing

The next step in the conjoint analysis study is data collection and entry. Each subject was asked to take the 22 cards and rank them from the most preferred combination to the least preferred combination. The combination numbers are entered as data. The data follow the `datalines` statement in the next DATA step. For the first subject, 4 was most preferred, 3 was second most preferred, ..., and 5 was the least preferred combination. The DATA step validates the data entry and converts the input to ranks.

```

title 'Frozen Diet Entrees';

%let m = 22; /* number of combinations */

* Read the input data and convert to ranks;
data ranks(drop=i k c1-c&m);
  input c1-c&m;
  array c[&m];
  array r[&m];
  do i = 1 to &m;
    k = c[i];
    if 1 le k le &m then do;
      if r[k] ne . then
        put 'ERROR: For subject ' _n_ +(-1) ', combination ' k
          'is given more than once.';
      r[k] = i; /* Convert to ranks. */
    end;
  else put 'ERROR: For subject ' _n_ +(-1) ', combination ' k
    'is invalid.';
  end;
do i = 1 to &m;
  if r[i] = . then
    put 'ERROR: For subject ' _n_ +(-1) ', combination ' i
      'is not given.';
  end;
  name = 'Subj' || put(_n_, z2.);
  datalines;
4 3 7 21 12 10 6 19 1 16 18 11 20 14 17 15 2 22 9 8 13 5
4 12 3 1 19 7 10 6 11 21 16 2 18 20 15 9 14 22 13 17 5 8
4 3 7 12 19 21 1 6 10 18 16 11 20 15 2 14 9 17 22 8 13 5
4 12 1 10 21 14 18 3 7 2 17 13 19 11 22 20 16 15 6 9 5 8
4 21 14 11 16 3 12 22 19 18 10 17 8 20 7 1 6 2 9 13 15 5
4 21 16 12 3 14 11 22 18 19 7 10 1 17 8 6 2 20 9 13 15 5
12 4 19 1 3 7 6 21 18 11 16 2 10 20 9 15 14 17 22 8 13 5
4 21 3 16 14 11 12 22 18 10 19 20 17 8 7 6 1 2 13 15 9 5
4 21 3 16 11 14 22 12 18 10 20 19 17 8 7 6 1 13 15 2 9 5
4 3 14 11 21 12 16 22 19 10 18 20 17 1 7 8 2 13 9 6 15 5
15 22 17 21 6 11 13 19 4 12 3 18 9 7 1 10 8 20 14 16 5 2
12 4 3 7 21 19 1 18 11 6 16 2 14 10 17 22 20 9 15 8 13 5
;

```

The macro variable `&m` is set to 22, the number of combinations. This is done to make it easier to modify the code for future use with different sized studies. For each subject, the numbers of the 22 products are read into the variables `c1` through `c22`. The do loop, `do i = 1 to &m`, loops over each of the products. Consider the first product: `k` is set to `c[i]`, which is `c[1]`, which is 4 since the fourth product was ranked first by the first subject. The first data integrity check, `if 1 le k le &m then do` ensures that the number is in the valid range, 1 to 22. Otherwise an error is printed. Since the number is valid, `r[k]` is checked to see if it is missing. If it is not missing, another error is printed. The array `r` consists of 22 variables `r1` through `r22`. These variables start out each pass through the DATA step as missing and end up as the ranks. If `r[k] eq .`, then the *k*th combination has not had

a rank assigned yet so everything is fine. If `r[k] ne .`, the same number appears twice in a subject's data so there is something wrong with the data entry. The statement `r[k] = i` assigns the ranks. For subject 1 and the first product, `k = c[i] = c[1] = 4` so the rank of the fourth product is set to 1 (`r[k] = r[4] = i = 1`). For subject 1 and the second product, `k = c[i] = c[2] = 3` so the rank of the third product is set to 2 (`r[k] = r[3] = i = 2`). For subject 1 and the last product, `k = c[i] = c[22] = 5` so the rank of the fifth product is set to 22 (`r[k] = r[5] = i = 22`). At the end of the `do i = 1 to &m` loop, each of the 22 variables in `r1-r22` should have been set to exactly one rank. If any of these variables are missing, then one or more product numbers did not appear in the data, so this is flagged as an error. The statement `name = 'Subj' || put(_n_, z2.)` creates a subject ID of the form `Subj01, Subj02, ..., Subj12`.

Say there was a mistake in data entry for the first subject—say product 17 had been entered as 7 instead of 17. We would get the following error messages.

```
ERROR: For subject 1, combination 7 is given more than once.
ERROR: For subject 1, combination 17 is not given.
```

If for the first subject, the 17 had been entered as 117 instead of 17, we would get the following error messages.

```
ERROR: For subject 1, combination 117 is invalid.
ERROR: For subject 1, combination 17 is not given.
```

The next step transposes the data set from one row per subject to one row per product. The `id name` statement on PROC TRANSPOSE names the rank variables `Subj01` through `Subj12`. Later, we will need to sort by these names. That is why we used leading zeros and names like `Subj01` instead of `Subj1`. Next, the input data set is merged with the design.

```
proc transpose data=ranks out=ranks2;
  id name;
run;
data both;
  merge sasuser.dietdes ranks2;
  drop _name_;
run;
proc print label;
  title2 'Data and Design Together';
run;
```

Frozen Diet Entrees									
Data and Design Together									
Obs	Ingredient	Fat	Price	Calories	w	Subj01	Subj02	Subj03	Subj04
1	Beef	2 Grams	\$2.59	250	.	9	4	7	3
2	Beef	5 Grams	\$2.59	250	1	17	12	15	10
3	Turkey	2 Grams	\$1.99	350	1	2	3	2	8
4	Turkey	2 Grams	\$1.99	250	1	1	1	1	1
5	Turkey	8 Grams	\$2.59	350	1	22	21	22	21
6	Chicken	2 Grams	\$2.59	350	1	7	8	8	19

7	Chicken	2 Grams	\$2.59	250	1	3	6	3	9
8	Chicken	8 Grams	\$2.29	350	1	20	22	20	22
9	Beef	5 Grams	\$2.59	350	1	19	16	17	20
10	Turkey	5 Grams	\$2.29	250	1	6	7	9	4
11	Beef	5 Grams	\$1.99	350	.	12	9	12	14
12	Beef	2 Grams	\$2.29	250	1	5	2	4	2
13	Turkey	8 Grams	\$2.59	250	1	21	19	21	12
14	Beef	8 Grams	\$1.99	250	1	14	17	16	6
15	Turkey	5 Grams	\$2.59	350	.	16	15	14	18
16	Chicken	5 Grams	\$1.99	350	1	10	11	11	17
17	Chicken	8 Grams	\$2.29	250	1	15	20	18	11
18	Chicken	5 Grams	\$2.29	250	.	11	13	10	7
19	Beef	2 Grams	\$2.29	350	1	8	5	5	13
20	Turkey	5 Grams	\$2.29	350	1	13	14	13	16
21	Chicken	5 Grams	\$1.99	250	1	4	10	6	5
22	Beef	8 Grams	\$1.99	350	1	18	18	19	15
Obs	Subj05	Subj06	Subj07	Subj08	Subj09	Subj10	Subj11	Subj12	
1	16	13	4	17	17	14	15	7	
2	18	17	12	18	20	17	22	12	
3	6	5	5	3	3	2	11	3	
4	1	1	2	1	1	1	9	2	
5	22	22	22	22	22	22	21	22	
6	17	16	7	16	16	20	5	10	
7	15	11	6	15	15	15	14	4	
8	13	15	20	14	14	16	17	20	
9	19	19	15	21	21	19	13	18	
10	11	12	13	10	10	10	16	14	
11	4	7	10	6	5	4	6	9	
12	7	4	1	7	8	6	10	1	
13	20	20	21	19	18	18	7	21	
14	3	6	17	5	6	3	19	13	
15	21	21	16	20	19	21	1	19	
16	5	3	11	4	4	7	20	11	
17	12	14	18	13	13	13	3	15	
18	10	9	9	9	9	11	12	8	
19	9	10	3	11	12	9	8	6	
20	14	18	14	12	11	12	18	17	
21	2	2	8	2	2	5	4	5	
22	8	8	19	8	7	8	2	16	

One more data set manipulation is sometimes necessary—the addition of *simulation* observations. Simulation observations are not rated by the subjects and do not contribute to the analysis. They are scored as passive observations. Simulations are *what-if* combinations. They are combinations that are entered to get a prediction of what their utility would have been if they had been rated. In this example, all combinations are added as simulations. The %MktEx macro is called to make a full-factorial design. The n= specification accepts expressions, so n=3*3*3*2 and n=54 are equivalent. The data all step reads in the design and data followed by the simulation observations. The flag variable f

indicates when the simulation observations are being processed. Simulation observations are given a weight of 0 to exclude them from the analysis and to distinguish them from the holdouts. Notice that the dependent variable has missing values for the simulations and nonmissing values for the holdouts and active observations.

```
proc format;
  value wf 1 = 'Active'
         . = 'Holdout'
         0 = 'Simulation';

run;

%mktx(3 3 3 2, n=3*3*3*2)
%mktlab(data=design, vars=Ingredient Fat Price Calories)

data all;
  set both final(in=f);
  if f then w = 0;
  format w wf.;
run;

proc print data=all(Ob=25 drop=subj04-subj12) label;
  title2 'Some of the Final Data Set';
run;
```

Here the data for the first three subjects and the first 25 rows of the data set.

Frozen Diet Entrees								
Some of the Final Data Set								
Obs	Ingredient	Fat	Price	Calories	w	Subj01	Subj02	Subj03
1	Beef	2 Grams	\$2.59	250	Holdout	9	4	7
2	Beef	5 Grams	\$2.59	250	Active	17	12	15
3	Turkey	2 Grams	\$1.99	350	Active	2	3	2
4	Turkey	2 Grams	\$1.99	250	Active	1	1	1
5	Turkey	8 Grams	\$2.59	350	Active	22	21	22
6	Chicken	2 Grams	\$2.59	350	Active	7	8	8
7	Chicken	2 Grams	\$2.59	250	Active	3	6	3
8	Chicken	8 Grams	\$2.29	350	Active	20	22	20
9	Beef	5 Grams	\$2.59	350	Active	19	16	17
10	Turkey	5 Grams	\$2.29	250	Active	6	7	9
11	Beef	5 Grams	\$1.99	350	Holdout	12	9	12
12	Beef	2 Grams	\$2.29	250	Active	5	2	4
13	Turkey	8 Grams	\$2.59	250	Active	21	19	21
14	Beef	8 Grams	\$1.99	250	Active	14	17	16
15	Turkey	5 Grams	\$2.59	350	Holdout	16	15	14
16	Chicken	5 Grams	\$1.99	350	Active	10	11	11
17	Chicken	8 Grams	\$2.29	250	Active	15	20	18
18	Chicken	5 Grams	\$2.29	250	Holdout	11	13	10

19	Beef	2 Grams	\$2.29	350	Active	8	5	5
20	Turkey	5 Grams	\$2.29	350	Active	13	14	13
21	Chicken	5 Grams	\$1.99	250	Active	4	10	6
22	Beef	8 Grams	\$1.99	350	Active	18	18	19
23	Chicken	8 Grams	\$2.59	350	Simulation	.	.	.
24	Chicken	8 Grams	\$2.59	350	Simulation	.	.	.
25	Chicken	8 Grams	\$2.59	350	Simulation	.	.	.

Metric Conjoint Analysis

In this part of this example, the conjoint analysis is performed with PROC TRANSREG.

```
ods exclude notes mvanova anova;
proc transreg data=all utilities short separators=', '
  method=morals outtest=utils;
  title2 'Conjoint Analysis';
  model identity(subj: / reflect) =
    class(Ingredient Fat Price Calories / zero=sum);
  weight w;
  output p ireplace out=results coefficients;
run;
```

The `proc`, `model`, and `output` statements are typical for a conjoint analysis of rank-order data with more than one subject. (In this analysis, we perform a metric conjoint analysis. It is more typical to perform nonmetric conjoint analysis of rank-order data. However, it is not absolutely required.) The `proc` statement specifies `method=morals`, which fits the conjoint analysis model separately for each subject. The `proc` statement also requests an `outtest=` data set, which contains the ANOVA and part-worth utilities tables from the printed output. In the `model` statement, the dependent variable list `subj:` specifies all variables in the `DATA=` data set that begin with the prefix `subj` (in this case `subj01-subj12`). The `weight` variable designates the active (`weight = 1`), holdout (`weight = .`), and simulation (`weight = 0`) observations. Only the active observations are used to compute the part-worth utilities. However, predicted utilities are computed for all observations, including active, holdouts, and simulations, using those part-worths. The `output` statement creates an `out=` data set beginning with all results for the first subject, followed by all subject two results, and so on.

Here are the results. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG. There is one set of output for each subject. Conjoint analysis fits individual-level models.

Frozen Diet Entrees Conjoint Analysis

The TRANSREG Procedure

Class Level Information

Class	Levels	Values
Ingredient	3	Chicken Beef Turkey

Fat	3	8 Grams	5 Grams	2 Grams
Price	3	\$2.59	\$2.29	\$1.99
Calories	2	350	250	
Number of Observations Read				76
Number of Observations Used				18
Sum of Weights Read				18
Sum of Weights Used				18

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj01)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj01)

Root MSE	1.81046	R-Square	0.9618
Dependent Mean	11.38889	Adj R-Sq	0.9351
Coeff Var	15.89675		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.3889	0.42673	
Ingredient, Chicken	1.5556	0.60349	13.095
Ingredient, Beef	-2.1111	0.60349	
Ingredient, Turkey	0.5556	0.60349	
Fat, 8 Grams	-6.9444	0.60349	50.000
Fat, 5 Grams	-0.1111	0.60349	
Fat, 2 Grams	7.0556	0.60349	
Price, \$2.59	-3.4444	0.60349	23.810
Price, \$2.29	0.2222	0.60349	
Price, \$1.99	3.2222	0.60349	
Calories, 350	-1.8333	0.42673	13.095
Calories, 250	1.8333	0.42673	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj02)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj02)

Root MSE	1.30809	R-Square	0.9788
Dependent Mean	11.77778	Adj R-Sq	0.9640
Coeff Var	11.10646		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.7778	0.30832	
Ingredient, Chicken	-1.0556	0.43603	8.451
Ingredient, Beef	0.1111	0.43603	
Ingredient, Turkey	0.9444	0.43603	
Fat, 8 Grams	-7.7222	0.43603	64.789
Fat, 5 Grams	0.1111	0.43603	
Fat, 2 Grams	7.6111	0.43603	
Price, \$2.59	-1.8889	0.43603	15.493
Price, \$2.29	0.1111	0.43603	
Price, \$1.99	1.7778	0.43603	
Calories, 350	-1.3333	0.30832	11.268
Calories, 250	1.3333	0.30832	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj03)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj03)

Root MSE	1.15470	R-Square	0.9844
Dependent Mean	11.66667	Adj R-Sq	0.9735
Coeff Var	9.89743		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.6667	0.27217	
Ingredient, Chicken	0.6667	0.38490	6.667
Ingredient, Beef	-1.0000	0.38490	
Ingredient, Turkey	0.3333	0.38490	
Fat, 8 Grams	-7.6667	0.38490	62.000
Fat, 5 Grams	-0.1667	0.38490	
Fat, 2 Grams	7.8333	0.38490	
Price, \$2.59	-2.6667	0.38490	20.667
Price, \$2.29	0.1667	0.38490	
Price, \$1.99	2.5000	0.38490	
Calories, 350	-1.3333	0.27217	10.667
Calories, 250	1.3333	0.27217	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj04)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj04)

Root MSE	1.05935	R-Square	0.9849
Dependent Mean	11.72222	Adj R-Sq	0.9743
Coeff Var	9.03711		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.7222	0.24969	
Ingredient, Chicken	-2.1111	0.35312	13.490
Ingredient, Beef	0.7222	0.35312	
Ingredient, Turkey	1.3889	0.35312	
Fat, 8 Grams	-2.7778	0.35312	22.484
Fat, 5 Grams	-0.2778	0.35312	
Fat, 2 Grams	3.0556	0.35312	
Price, \$2.59	-3.4444	0.35312	25.054
Price, \$2.29	0.3889	0.35312	
Price, \$1.99	3.0556	0.35312	
Calories, 350	-5.0556	0.24969	38.972
Calories, 250	5.0556	0.24969	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj05)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj05)

Root MSE	1.02198	R-Square	0.9854
Dependent Mean	11.22222	Adj R-Sq	0.9752
Coeff Var	9.10676		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.2222	0.24088	
Ingredient, Chicken	0.5556	0.34066	7.407
Ingredient, Beef	0.5556	0.34066	
Ingredient, Turkey	-1.1111	0.34066	
Fat, 8 Grams	-1.7778	0.34066	17.037
Fat, 5 Grams	-0.2778	0.34066	
Fat, 2 Grams	2.0556	0.34066	
Price, \$2.59	-7.2778	0.34066	63.704
Price, \$2.29	0.2222	0.34066	
Price, \$1.99	7.0556	0.34066	
Calories, 350	-1.3333	0.24088	11.852
Calories, 250	1.3333	0.24088	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj06)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj06)

Root MSE	1.67000	R-Square	0.9636
Dependent Mean	11.27778	Adj R-Sq	0.9381
Coeff Var	14.80785		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.2778	0.39362	
Ingredient, Chicken	1.1111	0.55667	11.015
Ingredient, Beef	0.6111	0.55667	
Ingredient, Turkey	-1.7222	0.55667	
Fat, 8 Grams	-2.8889	0.55667	24.622
Fat, 5 Grams	-0.5556	0.55667	
Fat, 2 Grams	3.4444	0.55667	
Price, \$2.59	-6.2222	0.55667	51.836
Price, \$2.29	-0.8889	0.55667	
Price, \$1.99	7.1111	0.55667	
Calories, 350	-1.6111	0.39362	12.527
Calories, 250	1.6111	0.39362	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj07)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj07)

Root MSE	1.06979	R-Square	0.9857
Dependent Mean	11.88889	Adj R-Sq	0.9756
Coeff Var	8.99821		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.8889	0.25215	
Ingredient, Chicken	0.2222	0.35660	7.353
Ingredient, Beef	0.7222	0.35660	
Ingredient, Turkey	-0.9444	0.35660	
Fat, 8 Grams	-7.6111	0.35660	68.382
Fat, 5 Grams	-0.2778	0.35660	
Fat, 2 Grams	7.8889	0.35660	
Price, \$2.59	-1.9444	0.35660	15.441
Price, \$2.29	0.3889	0.35660	
Price, \$1.99	1.5556	0.35660	
Calories, 350	-1.0000	0.25215	8.824
Calories, 250	1.0000	0.25215	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj08)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj08)

Root MSE	0.79582	R-Square	0.9915
Dependent Mean	11.16667	Adj R-Sq	0.9855
Coeff Var	7.12677		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.1667	0.18758	
Ingredient, Chicken	0.5000	0.26527	4.412
Ingredient, Beef	-0.5000	0.26527	
Ingredient, Turkey	0.0000	0.26527	
Fat, 8 Grams	-2.3333	0.26527	20.588
Fat, 5 Grams	0.0000	0.26527	
Fat, 2 Grams	2.3333	0.26527	
Price, \$2.59	-7.3333	0.26527	64.706
Price, \$2.29	-0.0000	0.26527	
Price, \$1.99	7.3333	0.26527	
Calories, 350	-1.1667	0.18758	10.294
Calories, 250	1.1667	0.18758	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj09)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj09)

Root MSE	1.05935	R-Square	0.9850
Dependent Mean	11.27778	Adj R-Sq	0.9745
Coeff Var	9.39325		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.2778	0.24969	
Ingredient, Chicken	0.6111	0.35312	7.389
Ingredient, Beef	-1.0556	0.35312	
Ingredient, Turkey	0.4444	0.35312	
Fat, 8 Grams	-2.0556	0.35312	18.473
Fat, 5 Grams	-0.0556	0.35312	
Fat, 2 Grams	2.1111	0.35312	
Price, \$2.59	-7.3889	0.35312	65.764
Price, \$2.29	-0.0556	0.35312	
Price, \$1.99	7.4444	0.35312	
Calories, 350	-0.9444	0.24969	8.374
Calories, 250	0.9444	0.24969	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj10)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj10)

Root MSE	0.90062	R-Square	0.9889
Dependent Mean	11.27778	Adj R-Sq	0.9812
Coeff Var	7.98577		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.2778	0.21228	
Ingredient, Chicken	-1.3889	0.30021	9.722
Ingredient, Beef	0.9444	0.30021	
Ingredient, Turkey	0.4444	0.30021	
Fat, 8 Grams	-2.0556	0.30021	18.750
Fat, 5 Grams	-0.3889	0.30021	
Fat, 2 Grams	2.4444	0.30021	
Price, \$2.59	-7.2222	0.30021	59.028
Price, \$2.29	0.2778	0.30021	
Price, \$1.99	6.9444	0.30021	
Calories, 350	-1.5000	0.21228	12.500
Calories, 250	1.5000	0.21228	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj11)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj11)

Root MSE	7.42369	R-Square	0.2393
Dependent Mean	12.16667	Adj R-Sq	-0.2932
Coeff Var	61.01660		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	12.1667	1.74978	
Ingredient, Chicken	1.6667	2.47456	23.950
Ingredient, Beef	-0.1667	2.47456	
Ingredient, Turkey	-1.5000	2.47456	
Fat, 8 Grams	0.6667	2.47456	45.378
Fat, 5 Grams	-3.3333	2.47456	
Fat, 2 Grams	2.6667	2.47456	
Price, \$2.59	-1.5000	2.47456	21.429
Price, \$2.29	0.1667	2.47456	
Price, \$1.99	1.3333	2.47456	
Calories, 350	-0.6111	1.74978	9.244
Calories, 250	0.6111	1.74978	

Frozen Diet Entrees
Conjoint Analysis

The TRANSREG Procedure

Identity(Subj12)
Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Subj12)

Root MSE	1.49443	R-Square	0.9717
Dependent Mean	11.66667	Adj R-Sq	0.9519
Coeff Var	12.80944		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	11.6667	0.35224	
Ingredient, Chicken	0.8333	0.49814	8.974
Ingredient, Beef	0.6667	0.49814	
Ingredient, Turkey	-1.5000	0.49814	
Fat, 8 Grams	-6.1667	0.49814	51.923
Fat, 5 Grams	-1.1667	0.49814	
Fat, 2 Grams	7.3333	0.49814	
Price, \$2.59	-2.8333	0.49814	23.718
Price, \$2.29	-0.5000	0.49814	
Price, \$1.99	3.3333	0.49814	
Calories, 350	-2.0000	0.35224	15.385
Calories, 250	2.0000	0.35224	

Next, we will print some of the output data set to see the predicted utilities for the first two subjects.

```
proc print data=results(drop=_depend_ t_depend_ intercept &_trgind) label;
  title2 'Predicted Utility';
  where w ne 0 and _depvar_ le 'Identity(Subj02)' and not (_type_ =: 'M');
  by _depvar_;
  label p_depend_ = 'Predicted Utility';
run;
```

We print `_TYPE_`, `_NAME_`, and the weight variable, `w`; drop the original and transformed dependent variable, `_depend_` and `t_depend_`; print the predicted values (predicted utilities), `p_depend_`; drop the intercept and coded independent variables; and print the original class variables. Note that the macro variable `&_trgind` is automatically created by PROC TRANSREG and its value is a list of the names of the coded variables. The `where` statement is used to exclude the simulation observations and just show results for the first two subjects. Here are the predicted utilities for each of the rated products for the first two subjects.

Frozen Diet Entrees
Predicted Utility

----- Dependent Variable Transformation(Name)=Identity(Subj01) -----

Obs	_TYPE_	_NAME_	w	Predicted Utility	Ingredient	Fat	Price	Calories
1		ROW1	Holdout	14.7222	Beef	2 Grams	\$2.59	250
2	SCORE	ROW2	Active	7.5556	Beef	5 Grams	\$2.59	250
3	SCORE	ROW3	Active	20.3889	Turkey	2 Grams	\$1.99	350
4	SCORE	ROW4	Active	24.0556	Turkey	2 Grams	\$1.99	250
5	SCORE	ROW5	Active	-0.2778	Turkey	8 Grams	\$2.59	350
6	SCORE	ROW6	Active	14.7222	Chicken	2 Grams	\$2.59	350
7	SCORE	ROW7	Active	18.3889	Chicken	2 Grams	\$2.59	250
8	SCORE	ROW8	Active	4.3889	Chicken	8 Grams	\$2.29	350
9	SCORE	ROW9	Active	3.8889	Beef	5 Grams	\$2.59	350
10	SCORE	ROW10	Active	13.8889	Turkey	5 Grams	\$2.29	250
11		ROW11	Holdout	10.5556	Beef	5 Grams	\$1.99	350
12	SCORE	ROW12	Active	18.3889	Beef	2 Grams	\$2.29	250
13	SCORE	ROW13	Active	3.3889	Turkey	8 Grams	\$2.59	250
14	SCORE	ROW14	Active	7.3889	Beef	8 Grams	\$1.99	250
15		ROW15	Holdout	6.5556	Turkey	5 Grams	\$2.59	350
16	SCORE	ROW16	Active	14.2222	Chicken	5 Grams	\$1.99	350
17	SCORE	ROW17	Active	8.0556	Chicken	8 Grams	\$2.29	250
18		ROW18	Holdout	14.8889	Chicken	5 Grams	\$2.29	250
19	SCORE	ROW19	Active	14.7222	Beef	2 Grams	\$2.29	350
20	SCORE	ROW20	Active	10.2222	Turkey	5 Grams	\$2.29	350
21	SCORE	ROW21	Active	17.8889	Chicken	5 Grams	\$1.99	250
22	SCORE	ROW22	Active	3.7222	Beef	8 Grams	\$1.99	350

```

----- Dependent Variable Transformation(Name)=Identity(Subj02) -----

```

Obs	_TYPE_	_NAME_	w	Predicted Utility	Ingredient	Fat	Price	Calories
79		ROW1	Holdout	18.9444	Beef	2 Grams	\$2.59	250
80	SCORE	ROW2	Active	11.4444	Beef	5 Grams	\$2.59	250
81	SCORE	ROW3	Active	20.7778	Turkey	2 Grams	\$1.99	350
82	SCORE	ROW4	Active	23.4444	Turkey	2 Grams	\$1.99	250
83	SCORE	ROW5	Active	1.7778	Turkey	8 Grams	\$2.59	350
84	SCORE	ROW6	Active	15.1111	Chicken	2 Grams	\$2.59	350
85	SCORE	ROW7	Active	17.7778	Chicken	2 Grams	\$2.59	250
86	SCORE	ROW8	Active	1.7778	Chicken	8 Grams	\$2.29	350
87	SCORE	ROW9	Active	8.7778	Beef	5 Grams	\$2.59	350
88	SCORE	ROW10	Active	14.2778	Turkey	5 Grams	\$2.29	250
89		ROW11	Holdout	12.4444	Beef	5 Grams	\$1.99	350
90	SCORE	ROW12	Active	20.9444	Beef	2 Grams	\$2.29	250
91	SCORE	ROW13	Active	4.4444	Turkey	8 Grams	\$2.59	250
92	SCORE	ROW14	Active	7.2778	Beef	8 Grams	\$1.99	250
93		ROW15	Holdout	9.6111	Turkey	5 Grams	\$2.59	350
94	SCORE	ROW16	Active	11.2778	Chicken	5 Grams	\$1.99	350
95	SCORE	ROW17	Active	4.4444	Chicken	8 Grams	\$2.29	250
96		ROW18	Holdout	12.2778	Chicken	5 Grams	\$2.29	250
97	SCORE	ROW19	Active	18.2778	Beef	2 Grams	\$2.29	350
98	SCORE	ROW20	Active	11.6111	Turkey	5 Grams	\$2.29	350
99	SCORE	ROW21	Active	13.9444	Chicken	5 Grams	\$1.99	250
100	SCORE	ROW22	Active	4.6111	Beef	8 Grams	\$1.99	350

Analyzing Holdouts

The next steps display the correlations between the predicted utility for holdout observations and their actual ratings. These correlations provide a measure of the validity of the results, since the holdout observations have zero weight and do not contribute to any of the calculations. The Pearson correlations are the ordinary correlation coefficients, and the Kendall Tau's are rank-based measures of correlation. These correlations should always be large. Subjects whose correlations are small may be unreliable.

PROC CORR is used to produce the correlations. Since the output is not very compact, ODS is used to suppress the normal printed output (`ods listing close`), output the Pearson correlations to an output data set P (`PearsonCorr=p`), and output the Kendall correlations to an output data set K (`KendallCorr=k`). The listing is reopened for normal output (`ods listing`), the two tables are merged renaming the variables to identify the correlation type, the subject number is pulled out of the subject variable names, and the results are printed.

```
ods output KendallCorr=k PearsonCorr=p;
ods listing close;
proc corr nosimple noprob kendall pearson
  data=results(where=(w=));
  title2 'Holdout Validation Results';
  var p_depend_;
  with t_depend_;
  by notsorted _depvar_;
  run;
ods listing;

data both(keep=subject pearson kendall);
  length Subject 8;
  merge p(rename=(p_depend_=Pearson))
        k(rename=(p_depend_=Kendall));
  subject = input(substr(_depvar_, 14, 2), best2.);
  run;

proc print; run;
```

Here are the results.

Frozen Diet Entrees
Holdout Validation Results

Obs	Subject	Pearson	Kendall
1	1	0.93848	0.66667
2	2	0.94340	1.00000
3	3	0.99038	1.00000
4	4	0.97980	1.00000
5	5	0.98930	1.00000
6	6	0.98649	1.00000
7	7	0.99029	1.00000
8	8	0.99296	1.00000
9	9	0.99873	1.00000
10	10	0.99973	1.00000
11	11	-0.98184	-1.00000
12	12	0.92920	1.00000

Most of the correlations look great! However, the results from subject 11 look suspect. Subject 11's holdout correlations are negative. We can return to page 532 and look at the conjoint results. Subject 11 has an R^2 of 0.2393. In contrast, all of the other subjects have an R^2 over 0.95. Subject 11 almost certainly did not take the task seriously, so his or her results will be discarded.

```
data results2;
  set results;
  if not (index(_depvar_, '11'));
  run;
```

```

data utils2;
  set utils;
  if not (index(_depvar_, '11'));
run;

```

Simulations

The next steps display simulation observations. The most preferred combinations are printed for each subject.

```

proc sort data=results2(where=(w=0)) out=sims(drop=&_trgind);
  by _depvar_ descending p_depend_;
run;

data sims; /* Pull out first 10 for each subject. */
  set sims;
  by _depvar_;
  retain n 0;
  if first._depvar_ then n = 0;
  n = n + 1;
  if n le 10;
  drop w _depend_ t_depend_ n _name_ _type_ intercept;
run;

proc print data=sims label;
  by _depvar_ ;
  title2 'Simulations Sorted by Decreasing Predicted Utility';
  title3 'Just the Ten Most Preferred Combinations are Printed';
  label p_depend_ = 'Predicted Utility';
run;

```

Frozen Diet Entrees
 Simulations Sorted by Decreasing Predicted Utility
 Just the Ten Most Preferred Combinations are Printed

----- Dependent Variable Transformation(Name)=Identity(Subj01) -----

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
1	22.0556	Chicken	2 Grams	\$2.29	250
2	22.0556	Chicken	2 Grams	\$2.29	250
3	22.0556	Chicken	2 Grams	\$2.29	250
4	21.3889	Chicken	2 Grams	\$1.99	350
5	21.3889	Chicken	2 Grams	\$1.99	350
6	21.3889	Chicken	2 Grams	\$1.99	350
7	20.3889	Turkey	2 Grams	\$1.99	350
8	20.3889	Turkey	2 Grams	\$1.99	350
9	20.3889	Turkey	2 Grams	\$1.99	350
10	18.3889	Beef	2 Grams	\$2.29	250

----- Dependent Variable Transformation(Name)=Identity(Subj02) -----

Obs	Predicted				
	Utility	Ingredient	Fat	Price	Calories
11	20.9444	Beef	2 Grams	\$2.29	250
12	20.9444	Beef	2 Grams	\$2.29	250
13	20.9444	Beef	2 Grams	\$2.29	250
14	20.7778	Turkey	2 Grams	\$1.99	350
15	20.7778	Turkey	2 Grams	\$1.99	350
16	20.7778	Turkey	2 Grams	\$1.99	350
17	19.7778	Chicken	2 Grams	\$2.29	250
18	19.7778	Chicken	2 Grams	\$2.29	250
19	19.7778	Chicken	2 Grams	\$2.29	250
20	19.7778	Turkey	2 Grams	\$2.59	250

----- Dependent Variable Transformation(Name)=Identity(Subj03) -----

Obs	Predicted				
	Utility	Ingredient	Fat	Price	Calories
21	21.6667	Chicken	2 Grams	\$2.29	250
22	21.6667	Chicken	2 Grams	\$2.29	250
23	21.6667	Chicken	2 Grams	\$2.29	250
24	21.3333	Chicken	2 Grams	\$1.99	350
25	21.3333	Chicken	2 Grams	\$1.99	350
26	21.3333	Chicken	2 Grams	\$1.99	350
27	21.0000	Turkey	2 Grams	\$1.99	350
28	21.0000	Turkey	2 Grams	\$1.99	350
29	21.0000	Turkey	2 Grams	\$1.99	350
30	20.0000	Beef	2 Grams	\$2.29	250

----- Dependent Variable Transformation(Name)=Identity(Subj04) -----

Obs	Predicted				
	Utility	Ingredient	Fat	Price	Calories
31	20.9444	Beef	2 Grams	\$2.29	250
32	20.9444	Beef	2 Grams	\$2.29	250
33	20.9444	Beef	2 Grams	\$2.29	250
34	20.2778	Beef	5 Grams	\$1.99	250
35	20.2778	Beef	5 Grams	\$1.99	250
36	20.2778	Beef	5 Grams	\$1.99	250
37	18.4444	Turkey	8 Grams	\$1.99	250
38	18.4444	Turkey	8 Grams	\$1.99	250
39	18.4444	Turkey	8 Grams	\$1.99	250
40	18.1111	Chicken	2 Grams	\$2.29	250

----- Dependent Variable Transformation(Name)=Identity(Subj05) -----

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
41	19.8889	Beef	5 Grams	\$1.99	250
42	19.8889	Beef	5 Grams	\$1.99	250
43	19.8889	Beef	5 Grams	\$1.99	250
44	19.5556	Chicken	2 Grams	\$1.99	350
45	19.5556	Chicken	2 Grams	\$1.99	350
46	19.5556	Chicken	2 Grams	\$1.99	350
47	18.3889	Beef	8 Grams	\$1.99	250
48	18.3889	Beef	8 Grams	\$1.99	250
49	18.3889	Beef	8 Grams	\$1.99	250
50	17.8889	Turkey	2 Grams	\$1.99	350

----- Dependent Variable Transformation(Name)=Identity(Subj06) -----

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
51	21.3333	Chicken	2 Grams	\$1.99	350
52	21.3333	Chicken	2 Grams	\$1.99	350
53	21.3333	Chicken	2 Grams	\$1.99	350
54	20.0556	Beef	5 Grams	\$1.99	250
55	20.0556	Beef	5 Grams	\$1.99	250
56	20.0556	Beef	5 Grams	\$1.99	250
57	18.5000	Turkey	2 Grams	\$1.99	350
58	18.5000	Turkey	2 Grams	\$1.99	350
59	18.5000	Turkey	2 Grams	\$1.99	350
60	17.7222	Beef	8 Grams	\$1.99	250

----- Dependent Variable Transformation(Name)=Identity(Subj07) -----

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
61	21.8889	Beef	2 Grams	\$2.29	250
62	21.8889	Beef	2 Grams	\$2.29	250
63	21.8889	Beef	2 Grams	\$2.29	250
64	21.3889	Chicken	2 Grams	\$2.29	250
65	21.3889	Chicken	2 Grams	\$2.29	250
66	21.3889	Chicken	2 Grams	\$2.29	250
67	20.5556	Chicken	2 Grams	\$1.99	350
68	20.5556	Chicken	2 Grams	\$1.99	350
69	20.5556	Chicken	2 Grams	\$1.99	350
70	19.3889	Turkey	2 Grams	\$1.99	350

----- Dependent Variable Transformation(Name)=Identity(Subj08) -----

Obs	Predicted		Ingredient	Fat	Price	Calories
	Utility					
71	20.1667		Chicken	2 Grams	\$1.99	350
72	20.1667		Chicken	2 Grams	\$1.99	350
73	20.1667		Chicken	2 Grams	\$1.99	350
74	19.6667		Turkey	2 Grams	\$1.99	350
75	19.6667		Turkey	2 Grams	\$1.99	350
76	19.6667		Turkey	2 Grams	\$1.99	350
77	19.1667		Beef	5 Grams	\$1.99	250
78	19.1667		Beef	5 Grams	\$1.99	250
79	19.1667		Beef	5 Grams	\$1.99	250
80	17.8333		Chicken	5 Grams	\$1.99	350

----- Dependent Variable Transformation(Name)=Identity(Subj09) -----

Obs	Predicted		Ingredient	Fat	Price	Calories
	Utility					
81	20.5000		Chicken	2 Grams	\$1.99	350
82	20.5000		Chicken	2 Grams	\$1.99	350
83	20.5000		Chicken	2 Grams	\$1.99	350
84	20.3333		Turkey	2 Grams	\$1.99	350
85	20.3333		Turkey	2 Grams	\$1.99	350
86	20.3333		Turkey	2 Grams	\$1.99	350
87	18.5556		Beef	5 Grams	\$1.99	250
88	18.5556		Beef	5 Grams	\$1.99	250
89	18.5556		Beef	5 Grams	\$1.99	250
90	18.3333		Chicken	5 Grams	\$1.99	350

----- Dependent Variable Transformation(Name)=Identity(Subj10) -----

Obs	Predicted		Ingredient	Fat	Price	Calories
	Utility					
91	20.2778		Beef	5 Grams	\$1.99	250
92	20.2778		Beef	5 Grams	\$1.99	250
93	20.2778		Beef	5 Grams	\$1.99	250
94	19.6111		Turkey	2 Grams	\$1.99	350
95	19.6111		Turkey	2 Grams	\$1.99	350
96	19.6111		Turkey	2 Grams	\$1.99	350
97	18.6111		Beef	8 Grams	\$1.99	250
98	18.6111		Beef	8 Grams	\$1.99	250
99	18.6111		Beef	8 Grams	\$1.99	250
100	18.1111		Turkey	8 Grams	\$1.99	250

----- Dependent Variable Transformation(Name)=Identity(Subj12) -----

Obs	Predicted Utility	Ingredient	Fat	Price	Calories
101	21.3333	Chicken	2 Grams	\$2.29	250
102	21.3333	Chicken	2 Grams	\$2.29	250
103	21.3333	Chicken	2 Grams	\$2.29	250
104	21.1667	Chicken	2 Grams	\$1.99	350
105	21.1667	Chicken	2 Grams	\$1.99	350
106	21.1667	Chicken	2 Grams	\$1.99	350
107	21.1667	Beef	2 Grams	\$2.29	250
108	21.1667	Beef	2 Grams	\$2.29	250
109	21.1667	Beef	2 Grams	\$2.29	250
110	18.8333	Turkey	2 Grams	\$1.99	350

Summarizing Results Across Subjects

Conjoint analyses are performed on an individual basis, but usually the goal is to summarize the results across subjects. The `outtest=` data set contains all of the information in the printed output and can be manipulated to create additional reports including a list of the individual R^2 s and the average of the importance values across subjects. Here is a listing of the variables in the `outtest=` data set.

```
proc contents data=utils2 position;
  ods select position;
  title2 'Variables in the OUTTEST= Data Set';
run;
```

Frozen Diet Entrees Variables in the OUTTEST= Data Set

The CONTENTS Procedure

Variables in Creation Order

#	Variable	Type	Len	Label
1	_DEPVAR_	Char	42	Dependent Variable Transformation(Name)
2	_TYPE_	Char	8	
3	Title	Char	80	Title
4	Variable	Char	42	Variable
5	Coefficient	Num	8	Coefficient
6	Statistic	Char	24	Statistic

7	Value	Num	8	Value
8	NumDF	Num	8	Num DF
9	DenDF	Num	8	Den DF
10	SSq	Num	8	Sum of Squares
11	MeanSquare	Num	8	Mean Square
12	F	Num	8	F Value
13	NumericP	Num	8	Numeric (Approximate) p Value
14	P	Char	9	Formatted p Value
15	LowerLimit	Num	8	95% Lower Confidence Limit
16	UpperLimit	Num	8	95% Upper Confidence Limit
17	StdError	Num	8	Standard Error
18	Importance	Num	8	Importance (% Utility Range)
19	Label	Char	256	Label

The individual R^2 s are displayed by printing the Value variable for observations whose Statistic value is “R-Square”.

```
proc print data=utils2 label;
  title2 'R-Squares';
  id _depvar_;
  var value;
  format value 4.2;
  where statistic = 'R-Square';
  label value = 'R-Square' _depvar_ = 'Subject';
run;
```

Frozen Diet Entrees	
R-Squares	
Subject	R-Square
Identity(Subj01)	0.96
Identity(Subj02)	0.98
Identity(Subj03)	0.98
Identity(Subj04)	0.98
Identity(Subj05)	0.99
Identity(Subj06)	0.96
Identity(Subj07)	0.99
Identity(Subj08)	0.99
Identity(Subj09)	0.99
Identity(Subj10)	0.99
Identity(Subj12)	0.97

The next steps extract the importance values and create a table. The DATA step extracts the importance values and creates row and column labels. The PROC TRANSPOSE step creates a subjects by attributes matrix from a vector (of the number of subjects times the number of attribute values). PROC PRINT displays the importance values, and PROC MEANS displays the average importances.

```

data im;
  set utils2;
  if n(importance); /* Exclude all missing, including specials.*/
  _depvar_ = scan(_depvar_, 2); /* Discard transformation. */
  label _depvar_ = scan(label, 1, ','); /* Use up to comma for label. */
  keep importance _depvar_ label;
run;

proc transpose data=im out=im(drop=_name_ _label_);
  id label;
  by notsorted _depvar_;
  var importance;
  label _depvar_ = 'Subject';
run;

proc print label;
  title2 'Importances';
  format _numeric_ 2.;
  id _depvar_;
run;

proc means mean;
  title2 'Average Importances';
run;

```

Frozen Diet Entrees
Importances

Subject	Ingredient	Fat	Price	Calories
Subj01	13	50	24	13
Subj02	8	65	15	11
Subj03	7	62	21	11
Subj04	13	22	25	39
Subj05	7	17	64	12
Subj06	11	25	52	13
Subj07	7	68	15	9
Subj08	4	21	65	10
Subj09	7	18	66	8
Subj10	10	19	59	13
Subj12	9	52	24	15

Frozen Diet Entrees
Average Importances

The MEANS Procedure

Variable	Mean
Ingredient	8.9069044
Fat	38.0953010
Price	39.0198700
Calories	13.9779245

On the average, price is the most important attribute followed very closely by fat content. These two attributes on the average account for 77% of preference. Calories and main ingredient account for the remaining 23%. Note that everyone does not have the same pattern of importance values. However, it is a little hard to compare subjects just by looking at the numbers.

We can make a nicer display of importances with stars flagging the most important attributes for each product as follows. These steps replace each importance variable with its formatted value followed by zero stars for 0 - 30, one star for 30 - 45, two stars for 45 - 60, three stars for 60 - 75, and so on. The value returned by the `ceil` function is the number of characters that are extracted from the string '*****'.

```
data im2;
  set im;
  label c1 = 'Ingredient' c2 = 'Fat' c3 = 'Price' c4 = 'Calories';
  c1 = put(ingredient, 2.) || substr('*****', 1, ceil(ingredient / 15));
  c2 = put(fat, 2.) || substr('*****', 1, ceil(fat / 15));
  c3 = put(price, 2.) || substr('*****', 1, ceil(price / 15));
  c4 = put(calories, 2.) || substr('*****', 1, ceil(calories / 15));
run;

proc print label;
  title2 'Importances';
  var c1-c4;
  id _depvar_;
run;
```

Frozen Diet Entrees						
Importances						
Subject	Ingredient	Fat		Price		Calories
Subj01	13	50	**	24		13
Subj02	8	65	***	15		11
Subj03	7	62	***	21		11
Subj04	13	22		25		39 *
Subj05	7	17		64	***	12
Subj06	11	25		52	**	13
Subj07	7	68	***	15		9
Subj08	4	21		65	***	10
Subj09	7	18		66	***	8
Subj10	10	19		59	**	13
Subj12	9	52	**	24		15

Subject 4 is more concerned about calories. However, most individuals seem to fall into one of two groups, either primarily price conscious then fat conscious, or primarily fat conscious then price conscious.

Both the `out=` data set and the `outtest=` data set contain the part-worth utilities. In the `out=` data set, they are contained in the observations whose `_type_` value is 'M COEFFI'. The part-worth utilities are the multiple regression coefficients. The names of the variables that contain the part-worth utilities are stored in the macro variable `&_trgind`, which is automatically created by PROC TRANSREG.

```
proc print data=results2 label;
  title2 'Part-Worth Utilities';
  where _type_ = 'M COEFFI';
  id _name_;
  var &_trgind;
run;
```

Frozen Diet Entrees Part-Worth Utilities						
NAME	Ingredient, Chicken	Ingredient, Beef	Ingredient, Turkey	Fat, 8 Grams	Fat, 5 Grams	
Subj01	1.55556	-2.11111	0.55556	-6.94444	-0.11111	
Subj02	-1.05556	0.11111	0.94444	-7.72222	0.11111	
Subj03	0.66667	-1.00000	0.33333	-7.66667	-0.16667	
Subj04	-2.11111	0.72222	1.38889	-2.77778	-0.27778	
Subj05	0.55556	0.55556	-1.11111	-1.77778	-0.27778	
Subj06	1.11111	0.61111	-1.72222	-2.88889	-0.55556	
Subj07	0.22222	0.72222	-0.94444	-7.61111	-0.27778	
Subj08	0.50000	-0.50000	0.00000	-2.33333	0.00000	
Subj09	0.61111	-1.05556	0.44444	-2.05556	-0.05556	
Subj10	-1.38889	0.94444	0.44444	-2.05556	-0.38889	
Subj12	0.83333	0.66667	-1.50000	-6.16667	-1.16667	

NAME	Fat, 2 Grams	Price, \$2.59	Price, \$2.29	Price, \$1.99	Calories, 350	Calories, 250
Subj01	7.05556	-3.44444	0.22222	3.22222	-1.83333	1.83333
Subj02	7.61111	-1.88889	0.11111	1.77778	-1.33333	1.33333
Subj03	7.83333	-2.66667	0.16667	2.50000	-1.33333	1.33333
Subj04	3.05556	-3.44444	0.38889	3.05556	-5.05556	5.05556
Subj05	2.05556	-7.27778	0.22222	7.05556	-1.33333	1.33333
Subj06	3.44444	-6.22222	-0.88889	7.11111	-1.61111	1.61111
Subj07	7.88889	-1.94444	0.38889	1.55556	-1.00000	1.00000
Subj08	2.33333	-7.33333	-0.00000	7.33333	-1.16667	1.16667
Subj09	2.11111	-7.38889	-0.05556	7.44444	-0.94444	0.94444
Subj10	2.44444	-7.22222	0.27778	6.94444	-1.50000	1.50000
Subj12	7.33333	-2.83333	-0.50000	3.33333	-2.00000	2.00000

These part-worth utilities can be clustered, for example using PROC FASTCLUS.

```
proc fastclus data=results2 maxclusters=3 out=clusts;
  where _type_ = 'M COEFFI';
  id _name_;
  var &_trgind;
run;

proc sort; by cluster; run;

proc print label;
  title2 'Part-Worth Utilities, Clustered';
  by cluster;
  id _name_;
  var &_trgind;
run;
```

Frozen Diet Entrees
Part-Worth Utilities, Clustered

----- Cluster=1 -----

NAME	Ingredient, Chicken	Ingredient, Beef	Ingredient, Turkey	Fat, 8 Grams	Fat, 5 Grams
Subj05	0.55556	0.55556	-1.11111	-1.77778	-0.27778
Subj06	1.11111	0.61111	-1.72222	-2.88889	-0.55556
Subj08	0.50000	-0.50000	0.00000	-2.33333	0.00000
Subj09	0.61111	-1.05556	0.44444	-2.05556	-0.05556
Subj10	-1.38889	0.94444	0.44444	-2.05556	-0.38889

NAME	Fat, 2 Grams	Price, \$2.59	Price, \$2.29	Price, \$1.99	Calories, 350	Calories, 250
Subj05	2.05556	-7.27778	0.22222	7.05556	-1.33333	1.33333
Subj06	3.44444	-6.22222	-0.88889	7.11111	-1.61111	1.61111
Subj08	2.33333	-7.33333	-0.00000	7.33333	-1.16667	1.16667
Subj09	2.11111	-7.38889	-0.05556	7.44444	-0.94444	0.94444
Subj10	2.44444	-7.22222	0.27778	6.94444	-1.50000	1.50000

----- Cluster=2 -----

NAME	Ingredient, Chicken	Ingredient, Beef	Ingredient, Turkey	Fat, 8 Grams	Fat, 5 Grams
Subj01	1.55556	-2.11111	0.55556	-6.94444	-0.11111
Subj02	-1.05556	0.11111	0.94444	-7.72222	0.11111
Subj03	0.66667	-1.00000	0.33333	-7.66667	-0.16667
Subj07	0.22222	0.72222	-0.94444	-7.61111	-0.27778
Subj12	0.83333	0.66667	-1.50000	-6.16667	-1.16667

NAME	Fat, 2 Grams	Price, \$2.59	Price, \$2.29	Price, \$1.99	Calories, 350	Calories, 250
Subj01	7.05556	-3.44444	0.22222	3.22222	-1.83333	1.83333
Subj02	7.61111	-1.88889	0.11111	1.77778	-1.33333	1.33333
Subj03	7.83333	-2.66667	0.16667	2.50000	-1.33333	1.33333
Subj07	7.88889	-1.94444	0.38889	1.55556	-1.00000	1.00000
Subj12	7.33333	-2.83333	-0.50000	3.33333	-2.00000	2.00000

----- Cluster=3 -----

NAME	Ingredient, Chicken	Ingredient, Beef	Ingredient, Turkey	Fat, 8 Grams	Fat, 5 Grams
Subj04	-2.11111	0.72222	1.38889	-2.77778	-0.27778

NAME	Fat, 2 Grams	Price, \$2.59	Price, \$2.29	Price, \$1.99	Calories, 350	Calories, 250
Subj04	3.05556	-3.44444	0.38889	3.05556	-5.05556	5.05556

The clusters reflect what we saw looking at the importance information. Subject 4, who is the only subject that is primarily calorie conscious, is in a separate cluster from everyone else. Cluster 1 subjects 5, 6, 8, 9, and 10 are primarily price conscious. Cluster 2 subjects 1, 2, 3, 7, and 12 are primarily fat conscious.

Spaghetti Sauce

This example uses conjoint analysis in a study of spaghetti sauce preferences. The goal is to investigate the main effects for all of the attributes and the interaction of brand and price, and to simulate market share. Rating scale data are gathered from a group of subjects. The example has eight parts.

- An efficient experimental design is generated with the %MktEx macro.
- Descriptions of the spaghetti sauces are generated.
- Data are collected, entered, and processed.
- The metric conjoint analysis is performed with PROC TRANSREG.
- Market share is simulated with the maximum utility model.
- Market share is simulated with the Bradley-Terry-Luce and logit models.
- The simulators are compared.
- Change in market share is investigated.

Create an Efficient Experimental Design with the %MktEx Macro

In this example, subjects were asked to rate their interest in purchasing hypothetical spaghetti sauces. The table shows the attributes, the attribute levels, and the number of *df* associated with each effect.

Experimental Design		
Effects	Levels	<i>df</i>
Intercept		1
Brand	Pregu, Sundance, Tomato Garden	2
Meat Content	Vegetarian, Meat, Italian Sausage	2
Mushroom Content	Mushrooms, No Mention	1
Natural Ingredients	All Natural Ingredients, No Mention	1
Price	\$1.99, \$2.29, \$2.49, \$2.79, \$2.99	4
Brand × Price		8

The brand names “Pregu”, “Sundance”, and “Tomato Garden” are artificial. Usually, real brand names would be used—your client’s or company’s brand and the competitors’ brands. The absence of a feature (for example, no mushrooms) is not mentioned in the product description, hence the “No Mention” in the table.

In this design there are 19 model *df*. A design with more than 19 runs must be generated if there are to be error *df*. A popular heuristic is to limit the design size to at most 30 runs. In this example, 30 runs allow us to have two observations in each of the 15 brand by price cells. Note however that when subjects are required to make that many judgments, there is the risk that the quality of the data will be poor. Caution should be used when generating designs with this many runs. We can use the %MktRuns macro to evaluate this and other design sizes. See page 597 for macro documentation

and information on installing and using SAS autocall macros. We specify the number of levels of each factor as the argument.

```
title 'Spaghetti Sauces';

%mktruns( 3 3 2 2 5 )
```

Spaghetti Sauces			
Design Summary			
	Number of Levels	Frequency	
	2	2	
	3	2	
	5	1	
Saturated = 11			
Full Factorial = 180			
Some Reasonable Design Sizes	Violations	Cannot Be Divided By	
180 *	0		
60	1	9	
90	1	4	
120	1	9	
30	2	4 9	
150	2	4 9	
210	2	4 9	
36	5	5 10 15	
72	5	5 10 15	
108	5	5 10 15	

* - 100% Efficient Design can be made with the MktEx Macro.

We see that 30 is a reasonable size, although it cannot be divided by $9 = 3 \times 3$ and $4 = 2 \times 2$, so perfect orthogonality will not be possible. We would need a much larger size like 60 or 180 to do better. Note that this output states “Saturated=11” referring to a main-effects model. In this example, we are also interested in the brand by price interaction. We can run the %MktRuns macro again, this time specifying the interaction.

```
%mktruns( 3 3 2 2 5, interact=1*5 )
```

Spaghetti Sauces

Design Summary

Number of Levels	Frequency
2	2
3	2
5	1

Spaghetti Sauces

Saturated = 19
Full Factorial = 180

Some Reasonable Design Sizes	Violations	Cannot Be Divided By
180	0	
90	1	4
60	2	9 45
120	2	9 45
30	3	4 9 45
150	3	4 9 45
210	3	4 9 45
36	8	5 10 15 30 45
72	8	5 10 15 30 45
108	8	5 10 15 30 45

Now the output states “Saturated=19”, which includes the 8 *df* for the interaction. We see as before that 30 cannot be divided by 4 = 2 × 2. We also see that 30 cannot be divide by 45 = 3 × 15 so each level of meat content will not appear equally often in each brand/price cell. Since we would need a much larger size to do better, we will use 30 runs.

The next steps create and evaluate the design. First, formats for each of the factors are created using PROC FORMAT. The %MktEx macro is called to create the design. The factors **x1 = Brand** and **x2 = Meat** are designated as three-level factors, **x3 = Mushroom** and **x4 = Ingredients** as two-level factors, and **x5 = Price** as a five-level factor. The **interact=1*5** option specifies that the interaction between the first and fifth factors must be estimable (**x1 × x5** which is brand by price), **n=30** specifies the number of runs, and **seed=289** specifies the random number seed. The **where** macro provides restrictions that eliminate unrealistic combinations. Specifically, products at the cheapest price, \$1.99, with meat, and products with Italian Sausage with All Natural Ingredients are eliminated from consideration.

We impose restrictions with the %MktEx macro by writing a macro, with IML statements, that quantifies the badness of each run of the design. The variable **bad** is set to zero when everything is fine, and values larger than zero when the row of the design does not conform to the restrictions. Ideally, when there are multiple restrictions, as there are here, the variable **bad** should be set to the number of violations, so the macro can know when it is moving in the right direction as it changes the design. Our first

restriction (contribution to the badness value) is $(x_2 = 3 \ \& \ x_4 = 1)$ and our second is $(x_5 = 1 \ \& \ (x_2 = 2 \ | \ x_2 = 3))$, where $\&$ means **and** and $|$ means **or**.[§] The restrictions correspond to $(\text{Meat} = \text{'Italian Sausage'} \ \& \ \text{Ingredients} = \text{'All Natural'})$ and $(\text{Price} = 1.99 \ \& \ (\text{Meat} = \text{'Meat'} \ | \ \text{Meat} = \text{'Italian Sausage'}))$. Each of these Boolean or logical expressions evaluates to 1 when the expression is true and 0 when it is false. The sum of the two restrictions is: 0 - no problem, 1 - one restriction violation, or 2 - two restriction violations.

The `%MktLab` macro assigns actual descriptive factor names instead of the default `x1-x5` and formats for the levels. The default input to the `%MktLab` macro is the data set `Randomized`, which is the randomized design created by the `%MktEx` macro.

The default output from the `%MktLab` macro is a data set called `Final`. We instead use the `out=` option to store the results in a permanent SAS data set. The `%MktEx` macro is used to display the frequencies for each level, the two-way frequencies, and the number of times each product occurs in the design (five-way frequencies).

```

title 'Spaghetti Sauces';

proc format;
  value br 1='Pregu'      2='Sundance'  3='Tomato Garden';
  value me 1='Vegetarian' 2='Meat'      3='Italian Sausage';
  value mu 1='Mushrooms'  2='No Mention';
  value in 1='All Natural' 2='No Mention';
  value pr 1='1.99' 2='2.29' 3='2.49' 4='2.79' 5='2.99';
run;

%macro where;
  bad = (x2 = 3 & x4 = 1) + (x5 = 1 & (x2 = 2 | x2 = 3));
%mend;

%mktx(3 3 2 2 5, interact=1*5, n=30, seed=289, restrictions=where)
%mktlab(vars=Brand Meat Mushroom Ingredients Price,
  statements=format brand br. meat me. mushroom mu.
  ingredients in. price pr.,
  out=sasuser.spag);
%mkteval;

proc print data=sasuser.spag; run;

```

[§]In the restrictions macro, you must use the logical symbols `|` `&` `^` `~` `>` `<` `>=` `<=` `=` `^=` `-=` and *not* the logical words `OR` `AND` `NOT` `GT` `LT` `GE` `LE` `EQ` `NE`.

Here is some of the output from the %MktEx macro.

Spaghetti Sauces

Algorithm Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes

1	Start	92.6280		Can
1	2 1	92.6280	92.6280	Conforms
1	End	92.6280		
2	Start	78.9640		Tab,Unb
2	28 1	91.5726		Conforms
2	End	91.6084		
3	Start	78.9640		Tab,Unb
3	1 1	91.5434		Conforms
3	End	91.6084		
4	Start	77.5906		Tab,Ran
4	28 1	91.9486		Conforms
4	5 4	92.6280	92.6280	
4	End	92.6280		
.				
.				
.				
21	Start	74.7430		Ran,Mut,Ann
21	24 1	89.9706		Conforms
21	End	91.6084		

Spaghetti Sauces

Design Search History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes

0	Initial	92.6280	92.6280	Ini
1	Start	92.6280		Can
1	2 1	92.6280	92.6280	Conforms
1	End	92.6280		

Spaghetti Sauces

Design Refinement History

Design	Row,Col	Current D-Efficiency	Best D-Efficiency	Notes
0	Initial	92.6280	92.6280	Ini
1	Start	90.4842		Pre,Mut,Ann
1	2 1	91.2145		Conforms
1	End	91.6084		
.				
.				
.				
6	Start	91.1998		Pre,Mut,Ann
6	2 1	91.6084		Conforms
6	End	91.6084		

NOTE: Stopping since it appears that no improvement is possible.

Spaghetti Sauces

The OPTEX Procedure

Class Level Information

Class	Levels	-Values--
x1	3	1 2 3
x2	3	1 2 3
x3	2	1 2
x4	2	1 2
x5	5	1 2 3 4 5

Spaghetti Sauces

The OPTEX Procedure

Design Number	D-Efficiency	A-Efficiency	G-Efficiency	Average Prediction Standard Error
1	92.6280	82.6056	97.6092	0.7958

The *D*-Efficiency looks reasonable at 92.63. For this problem, the full-factorial design is small (180 runs), and the macro found the same *D*-efficiency several times. This suggests that we have probably

indeed found the optimal design for this situation. Here is the output from the %MktEval macro.

Spaghetti Sauces
 Canonical Correlations Between the Factors
 There are 2 Canonical Correlations Greater Than 0.316

	Brand	Meat	Mushroom	Ingredients	Price
Brand	1	0.21	0	0.17	0
Meat	0.21	1	0.08	0.42	0.52
Mushroom	0	0.08	1	0	0
Ingredients	0.17	0.42	0	1	0.17
Price	0	0.52	0	0.17	1

Spaghetti Sauces
 Canonical Correlations > 0.316 Between the Factors
 There are 2 Canonical Correlations Greater Than 0.316

		r	r Square
Meat	Price	0.52	0.27
Meat	Ingredients	0.42	0.17

Spaghetti Sauces
 Summary of Frequencies
 There are 2 Canonical Correlations Greater Than 0.316
 * - Indicates Unequal Frequencies

Frequencies

Brand	10 10 10
* Meat	15 9 6
Mushroom	15 15
* Ingredients	12 18
Price	6 6 6 6 6
* Brand Meat	4 3 3 5 4 1 6 2 2
Brand Mushroom	5 5 5 5 5 5
* Brand Ingredients	3 7 5 5 4 6
Brand Price	2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
* Meat Mushroom	7 8 5 4 3 3
* Meat Ingredients	7 8 5 4 0 6
* Meat Price	6 3 2 2 2 0 2 2 3 2 0 1 2 1 2
* Mushroom Ingredients	6 9 6 9
Mushroom Price	3 3 3 3 3 3 3 3 3 3
* Ingredients Price	3 3 2 2 2 3 3 4 4 4
N-Way	1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
	1 1 1 1 1 1 1 1 1 1 1

The meat and price factors are correlated, as are the meat and ingredients factors. This is not surprising since we excluded cells for these factor combinations and hence forced some correlations. The rest of the correlations are small.

The frequencies look good. The n -way frequencies at the end of this listing show that each product occurs only once. Each brand, price, and brand/price combination occurs equally often, as does each mushroom level. There are more vegetarian sauces (the first formatted level) than either of the meat sauces because of the restrictions that meat cannot occur at the lowest price and Italian sausage cannot be paired with all-natural ingredients. The design is shown next.

Spaghetti Sauces

Obs	Brand	Meat	Mushroom	Ingredients	Price
1	Pregu	Meat	No Mention	No Mention	2.79
2	Tomato Garden	Vegetarian	No Mention	No Mention	2.79
3	Pregu	Meat	Mushrooms	All Natural	2.29
4	Tomato Garden	Vegetarian	Mushrooms	All Natural	2.49
5	Sundance	Vegetarian	Mushrooms	No Mention	1.99
6	Pregu	Italian Sausage	No Mention	No Mention	2.49
7	Tomato Garden	Vegetarian	No Mention	No Mention	2.99
8	Tomato Garden	Italian Sausage	Mushrooms	No Mention	2.29
9	Pregu	Vegetarian	Mushrooms	No Mention	2.49
10	Pregu	Vegetarian	No Mention	No Mention	2.29
11	Sundance	Vegetarian	Mushrooms	No Mention	2.79
12	Tomato Garden	Vegetarian	Mushrooms	No Mention	1.99
13	Sundance	Meat	No Mention	No Mention	2.29
14	Sundance	Meat	Mushrooms	No Mention	2.99
15	Pregu	Italian Sausage	Mushrooms	No Mention	2.79
16	Tomato Garden	Italian Sausage	Mushrooms	No Mention	2.99
17	Sundance	Vegetarian	Mushrooms	All Natural	2.29
18	Pregu	Meat	Mushrooms	All Natural	2.99
19	Tomato Garden	Meat	No Mention	No Mention	2.49
20	Sundance	Meat	Mushrooms	All Natural	2.49
21	Pregu	Vegetarian	No Mention	All Natural	1.99
22	Sundance	Meat	No Mention	All Natural	2.79
23	Tomato Garden	Vegetarian	No Mention	All Natural	1.99
24	Sundance	Italian Sausage	No Mention	No Mention	2.49
25	Sundance	Vegetarian	No Mention	All Natural	1.99
26	Sundance	Vegetarian	No Mention	All Natural	2.99
27	Pregu	Italian Sausage	No Mention	No Mention	2.99
28	Tomato Garden	Vegetarian	No Mention	All Natural	2.29
29	Pregu	Vegetarian	Mushrooms	No Mention	1.99
30	Tomato Garden	Meat	Mushrooms	All Natural	2.79

Generating the Questionnaire

Next, preparations are made for data collection. A DATA step is used to print descriptions of each product combination. Here is an example:

```
Try Prego brand vegetarian spaghetti sauce, now with
mushrooms. A 26 ounce jar serves four adults for only
$1.99.
```

Remember that “No Mention” is not mentioned. The following step prints the questionnaires including a cover sheet.

```
options ls=80 ps=74 nonumber nodate;
title;

data _null_;
  set sasuser.spag;
  length lines $ 500 aline $ 60;
  file print linesleft=ll;

  * Format meat level, preserve 'Italian' capitalization;
  aline = lowercase(put(meat, me.));
  if aline =: 'ita' then substr(aline, 1, 1) = 'I';

  * Format meat differently for 'vegetarian';
  if meat > 1
  then lines = 'Try ' || trim(put(brand, br.)) ||
              ' brand spaghetti sauce with ' || aline;
  else lines = 'Try ' || trim(put(brand, br.)) ||
              ' brand ' || trim(aline) || ' spaghetti sauce ';

  * Add mushrooms, natural ingredients to text line;
  n = (put(ingredients, in.) =: 'All');
  m = (put(mushroom, mu.) =: 'Mus');

  if n or m then do;
    lines = trim(lines) || ', now with';

    if m then do;
      lines = trim(lines) || ' ' || lowercase(put(mushroom, mu.));
      if n then lines = trim(lines) || ' and';
    end;
    if n then lines = trim(lines) || ' ' ||
                    lowercase(put(ingredients, in.)) || ' ingredients';
  end;

  * Add price;
  lines = trim(lines) ||
        '. A 26 ounce jar serves four adults for only $' ||
        put(price, pr.) || '.';
```

```

* Print cover page, with subject number, instructions, and rating scale;
if _n_ = 1 then do;
  put ///// +41 'Subject: _____' ////
  +5 'Please rate your willingness to purchase the following' /
  +5 'products on a nine point scale.' ///
  +9 '1  Definitely Would Not Purchase This Product' ///
  +9 '2'  ///
  +9 '3  Probably Would Not Purchase This Product' ///
  +9 '4'  ///
  +9 '5  May or May Not Purchase This Product' ///
  +9 '6'  ///
  +9 '7  Probably Would Purchase This Product' ///
  +9 '8'  ///
  +9 '9  Definitely Would Purchase This Product' ////
  +5 'Please rate every product and be sure to rate' /
  +5 'each product only once.' /////
  +5 'Thank you for your participation!';
  put _page_;
end;
if ll < 8 then put _page_;
* Break up description, print on several lines;

start = 1;
do l = 1 to 10 until(aline = ' ');

  * Find a good place to split, blank or punctuation;
  stop = start + 60;
  do i = stop to start by -1 while(substr(lines, i, 1) ne ' '); end;
  do j = i to max(start, i - 8) by -1;
    if substr(lines, j, 1) in ('.' ',') then do; i = j; j = 0; end;
  end;

  stop = i; len = stop + 1 - start;
  aline = substr(lines, start, len);
  start = stop + 1;
  if l = 1 then put +5 _n_ 2. ') ' aline;
  else          put +9 aline;
end;

* Print rating scale;
put +9 'Definitely          Definitely ' /
  +9 'Would Not   1   2   3   4   5   6   7   8   9  Would      ' /
  +9 'Purchase          Purchase      ' //;
run;

options ls=80 ps=60 nonumber nodate;

```

In the interest of space, not all questions are printed.

Subject: _____

Please rate your willingness to purchase the following products on a nine point scale.

1 Definitely Would Not Purchase This Product

2

3 Probably Would Not Purchase This Product

4

5 May or May Not Purchase This Product

6

7 Probably Would Purchase This Product

8

9 Definitely Would Purchase This Product

Please rate every product and be sure to rate each product only once.

Thank you for your participation!

- .
.
.
- 30) Try Tomato Garden brand spaghetti sauce with meat, now with mushrooms and all natural ingredients. A 26 ounce jar serves four adults for only \$2.79.

Definitely												Definitely
Would Not	1	2	3	4	5	6	7	8	9			Would
Purchase												Purchase

Data Processing

The data are entered next. Some cases have ordinary '.' missing values. This code was used at data entry for no response. When there were multiple responses or the response was not clear, the special underscore missing value was used. The statement `missing _` specifies that underscore missing values are to be expected in the data. The `input` statement reads the subject number and the 30 ratings. A name like `Subj001`, `Subj002`, ..., `Subj030` is created from the subject number. If there are any missing data, all data for that subject are excluded by the `if nmiss(of rate:) = 0` statement.

```

title 'Spaghetti Sauces';

data rawdata;
  missing _;
  input subj @5 (rate1-rate30) (1.);
  name = compress('Sub' || put(subj, z3.));
  if nmiss(of rate:) = 0;
  datalines;
1 319591129691132168146121171191
2 749173216928911175549891841791
3 449491116819413186158171961791
.
.
.
14 1139812951994_9466149198915699
.
.
.
19 2214922399981121.1116161941991
.
.
.
;
```

Next, the data are transposed from one row per subject and 30 columns to one column per subject and 30 rows, one for each product rated. Then the data are merged with the experimental design.

```
proc transpose data=rawdata(drop=subj) out=temp(drop=_name_);
  id name;
  run;

data inputdata; merge sasuser.spag temp; run;
```

Metric Conjoint Analysis

Next, we use PROC TRANSREG to perform the conjoint analysis.

```
ods exclude notes mvanova anova;
proc transreg data=inputdata utilities short separators=' ' ', '
  lprefix=0 outtest=utils method=morals;
  title2 'Conjoint Analysis';
  model identity(sub:) =
    class(brand | price meat mushroom ingredients / zero=sum);
  output p ireplace out=results1 coefficients;
  run;
```

The `utilities` option requests conjoint analysis output, and the `short` option suppresses the iteration histories. The `lprefix=0` option specifies that zero variable name characters are to be used to construct the labels for the part-worths; the labels will simply consist of formatted values. The `outtest=` option creates an output SAS data set, `Utils`, that contains all of the statistical results. The `method=morals`, algorithm fits the conjoint analysis model separately for each subject. We specify `ods exclude notes mvanova anova` to exclude ANOVA information (which we usually want to ignore) and provide more parsimonious output.

The `model` statement names the ratings for each subject as dependent variables and the factors as independent variables. Since this is a metric conjoint analysis, `identity` is specified for the ratings. The `identity` transformation is the no-transformation option, which is used for variables that need to enter the model with no further manipulations. The factors are specified as `class` variables, and the `zero=sum` option is specified to constrain the parameter estimates to sum to zero within each effect. The `brand | price` specification asks for a simple `brand` effect, a simple `price` effect, and the `brand * price` interaction.

The `p` option in the `output` statement requests predicted values, the `ireplace` option suppresses the output of transformed independent variables, and the `coefficients` option requests that the part-worth utilities be output. These options control the contents of the `out=results` data set, which contains the ratings, predicted utilities for each product, indicator variables, and the part-worth utilities.

In the interest of space, only the results for the first subject are printed here. Recall that we used an `ods exclude` statement and we used PROC TEMPLATE on page 485 to customize the output from PROC TRANSREG.

Conjoint Analysis

The TRANSREG Procedure

Class Level Information

Class	Levels	Values
Brand	3	Pregu Sundance Tomato Garden
Price	5	1.99 2.29 2.49 2.79 2.99
Meat	3	Vegetarian Meat Italian Sausage
Mushroom	2	Mushrooms No Mention
Ingredients	2	All Natural No Mention
Number of Observations Read		30
Number of Observations Used		30

Conjoint Analysis

The TRANSREG Procedure

Identity(Sub001)
 Algorithm converged.

The TRANSREG Procedure Hypothesis Tests for Identity(Sub001)

Root MSE	2.09608	R-Square	0.8344
Dependent Mean	3.73333	Adj R-Sq	0.5635
Coeff Var	56.14499		

Part-Worth Utilities

Label	Utility	Standard Error	Importance (% Utility Range)
Intercept	3.0675	0.45364	
Pregu	2.0903	0.55937	28.924
Sundance	0.2973	0.55886	
Tomato Garden	-2.3876	0.55205	

1.99	-0.6836	0.91331	7.134
2.29	0.3815	0.77035	
2.49	0.4209	0.78975	
2.79	-0.5397	0.79677	
2.99	0.4209	0.78975	
Pregu, 1.99	0.7430	1.09161	15.639
Pregu, 2.29	0.9491	1.13055	
Pregu, 2.49	-0.7433	1.14528	
Pregu, 2.79	-1.0115	1.13157	
Pregu, 2.99	0.0626	1.13769	
Sundance, 1.99	0.0361	1.09135	
Sundance, 2.29	-1.2578	1.09310	
Sundance, 2.49	-0.1443	1.16287	
Sundance, 2.79	1.1633	1.09574	
Sundance, 2.99	0.2027	1.12077	
Tomato Garden, 1.99	-0.7791	1.08788	
Tomato Garden, 2.29	0.3087	1.16798	
Tomato Garden, 2.49	0.8876	1.16026	
Tomato Garden, 2.79	-0.1518	1.10376	
Tomato Garden, 2.99	-0.2654	1.13455	
Vegetarian	2.2828	0.68783	27.813
Meat	-0.2596	0.70138	
Italian Sausage	-2.0231	0.86266	
Mushrooms	1.5514	0.38441	20.042
No Mention	-1.5514	0.38441	
All Natural	-0.0347	0.45814	0.448
No Mention	0.0347	0.45814	

The next steps process the `outtest=` data set, saving the R^2 , adjusted R^2 , and df . Subjects whose adjusted R^2 is less than 0.3 (R^2 approximately 0.73) are flagged for exclusion. We want the final analysis to be based on subjects who seemed to be taking the task seriously. The next steps flag the subjects whose fit seems bad and create a macro variable `&droplist` that contains a list of variables to be dropped from the final analysis.

```

data model;
  set utils;
  if statistic in ('R-Square', 'Adj R-Sq', 'Model');
  Subj = scan(_depvar_, 2);
  if statistic = 'Model' then do;
    value = numdf;
    statistic = 'Num DF';
    output;
    value = dendf;
    statistic = 'Den DF';
    output;
    value = dendf + numdf + 1;
    statistic = 'N';
  end;
  output;
  keep statistic value subj;
run;

proc transpose data=model out=summ;
  by subj;
  idlabel statistic;
  id statistic;
run;

data summ2(drop=list);
  length list $ 1000;
  retain list;
  set summ end=eof;
  if adj_r_sq < 0.3 then do;
    Small = '*';
    list = trim(list) || ' ' || subj;
  end;
  if eof then call symput('droplist', trim(list));
run;

%put &droplist;

proc print label data=summ2(drop=_name_ _label_); run;

```

The `outtest=` data set contains for each subject the ANOVA, R^2 , and part-worth utility tables. The numerator df is found in the variable `NumDF`, the denominator df is found in the variable `DenDF`, and the R^2 and adjusted R^2 are found in the variable `Value`. The first DATA step processes the `outtest=` data set, stores all of the statistics of interest in the variable `Value`, and discards the extra observations and variables. The PROC TRANSPOSE step creates a data set with one observation per subject. Here is the `&droplist` macro variable.

```
Sub011 Sub021 Sub031 Sub051 Sub071 Sub081 Sub092 Sub093 Sub094 Sub096
```

Here is some of the R^2 and df summary. We see the df are right, and most of the R^2 's look good.

Conjoint Analysis							
Obs	Subj	Num DF	Den DF	N	R-Square	Adj R-Sq	Small
1	Sub001	18	11	30	0.83441	0.56345	
2	Sub002	18	11	30	0.91844	0.78497	
3	Sub003	18	11	30	0.92908	0.81302	
.
10	Sub010	18	11	30	0.97643	0.93786	
.
84	Sub091	18	11	30	0.85048	0.60581	
85	Sub092	18	11	30	0.64600	0.06673	*
86	Sub093	18	11	30	0.45024	-0.44936	*
87	Sub094	18	11	30	0.62250	0.00477	*
88	Sub095	18	11	30	0.85996	0.63081	
89	Sub096	18	11	30	0.73321	0.29664	*
90	Sub097	18	11	30	0.94155	0.84589	
91	Sub099	18	11	30	0.88920	0.70789	
92	Sub100	18	11	30	0.90330	0.74507	

We can run the conjoint again, this time using the `drop=&droplist` data set option to drop the subjects with poor fit. In the interest of space, the `noprint` option was specified on this step. The printed output will be the same as in the previous step, except for the fact that a few subject's tables are deleted.

```
proc transreg data=inputdata(drop=&droplist) utilities short noprint
  separators=' ' ', ' lprefix=0 outtest=utils method=morals;
  title2 'Conjoint Analysis';
  model identity(sub:) =
    class(brand | price meat mushroom ingredients / zero=sum);
  output p ireplace out=results2 coefficients;
run;
```

Simulating Market Share

In many conjoint analysis studies, the conjoint analysis is not the primary goal. The conjoint analysis is used to generate part-worth utilities, which are then used as input to consumer choice and market share simulators. The end result for a product is its expected “preference share,” which when properly weighted can be used to predict the proportion of times that the product will be purchased. The effects on market share of introducing new products can also be simulated.

One of the most popular ways to simulate market share is with the maximum utility model, which assumes each subject will buy with probability one the product for which he or she has the highest utility. The probabilities for each product are averaged across subjects to get predicted market share.

Other simulation methods include the Bradley-Terry-Luce (BTL) model and the logit model. Unlike the maximum utility model, the BTL and the logit models do not assign all of the probability of choice to the most preferred alternative. Probability is a continuous function of predicted utility. In the maximum utility model, probability of choice is a binary step function of utility. In the BTL model, probability of choice is a linear function of predicted utility. In the logit model, probability of choice is an increasing nonlinear logit function of predicted utility. The BTL model computes the probabilities by dividing each utility by the sum of the predicted utilities within each subject. The logit model divides the exponentiated predicted utilities by the sum of exponentiated utilities, again within subject.

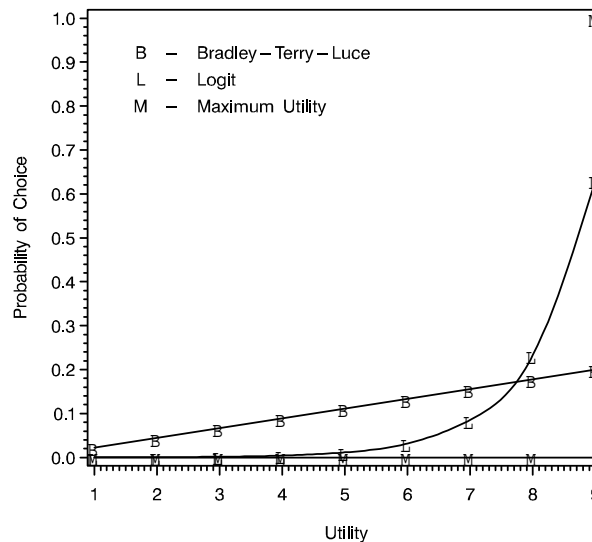
$$\text{Maximum Utility: } p_{ijk} = \begin{cases} 1.0 & \text{if } y_{ijk} = \text{MAX}(y_{ijk}), \\ 0.0 & \text{otherwise} \end{cases}$$

$$\text{BTL: } p_{ijk} = y_{ijk} / \sum \sum \sum y_{ijk}$$

$$\text{Logit: } p_{ijk} = \exp(y_{ijk}) / \sum \sum \sum \exp(y_{ijk})$$

The following plot shows the different assumptions made by the three choice simulators. This plot shows expected market share for a subject with utilities ranging from one to nine.

Simulator Comparisons



The maximum utility line is flat at zero until it reaches the maximum utility, where it jumps to 1.0. The BTL line increases from 0.02 to 0.20 as utility ranges from 1 to 9. The logit function increases exponentially, with small utilities mapping to near-zero probabilities and the largest utility mapping to a proportion of 0.63.

The maximum utility, BTL, and logit models are based on different assumptions and produce different results. The maximum utility model has the advantage of being scale-free. Any strictly monotonic transformation of each subject’s predicted utilities will produce the same market share. However, this model is unstable because it assigns a zero probability of choice to all alternatives that do not have the maximum predicted utility, including those that have predicted utilities near the maximum. The disadvantage of the BTL and logit models is that results are not invariant under linear transformations of the predicted utilities. These methods are considered inappropriate by some researchers for this reason. With negative predicted utilities, the BTL method produces negative probabilities, which are

invalid. The BTL results change when a constant is added to the predicted utilities but do not change when a constant is multiplied by the predicted utilities. Conversely, the logit results change when a constant is multiplied by the predicted utilities but do not change when a constant is added to the predicted utilities. The BTL method is not often used in practice, the logit model is sometimes used, and the maximum utility model is most often used. Refer to Finkbeiner (1988) for a discussion of conjoint analysis choice simulators. Do not confuse a logit model choice simulator and the multinomial logit model; they are quite different.

The three simulation methods will produce different results. This is because all three methods make different assumptions about how consumers translate utility into choice. To see why the models differ, imagine a product that is everyone's second choice. Further imagine that there is wide-spread disagreement on first choice. Every other product is someone's first choice, and all other products are preferred about equally often. In the maximum utility model, this second choice product will have zero probability of choice because no one would choose it first. In the other models, it should be the most preferred, because for every individual it will have a high, near-maximum probability of choice. Of course, preference patterns are not usually as weird as the one just described. If consumers are perfectly rational and always choose the alternative with the highest utility, then the maximum utility model is correct. However, you need to be aware that your results will depend on the choice of simulator model and in BTL and logit, the scaling of the utilities. One reason why the discrete choice model is popular in marketing research is discrete choice models choices directly, whereas conjoint simulates choices indirectly.

Here is the code that made the plot. You can try this program with different minima and maxima to see the effects of linear transformations of the predicted utilities.

```
%let min = 1;
%let max = 9;
%let by = 1;
%let list = &min to &max by &by;
data a;
  sumb = 0;
  suml = 0;
  do u = &list;
    logit = exp(u);
    btl = u;
    sumb = sumb + btl;
    suml = suml + logit;
  end;
  do u = &list;
    logit = exp(u);
    btl = u;
    max = abs(u - (&max)) < (0.5 * (&by));
    btl = btl / sumb;
    logit = logit / suml;
    output;
  end;
run;
```

```

goptions ftext=swiss colors=(black) hsize=4.5in vsize=4.5in;
proc gplot;
  title h=1.5 'Simulator Comparisons';
  plot max * u = 1 btl * u = 2 logit * u = 3 /
    vaxis=axis2 haxis=axis1 overlay frame;
  symbol1 v=M i=step;
  symbol2 v=B i=join;
  symbol3 v=L i=spline;
  axis1 order=(&list) label=('Utility');
  axis2 order=(0 to 1 by 0.1)
    label=(angle=90 "Probability of Choice");
  note move=(2.5cm, 9.2cm)
    font=swissu 'B - ' font=swiss 'Bradley-Terry-Luce';
  note move=(2.5cm, 8.7cm)
    font=swissu 'L - ' font=swiss 'Logit';
  note move=(2.5cm, 8.2cm)
    font=swissu 'M - ' font=swiss 'Maximum Utility';
run; quit;

```

Simulating Market Share, Maximum Utility Model

This section shows how to use the predicted utilities from a conjoint analysis to simulate choice and predict market share. The end result for a hypothetical product is its expected market share, which is a prediction of the proportion of times that the product will be purchased. Note however, that a term like “expected market share,” while widely used, is a misnomer. Without purchase volume data, it is unlikely that these numbers would mirror true market share. Nevertheless, conjoint analysis is a useful and popular marketing research technique.

A SAS macro is used to simulate market share. It takes a `method=morals` output data set from PROC TRANSREG and creates a data set with expected market share for each combination. First, market share is computed with the maximum utility model. The macro finds the most preferred combination(s) for each subject, which are those combinations with the largest predicted utility, and assigns the probability that each combination will be purchased. Typically, for each subject, one product will have a probability of purchase of 1.0, and all other products will have zero probability of purchase. However, when two predicted utilities are tied for the maximum, that subject will have two probabilities of 0.5 and the rest will be zero. The probabilities are averaged across subjects for each product to get market share. Subjects can be differentially weighted.

```

                                /*-----*/
                                /* Simulate Market Share          */
                                /*-----*/
%macro sim(data=_last_, /* SAS data set with utilities.          */
            idvars=,    /* Additional variables to display with */
                                /* market share results.          */
            weights=,  /* By default, each subject contributes */
                                /* equally to the market share      */
                                /* computations. To differentially   */
                                /* weight the subjects, specify a vector */
                                /* of weights, one per subject.      */
                                /* Separate the weights by blanks.    */
            out=shares, /* Output data set name.              */
            method=max  /* max   - maximum utility model.      */
                                /* btl   - Bradley-Terry-Luce model.    */
                                /* logit - logit model.                */
                                /* WARNING: The Bradley-Terry-Luce model */
                                /* and the logit model results are not  */
                                /* invariant under linear                */
                                /* transformations of the utilities.    */
                                /*-----*/
);
options nonotes;

%if &method = btl or &method = logit %then
    %put WARNING: The Bradley-Terry-Luce model and the logit model
    results are not invariant under linear transformations of the
    utilities.;
%else %if &method ne max %then %do;
    %put WARNING: Invalid method &method.. Assuming method=max.;
    %let method = max;
%end;

* Eliminate coefficient observations, if any;
data temp1;
    set &data(where=( _type_ = 'SCORE' or _type_ = ' ' ));
run;

* Determine number of runs and subjects.;
proc sql;
    create table temp2 as select nruns,
        count(nruns) as nsubs, count(distinct nruns) as chk
    from (select count( _depvar_ ) as nruns
    from temp1 where _type_ in ('SCORE', ' ') group by _depvar_);
quit;

```

```

data _null_;
  set temp2;
  call symput('nruns', compress(put(nruns, 5.0)));
  call symput('nsubs', compress(put(nsubs, 5.0)));
  if chk > 1 then do;
    put 'ERROR: Corrupt input data set.';
    call symput('okay', 'no');
  end;
  else call symput('okay', 'yes');
run;

%if &okay ne yes %then %do;
  proc print;
    title2 'Number of runs should be constant across subjects';
  run;
  %goto endit;
%end;

%else %put NOTE: &nruns runs and &nsubs subjects.;
%let w = %scan(&weights, %eval(&nsubs + 1), %str( ));
%if %length(&w) > 0 %then %do;
  %put ERROR: Too many weights.;
  %goto endit;
%end;

* Form nruns by nsubs data set of utilities;
data temp2;
  keep _u1 - _u&nsubs &idvars;
  array u[&nsubs] _u1 - _u&nsubs;
  do j = 1 to &nruns;

    * Read ID variables;
    set temp1(keep=&idvars) point = j;

    * Read utilities;
    k = j;
    do i = 1 to &nsubs;
      set temp1(keep=p_depend_) point = k;
      u[i] = p_depend_;
      %if &method = logit %then u[i] = exp(u[i]);;
      k = k + &nruns;
    end;

    output;
  end;

stop;
run;

```

```

* Set up for maximum utility model;
%if &method = max %then %do;

    * Compute maximum utility for each subject;
    proc means data=temp2 noprint;
        var _u1-_u&nsubs;
        output out=temp1 max=_sum1 - _sum&nsubs;
    run;

    * Flag maximum utility;
    data temp2(keep=_u1 - _u&nsubs &idvars);
        if _n_ = 1 then set temp1(drop=_type_ _freq_);
        array u[&nsubs] _u1 - _u&nsubs;
        array m[&nsubs] _sum1 - _sum&nsubs;
        set temp2;
        do i = 1 to &nsubs;
            u[i] = ((u[i] - m[i]) > -1e-8); /* < 1e-8 is considered 0 */
        end;
    run;

%end;

* Compute sum for each subject;
proc means data=temp2 noprint;
    var _u1-_u&nsubs;
    output out=temp1 sum=_sum1 - _sum&nsubs;
run;

* Compute expected market share;
data &out(keep=share &idvars);
    if _n_ = 1 then set temp1(drop=_type_ _freq_);
    array u[&nsubs] _u1 - _u&nsubs;
    array m[&nsubs] _sum1 - _sum&nsubs;
    set temp2;

    * Compute final probabilities;

    do i = 1 to &nsubs;
        u[i] = u[i] / m[i];
    end;

    * Compute expected market share;

%if %length(&weights) = 0 %then %do;
    Share = mean(of _u1 - _u&nsubs);
%end;

```

```

%else %do;
  Share = 0;
  wsum = 0;
  %do i = 1 %to &nsubs;
    %let w = %scan(&weights, &i, %str( ));
    %if %length(&w) = 0 %then %let w = .;
    if &w < 0 then do;
      if _n_ > 1 then stop;
      put "ERROR: Invalid weight &w..";
      call symput('okay', 'no');
    end;
    share = share + &w * _u&i;
    wsum = wsum + &w;
  %end;
  share = share / wsum;
%end;
run;
options notes;

%if &okay ne yes %then %goto endit;
proc sort;
  by descending share &idvars;
run;
proc print label noobs;
  title2 'Expected Market Share';
  title3 %if &method = max %then "Maximum Utility Model";
         %else %if &method = btl %then "Bradley-Terry-Luce Model";
         %else "Logit Model";;
run;

%endit:

%mend;
title 'Spaghetti Sauces';

%sim(data=results2, out=maxutils, method=max,
      idvars=price brand meat mushroom ingredients);

```

Spaghetti Sauces
Expected Market Share
Maximum Utility Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Sundance	1.99	Vegetarian	Mushrooms	No Mention	0.18293
Pregu	1.99	Vegetarian	No Mention	All Natural	0.14228
Tomato Garden	2.29	Italian Sausage	Mushrooms	No Mention	0.12195
Pregu	2.29	Vegetarian	No Mention	No Mention	0.10976
Pregu	1.99	Vegetarian	Mushrooms	No Mention	0.10366
Tomato Garden	1.99	Vegetarian	Mushrooms	No Mention	0.09146
Tomato Garden	1.99	Vegetarian	No Mention	All Natural	0.07520
Sundance	2.29	Vegetarian	Mushrooms	All Natural	0.07317
Sundance	1.99	Vegetarian	No Mention	All Natural	0.05081
Pregu	2.29	Meat	Mushrooms	All Natural	0.02439
Sundance	2.29	Meat	No Mention	No Mention	0.01220
Sundance	2.49	Italian Sausage	No Mention	No Mention	0.01220
Tomato Garden	2.29	Vegetarian	No Mention	All Natural	0.00000
Pregu	2.49	Vegetarian	Mushrooms	No Mention	0.00000
Pregu	2.49	Italian Sausage	No Mention	No Mention	0.00000
Sundance	2.49	Meat	Mushrooms	All Natural	0.00000
Tomato Garden	2.49	Vegetarian	Mushrooms	All Natural	0.00000
Tomato Garden	2.49	Meat	No Mention	No Mention	0.00000
Pregu	2.79	Meat	No Mention	No Mention	0.00000
Pregu	2.79	Italian Sausage	Mushrooms	No Mention	0.00000
Sundance	2.79	Vegetarian	Mushrooms	No Mention	0.00000
Sundance	2.79	Meat	No Mention	All Natural	0.00000
Tomato Garden	2.79	Vegetarian	No Mention	No Mention	0.00000
Tomato Garden	2.79	Meat	Mushrooms	All Natural	0.00000
Pregu	2.99	Meat	Mushrooms	All Natural	0.00000
Pregu	2.99	Italian Sausage	No Mention	No Mention	0.00000
Sundance	2.99	Vegetarian	No Mention	All Natural	0.00000
Sundance	2.99	Meat	Mushrooms	No Mention	0.00000
Tomato Garden	2.99	Vegetarian	No Mention	No Mention	0.00000
Tomato Garden	2.99	Italian Sausage	Mushrooms	No Mention	0.00000

The largest market share (18.29%) is for Sundance brand vegetarian sauce with mushrooms costing \$1.99. The next largest share (14.23%) is Pregu brand vegetarian sauce with all natural ingredients costing \$1.99. Five of the seven most preferred sauces all cost \$1.99—the minimum. It is not clear from this simulation if any brand is the leader.

Simulating Market Share, Bradley-Terry-Luce and Logit Models

The Bradley-Terry-Luce model and the logit model are also available in the %SIM macro.

```

title 'Spaghetti Sauces';

%sim(data=results2, out=bt1, method=bt1,
      idvars=price brand meat mushroom ingredients);

%sim(data=results2, out=logit, method=logit,
      idvars=price brand meat mushroom ingredients);

```

Spaghetti Sauces Expected Market Share Bradley-Terry-Luce Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Pregu	1.99	Vegetarian	Mushrooms	No Mention	0.053479
Sundance	1.99	Vegetarian	Mushrooms	No Mention	0.052990
Tomato Garden	1.99	Vegetarian	Mushrooms	No Mention	0.051751
Pregu	1.99	Vegetarian	No Mention	All Natural	0.050683
Sundance	1.99	Vegetarian	No Mention	All Natural	0.050193
Tomato Garden	1.99	Vegetarian	No Mention	All Natural	0.048955
Sundance	2.29	Vegetarian	Mushrooms	All Natural	0.048236
Pregu	2.29	Vegetarian	No Mention	No Mention	0.043972
Tomato Garden	2.29	Vegetarian	No Mention	All Natural	0.042035
Pregu	2.49	Vegetarian	Mushrooms	No Mention	0.041532
Pregu	2.29	Meat	Mushrooms	All Natural	0.041063
Sundance	2.29	Meat	No Mention	No Mention	0.036321
Tomato Garden	2.29	Italian Sausage	Mushrooms	No Mention	0.032995
Sundance	2.79	Vegetarian	Mushrooms	No Mention	0.032067
Sundance	2.49	Meat	Mushrooms	All Natural	0.031310
Tomato Garden	2.49	Vegetarian	Mushrooms	All Natural	0.031057
Sundance	2.99	Vegetarian	No Mention	All Natural	0.026879
Pregu	2.49	Italian Sausage	No Mention	No Mention	0.026046
Pregu	2.99	Meat	Mushrooms	All Natural	0.025318
Pregu	2.79	Meat	No Mention	No Mention	0.025038
Tomato Garden	2.79	Vegetarian	No Mention	No Mention	0.024325
Pregu	2.79	Italian Sausage	Mushrooms	No Mention	0.024263
Sundance	2.49	Italian Sausage	No Mention	No Mention	0.022383
Sundance	2.99	Meat	Mushrooms	No Mention	0.022264
Tomato Garden	2.99	Vegetarian	No Mention	No Mention	0.022113
Sundance	2.79	Meat	No Mention	All Natural	0.021858
Tomato Garden	2.79	Meat	Mushrooms	All Natural	0.021415
Tomato Garden	2.49	Meat	No Mention	No Mention	0.019142
Pregu	2.99	Italian Sausage	No Mention	No Mention	0.016391
Tomato Garden	2.99	Italian Sausage	Mushrooms	No Mention	0.013926

Spaghetti Sauces
Expected Market Share
Logit Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Sundance	1.99	Vegetarian	Mushrooms	No Mention	0.10463
Pregu	1.99	Vegetarian	No Mention	All Natural	0.09621
Tomato Garden	1.99	Vegetarian	Mushrooms	No Mention	0.09001
Pregu	1.99	Vegetarian	Mushrooms	No Mention	0.08358
Pregu	2.29	Vegetarian	No Mention	No Mention	0.07755
Sundance	2.29	Vegetarian	Mushrooms	All Natural	0.07102
Tomato Garden	1.99	Vegetarian	No Mention	All Natural	0.06872
Tomato Garden	2.29	Italian Sausage	Mushrooms	No Mention	0.06735
Sundance	1.99	Vegetarian	No Mention	All Natural	0.06419
Pregu	2.29	Meat	Mushrooms	All Natural	0.04137
Pregu	2.49	Vegetarian	Mushrooms	No Mention	0.03578
Sundance	2.29	Meat	No Mention	No Mention	0.03273
Sundance	2.49	Italian Sausage	No Mention	No Mention	0.02081
Tomato Garden	2.99	Italian Sausage	Mushrooms	No Mention	0.02055
Sundance	2.79	Vegetarian	Mushrooms	No Mention	0.02022
Tomato Garden	2.29	Vegetarian	No Mention	All Natural	0.01996
Pregu	2.79	Italian Sausage	Mushrooms	No Mention	0.01233
Pregu	2.49	Italian Sausage	No Mention	No Mention	0.01199
Sundance	2.49	Meat	Mushrooms	All Natural	0.01010
Sundance	2.99	Meat	Mushrooms	No Mention	0.00964
Pregu	2.79	Meat	No Mention	No Mention	0.00763
Pregu	2.99	Italian Sausage	No Mention	No Mention	0.00637
Pregu	2.99	Meat	Mushrooms	All Natural	0.00547
Tomato Garden	2.49	Vegetarian	Mushrooms	All Natural	0.00538
Tomato Garden	2.79	Meat	Mushrooms	All Natural	0.00516
Sundance	2.99	Vegetarian	No Mention	All Natural	0.00399
Sundance	2.79	Meat	No Mention	All Natural	0.00266
Tomato Garden	2.79	Vegetarian	No Mention	No Mention	0.00209
Tomato Garden	2.99	Vegetarian	No Mention	No Mention	0.00162
Tomato Garden	2.49	Meat	No Mention	No Mention	0.00088

The three methods produce different results.

Change in Market Share

The following steps simulate what would happen to the market if new products were introduced. Simulation observations are added to the data set and given zero weight. The conjoint analyses are rerun to compute the predicted utilities for the active observations and the simulations. The maximum utility model is used.

Recall that the design has numeric variables with values like 1, 2, and 3. Formats are used to print the descriptions of the levels of the attributes. The first thing we want to do is read in products

to simulate. We could read in values like 1, 2, and 3 or we could read in more descriptive values and convert them to numerics using informats. We chose the latter approach. First we use PROC FORMAT to create the informats. Previously, we created formats with PROC FORMAT by specifying a `value` statement followed by pairs of the form *numeric-value=descriptive-character-string*. We create an informat with PROC FORMAT by specifying an `invalue` statement followed by pairs of the form *descriptive-character-string=numeric-value*.

```

title 'Spaghetti Sauces';

proc format;
  invalue inbrand 'Preg'=1 'Sun' =2 'Tom' =3;
  invalue inmeat  'Veg' =1 'Meat'=2 'Ital'=3;
  invalue inmush  'Mush'=1 'No'  =2;
  invalue iningre 'Nat' =1 'No'  =2;
  invalue inprice '1.99'=1 '2.29'=2 '2.49'=3 '2.79'=4 '2.99'=5;
run;

```

Next, we read the observations we want to consider for a sample market using the informats we just created. An `input` statement specification of the form “*variable : informat*” reads values starting with the first nonblank character.

```

data simulat;
  input brand      : inbrand.
        meat      : inmeat.
        mushroom   : inmush.
        ingredients : iningre.
        price      : inprice.;
  datalines;
Preg Veg  Mush Nat  1.99
Sun  Veg  Mush Nat  1.99
Tom  Veg  Mush Nat  1.99
Preg Meat Mush Nat  2.49
Sun  Meat Mush Nat  2.49
Tom  Meat Mush Nat  2.49
Preg Ital Mush Nat  2.79
Sun  Ital Mush Nat  2.79
Tom  Ital Mush Nat  2.79
;

```

Next, the original input data set is combined with the simulation observations. The subjects with poor fit are dropped and a `weight` variable is created to flag the simulation observations. The `weight` variable is not strictly necessary since all of the simulation observations will have missing values on the ratings so will be excluded from the analysis that way. Still, it is good practice to explicitly use weights to exclude observations.

```

data inputdata2(drop=&droplist);
  set inputdata(in=w) simulat;
  Weight = w;
run;

```

```

proc print;
  title2 'Simulation Observations Have a Weight of Zero';
  id weight;
  var brand -- price;
run;

```

Spaghetti Sauces
Simulation Observations Have a Weight of Zero

Weight	Brand	Meat	Mushroom	Ingredients	Price
1	Pregu	Meat	No Mention	No Mention	2.79
1	Tomato Garden	Vegetarian	No Mention	No Mention	2.79
1	Pregu	Meat	Mushrooms	All Natural	2.29
1	Tomato Garden	Vegetarian	Mushrooms	All Natural	2.49
1	Sundance	Vegetarian	Mushrooms	No Mention	1.99
1	Pregu	Italian Sausage	No Mention	No Mention	2.49
1	Tomato Garden	Vegetarian	No Mention	No Mention	2.99
1	Tomato Garden	Italian Sausage	Mushrooms	No Mention	2.29
1	Pregu	Vegetarian	Mushrooms	No Mention	2.49
1	Pregu	Vegetarian	No Mention	No Mention	2.29
1	Sundance	Vegetarian	Mushrooms	No Mention	2.79
1	Tomato Garden	Vegetarian	Mushrooms	No Mention	1.99
1	Sundance	Meat	No Mention	No Mention	2.29
1	Sundance	Meat	Mushrooms	No Mention	2.99
1	Pregu	Italian Sausage	Mushrooms	No Mention	2.79
1	Tomato Garden	Italian Sausage	Mushrooms	No Mention	2.99
1	Sundance	Vegetarian	Mushrooms	All Natural	2.29
1	Pregu	Meat	Mushrooms	All Natural	2.99
1	Tomato Garden	Meat	No Mention	No Mention	2.49
1	Sundance	Meat	Mushrooms	All Natural	2.49
1	Pregu	Vegetarian	No Mention	All Natural	1.99
1	Sundance	Meat	No Mention	All Natural	2.79
1	Tomato Garden	Vegetarian	No Mention	All Natural	1.99
1	Sundance	Italian Sausage	No Mention	No Mention	2.49
1	Sundance	Vegetarian	No Mention	All Natural	1.99
1	Sundance	Vegetarian	No Mention	All Natural	2.99
1	Pregu	Italian Sausage	No Mention	No Mention	2.99
1	Tomato Garden	Vegetarian	No Mention	All Natural	2.29
1	Pregu	Vegetarian	Mushrooms	No Mention	1.99
1	Tomato Garden	Meat	Mushrooms	All Natural	2.79
0	Pregu	Vegetarian	Mushrooms	All Natural	1.99
0	Sundance	Vegetarian	Mushrooms	All Natural	1.99
0	Tomato Garden	Vegetarian	Mushrooms	All Natural	1.99
0	Pregu	Meat	Mushrooms	All Natural	2.49
0	Sundance	Meat	Mushrooms	All Natural	2.49

0	Tomato Garden	Meat	Mushrooms	All Natural	2.49
0	Pregu	Italian Sausage	Mushrooms	All Natural	2.79
0	Sundance	Italian Sausage	Mushrooms	All Natural	2.79
0	Tomato Garden	Italian Sausage	Mushrooms	All Natural	2.79

The next steps run the conjoint analyses suppressing the printed output using the `noprnt` option. The statement `weight weight` is specified since we want the simulation observations (which have zero weight) excluded from contributing to the analysis. However, the procedure will still compute an expected utility for every observation including observations with zero, missing, and negative weights. The `outtest=` data set is created like before so we can check to make sure the df and R^2 look reasonable.

```
ods exclude notes mvanova anova;
proc transreg data=inputdata2 utilities short noprint
  separators=', ' lprefix=0 method=morals outtest=utils;
  title2 'Conjoint Analysis';
  model identity(sub:) =
    class(brand | price meat mushroom ingredients / zero=sum);
  output p ireplace out=results3 coefficients;
  weight weight;
run;

data model;
  set utils;
  if statistic in ('R-Square', 'Adj R-Sq', 'Model');
  Subj = scan(_depvar_, 2);
  if statistic = 'Model' then do;
    value = numdf;
    statistic = 'Num DF';
    output;
    value = dendf;
    statistic = 'Den DF';
    output;
    value = dendf + numdf + 1;
    statistic = 'N';
  end;
  output;
  keep statistic value subj;
run;

proc transpose data=model out=summ;
  by subj;
  idlabel statistic;
  id statistic;
run;

proc print label data=summ(drop=_name_ _label_); run;
```

The SAS log tells us that the nine simulation observations were deleted both because of zero weight and because of missing values in the dependent variables.

NOTE: 9 observations were deleted from the analysis but not from the output data set due to missing values.

NOTE: 9 observations were deleted from the analysis but not from the output data set due to nonpositive weights.

NOTE: A total of 9 observations were deleted.

The *df* and R^2 results, some of which are shown next, look fine.

Spaghetti Sauces
Conjoint Analysis

Obs	Subj	Num DF	Den DF	N	R-Square	Adj R-Sq
1	Sub001	18	11	30	0.83441	0.56345
2	Sub002	18	11	30	0.91844	0.78497
3	Sub003	18	11	30	0.92908	0.81302
.
.
81	Sub099	18	11	30	0.88920	0.70789
82	Sub100	18	11	30	0.90330	0.74507

The simulation observations are pulled out of the out= data set, and the %SIM macro is run to simulate market share.

```
data results4;
  set results3;
  where weight = 0;
  run;

%sim(data=results4, out=shares2, method=max,
      idvars=price brand meat mushroom ingredients);
```

Spaghetti Sauces
Expected Market Share
Maximum Utility Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Pregu	1.99	Vegetarian	Mushrooms	All Natural	0.35976
Sundance	1.99	Vegetarian	Mushrooms	All Natural	0.29878
Tomato Garden	1.99	Vegetarian	Mushrooms	All Natural	0.19512
Tomato Garden	2.79	Italian Sausage	Mushrooms	All Natural	0.08537
Sundance	2.79	Italian Sausage	Mushrooms	All Natural	0.02439
Pregu	2.49	Meat	Mushrooms	All Natural	0.01220
Sundance	2.49	Meat	Mushrooms	All Natural	0.01220
Pregu	2.79	Italian Sausage	Mushrooms	All Natural	0.01220
Tomato Garden	2.49	Meat	Mushrooms	All Natural	0.00000

For this set of products, the inexpensive vegetarian sauces have the greatest market share with Prego brand preferred over Sundance and Tomato Garden. Now we'll consider adding six more products to the market, the six meat sauces we just saw, but at a lower price.

```

data simulat2;
  input brand      : inbrand.
        meat      : inmeat.
        mushroom  : inmush.
        ingredients : iningre.
        price      : inprice.;
  datalines;
Preg Meat Mush Nat 2.29
Sun  Meat Mush Nat 2.29
Tom  Meat Mush Nat 2.29
Preg Ital Mush Nat 2.49
Sun  Ital Mush Nat 2.49
Tom  Ital Mush Nat 2.49
;
data inputdata3(drop=&droplist);
  set inputdata(in=w) simulat simulat2;
  weight = w;
  run;

ods exclude notes mvanova anova;
proc transreg data=inputdata3 utilities short noprint
  separators=', ' lprefix=0 method=morals outtest=utils;
  title2 'Conjoint Analysis';
  model identity(sub:) =
    class(brand | price meat mushroom ingredients / zero=sum);
  output p ireplace out=results5 coefficients;
  weight weight;
  run;

```

Now we see that 15 simulation observations were excluded.

NOTE: 15 observations were deleted from the analysis but not from the output data set due to missing values.

NOTE: 15 observations were deleted from the analysis but not from the output data set due to nonpositive weights.

NOTE: A total of 15 observations were deleted.

These steps extract the df and R^2 .

```

data model;
  set utils;
  if statistic in ('R-Square', 'Adj R-Sq', 'Model');
  Subj = scan(_depvar_, 2);
  if statistic = 'Model' then do;
    value = numdf;
    statistic = 'Num DF';
    output;
    value = dendf;
    statistic = 'Den DF';
    output;
    value = dendf + numdf + 1;
    statistic = 'N';
  end;
  output;
  keep statistic value subj;
run;

proc transpose data=model out=summ;
  by subj;
  idlabel statistic;
  id statistic;
  run;

proc print label data=summ(drop=_name_ _label_); run;

```

The df and R^2 still look fine.

Spaghetti Sauces Conjoint Analysis						
Obs	Subj	Num DF	Den DF	N	R-Square	Adj R-Sq
1	Sub001	18	11	30	0.83441	0.56345
2	Sub002	18	11	30	0.91844	0.78497
3	Sub003	18	11	30	0.92908	0.81302
.						
.						
.						
81	Sub099	18	11	30	0.88920	0.70789
82	Sub100	18	11	30	0.90330	0.74507

Now we'll run the simulation with all 15 simulation observations.

```

data results6;
  set results5;
  where weight = 0;
  run;

%sim(data=results6, out=shares3, method=max,
      idvars=price brand meat mushroom ingredients);

```

Spaghetti Sauces
Expected Market Share
Maximum Utility Model

Brand	Price	Meat	Mushroom	Ingredients	Share
Sundance	1.99	Vegetarian	Mushrooms	All Natural	0.25813
Pregu	1.99	Vegetarian	Mushrooms	All Natural	0.20935
Pregu	2.29	Meat	Mushrooms	All Natural	0.19512
Tomato Garden	1.99	Vegetarian	Mushrooms	All Natural	0.15447
Sundance	2.49	Italian Sausage	Mushrooms	All Natural	0.08537
Sundance	2.29	Meat	Mushrooms	All Natural	0.03659
Tomato Garden	2.49	Italian Sausage	Mushrooms	All Natural	0.01829
Tomato Garden	2.29	Meat	Mushrooms	All Natural	0.01220
Pregu	2.49	Italian Sausage	Mushrooms	All Natural	0.01220
Tomato Garden	2.79	Italian Sausage	Mushrooms	All Natural	0.01220
Sundance	2.79	Italian Sausage	Mushrooms	All Natural	0.00610
Pregu	2.49	Meat	Mushrooms	All Natural	0.00000
Sundance	2.49	Meat	Mushrooms	All Natural	0.00000
Tomato Garden	2.49	Meat	Mushrooms	All Natural	0.00000
Pregu	2.79	Italian Sausage	Mushrooms	All Natural	0.00000

These steps merge the data set containing the old market shares with the data set containing the new market shares to show the effect of adding the new products.

```

title 'Spaghetti Sauces';

proc sort data=shares2;
  by price brand meat mushroom ingredients;
  run;

proc sort data=shares3;
  by price brand meat mushroom ingredients;
  run;

data both;
  merge shares2(rename=(share=OldShare)) shares3;
  by price brand meat mushroom ingredients;
  if oldshare = . then Change = 0;
  else change = oldshare;
  change = share - change;
  run;

```

```

proc sort;
  by descending share price brand meat mushroom ingredients;
run;
options missing=' ';
proc print noobs;
  title2 'Expected Market Share and Change';
  var price brand meat mushroom ingredients
      oldshare share change;
  format oldshare -- change 6.3;
run;
options missing=.;

```

Spaghetti Sauces
Expected Market Share and Change

Price	Brand	Meat	Mushroom	Ingredients	Old	
					Share	Share Change
1.99	Sundance	Vegetarian	Mushrooms	All Natural	0.299	0.258 -0.041
1.99	Pregu	Vegetarian	Mushrooms	All Natural	0.360	0.209 -0.150
2.29	Pregu	Meat	Mushrooms	All Natural		0.195 0.195
1.99	Tomato Garden	Vegetarian	Mushrooms	All Natural	0.195	0.154 -0.041
2.49	Sundance	Italian Sausage	Mushrooms	All Natural		0.085 0.085
2.29	Sundance	Meat	Mushrooms	All Natural		0.037 0.037
2.49	Tomato Garden	Italian Sausage	Mushrooms	All Natural		0.018 0.018
2.29	Tomato Garden	Meat	Mushrooms	All Natural		0.012 0.012
2.49	Pregu	Italian Sausage	Mushrooms	All Natural		0.012 0.012
2.79	Tomato Garden	Italian Sausage	Mushrooms	All Natural	0.085	0.012 -0.073
2.79	Sundance	Italian Sausage	Mushrooms	All Natural	0.024	0.006 -0.018
2.49	Pregu	Meat	Mushrooms	All Natural	0.012	0.000 -0.012
2.49	Sundance	Meat	Mushrooms	All Natural	0.012	0.000 -0.012
2.49	Tomato Garden	Meat	Mushrooms	All Natural	0.000	0.000 0.000
2.79	Pregu	Italian Sausage	Mushrooms	All Natural	0.012	0.000 -0.012

We see that the vegetarian sauces are most preferred, but we predict they would lose share if the new meat sauces were entered in the market. In particular, the Sundance and Pregu meat sauces would gain significant market share under this model.

PROC TRANSREG Specifications

PROC TRANSREG (transformation regression) is used to perform conjoint analysis and many other types of analyses, including simple regression, multiple regression, redundancy analysis, canonical correlation, analysis of variance, and external unfolding, all with nonlinear transformations of the variables. This section documents the statements and options available in PROC TRANSREG that are commonly used in conjoint analyses. Refer to “The TRANSREG Procedure” in the *SAS/STAT User’s Guide* for more information on PROC TRANSREG. This section documents only a small subset of the capabilities of PROC TRANSREG.

The following statements are used in the TRANSREG procedure for conjoint analysis:

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
    <a-options> <o-options>;
MODEL transform(dependents </ t-options>) =
    transform(independents </ t-options>)
    <transform(independents </ t-options>) ...> </ a-options>;
OUTPUT <OUT=SAS-data-set> <o-options>;
WEIGHT variable;
ID variables;
BY variables;
```

Specify the `proc` and `model` statements to use PROC TRANSREG. The `output` statement is required to produce an `out=` output data set, which contains the transformations, indicator variables, and predicted utility for each product. The `outtest=` data set, which contains the ANOVA, regression, and part-worth utility tables, is requested in the `proc` statement. All options can be abbreviated to their first three letters.

PROC TRANSREG *Statement*

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
    <a-options> <o-options>;
```

The `data=` and `outtest=` options can appear only in the PROC TRANSREG statement. The algorithm options (*a-options*) appear in the `proc` or `model` statement. The output options (*o-options*) can appear in the `proc` or `output` statement.

DATA=SAS-data-set

specifies the input SAS data. If the `data=` option is not specified, PROC TRANSREG uses the most recently created SAS data set.

OUTTEST=SAS-data-set

specifies an output data set that will contain the ANOVA table, R^2 , and the conjoint analysis part-worth utilities, and the attribute importances.

Algorithm Options

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
    <a-options> <o-options>;
MODEL transform(dependents </ t-options>) =
    transform(independents </ t-options>)
    <transform(independents </ t-options>) ...> </ a-options>;
```

Algorithm options can appear in the `proc` or `model` statement as *a-options*.

CONVERGE=*n*

specifies the minimum average absolute change in standardized variable scores that is required to continue iterating. By default, `converge=0.00001`.

DUMMY

requests a canonical initialization. When `spline` transformations are requested, specify `dummy` to solve for the optimal transformations without iteration. Iteration is only necessary when there are monotonicity constraints.

LPREFIX=*n*

specifies the number of first characters of a `class` variable's label (or name if no label is specified) to use in constructing labels for part-worth utilities. For example, the default label for `Brand=Duff` is "Brand Duff". If you specify `lprefix=0` then the label is simply "Duff".

MAXITER=*n*

specifies the maximum number of iterations. By default, `maxiter=30`.

NOPRINT

suppresses the display of all output.

ORDER=FORMATTED

ORDER=INTERNAL

specifies the order in which the `CLASS` variable levels are reported. The default, `order=internal`, sorts by unformatted value. Specify `order=formatted` when you want the levels sorted by formatted value. Sort order is machine dependent. Note that in Version 6 and Version 7 of the SAS System, the default sort order was `order=formatted`. The default was changed to `order=internal` in Version 8 to be consistent with Base SAS procedures.

METHOD=MORALS

METHOD=UNIVARIATE

specifies the iterative algorithm. Both `method=morals` and `method=univariate` fit univariate multiple regression models with the possibility of nonlinear transformations of the variables. They differ in the way they structure the output data set when there is more than one dependent variable. When it can be used, `method=univariate` is more efficient than `method=morals`.

You can use `method=univariate` when no transformations of the independent variables are requested, for example when the independent variables are all designated `class`, `identity`, or `pspline`. In this case, the final set of independent variables will be the same for all subjects. If transformations such as

`monotone`, `identity`, `spline` or `mspline` are specified for the independent variables, the transformed independent variables may be different for each dependent variable and so must be output separately for each dependent variable. In conjoint analysis, there will typically be one dependent variable for each subject. This is illustrated in the examples.

With `method=univariate` and more than one dependent variable, PROC TRANSREG creates a data set with the same number of score observations as the original but with more variables. The untransformed dependent variable names are unchanged. The default transformed dependent variable names consist of the prefix “T” and the original variable names. The default predicted value names consist of the prefix “P” and the original variable names. The full set of independent variables appears once.

When more than one dependent variable is specified, `method=morals` creates a *rolled-out* data set with the dependent variable in `_depend_`, its transformation in `t_depend_`, and its predicted values in `p_depend_`. The full set of independents is repeated for each (original) dependent variable.

The procedure chooses a default method based on what is specified in the `model` statement. When transformations of the independent variables are requested, the default method is `morals`. Otherwise the default method is `univariate`.

`SEPARATORS=string-1 <string-2 >`

specifies separators for creating labels for the part-worth utilities. By default, `separators=' ' * '` (“blank” and “blank asterisk blank”). The first value is used to separate variable names and values in interactions. The second value is used to separate interaction components. For example, the default label for `Brand=Duff` is “Brand Duff”. If you specify `separators=', '` then the label is “Brand, Duff”. Furthermore, the default label for the interaction of `Brand=Duff` and `Price=3.99` is “Brand Duff * Price 3.99”. You could specify `lprefix=0` and `separators=' ' @ '` to instead create labels like “Duff @ 3.99”. You use the `lprefix=0` option when you want to construct labels using zero characters of the variable name, that is when you want to construct labels from just the formatted level. The option `separators=' ' @ '` specifies in the second string a separator of the form “blank at blank”. In this case, the first string is ignored because with `lprefix=0` there is no name to separate from the level.

SHORT

suppresses the iteration histories. For most standard metric conjoint analyses, no iterations are necessary, so specifying `short` eliminates unnecessary output. PROC TRANSREG will print a message if it ever fails to converge, so it is usually safe to specify the `short` option.

UTILITIES

prints the part-worth utilities and importances table and an ANOVA table. Note that you can use an `ods exclude` statement to exclude ANOVA tables and unnecessary notes from the conjoint output (see page 486).

Output Options

```
PROC TRANSREG <DATA=SAS-data-set> <OUTTEST=SAS-data-set>
              <a-options> <o-options>;
              OUTPUT <OUT=SAS-data-set> <o-options>;
```

The `out=` option can only appear in the `output` statement. The other output options can appear in the `proc` or `output` statement as *o-options*.

COEFFICIENTS

outputs the part-worth utilities to the `out=` data set.

P

includes the predicted values in the `out=` output data set, which are the predicted utilities for each product. By default, the predicted values variable name is the original dependent variable name prefixed with a “P”.

IREPLACE

replaces the original independent variables with the transformed independent variables in the output data set. The names of the transformed variables in the output data set correspond to the names of the original independent variables in the input data set.

OUT=*SAS-data-set*

names the output data set. When an `output` statement is specified without the `out=` option, PROC TRANSREG creates a data set and uses the `DATA n` convention. To create a permanent SAS data set, specify a two-level name. The data set will contain the original input variables, the coded indicator variables, the transformation of the dependent variable, and the optionally predicted utilities for each product.

RESIDUALS

outputs to the `out=` data set the differences between the observed and predicted utilities. By default, the residual variable name is the original dependent variable name prefixed with an “R”.

Transformations and Expansions

```
MODEL transform(dependents </ t-options>) =
      transform(independents </ t-options>)
      <transform(independents </ t-options>) ...> </ a-options>;
```

The operators “*”, “|”, and “@” from the GLM procedure are available for interactions with `class` variables.

```
class(a * b ...
      c | d ...
      e | f ... @ n)
```

For example, this statement fits 100 individual main-effects models:

```
model identity(rating1-rating100) = class(x1-x5 / zero=sum);
```

This fits models with main effects and all two-way interactions:

```
model identity(rating1-rating100) = class(x1|x2|x3|x4|x5@2 / zero=sum);
```

This fits models with main effects and some two-way interactions:

```
model identity(rating1-rating100) = class(x1-x5 x1*x2 x3*x4 / zero=sum);
```

You can also fit separate price functions within each brand by specifying:

```
model identity(rating1-rating100) =
      class(brand / zero=none) | spline(price);
```

The list `x1-x5` is equivalent to `x1 x2 x3 x4 x5`. The vertical bar specifies all main effects and interactions, and the `at` sign limits the interactions. For example, `@2` limits the model to main effects and two-way interactions. The list `x1|x2|x3|x4|x5@2` is equivalent to `x1 x2 x1 * x2 x3 x1 * x3 x2 * x3 x4 x1 * x4 x2 * x4 x3 * x4 x5 x1 * x5 x2 * x5 x3 * x5 x4 * x5`. The specification `x1 * x2` indicates the two-way interaction between `x1` and `x2`, and `x1 * x2 * x3` indicates the three-way interaction between `x1`, `x2`, and `x3`.

Each of the following can be specified in the `model` statement as a *transform*. The `pspline` and `class` expansions create more than one output variable for each input variable. The rest are transformations that create one output variable for each input variable.

CLASS

designates variables for analysis as nominal-scale-of-measurement variables. For conjoint analysis, the `zero=sum` *t-option* is typically specified: `class(variables / zero=sum)`. Variables designated as `class` variables are expanded to a set of indicator variables. Usually the number output variables for each `class` variable is the number of different values in the input variables. Dependent variables should not be designated as `class` variables.

IDENTITY

variables are not changed by the iterations. The `identity(variables)` specification designates interval-scale-of-measurement variables when no transformation is permitted. When small data values mean high preference, you will need to use the `reflect` transformation option.

MONOTONE

monotonically transforms variables; ties are preserved. When `monotone(variables)` is used with dependent variables, a nonmetric conjoint analysis is performed. When small data values mean high preference, you will need to use the `reflect` transformation option. The `monotone` specification can also be used with independent variables to impose monotonicity on the part-worth utilities. When it is known that monotonicity should exist in an attribute variable, using `monotone` instead of `class` for that attribute may improve prediction. An option exists in PROC TRANSREG for optimally untying tied values, but this option should not be used because it almost always produces a degenerate result.

MSPLINE

monotonically and smoothly transforms variables. By default, `mspline(variables)` fits a monotonic quadratic spline with no knots. Knots are specified as *t-options*, for example `mspline(variables / nknots=3)` or `mspline(variables / knots=5 to 15 by 5)`. Like `monotone`, `mspline` finds a monotonic transformation. Unlike `monotone`, `mspline` places a bound on the *df* (number of knots + degree) used by the transformation. With `mspline`, it is possible to allow for nonlinearity in the responses and still have error *df*. This is not always possible with `monotone`. When small data values mean high preference, you will need to use the `reflect` transformation option. You can also use `mspline` with attribute variables to impose monotonicity on the part-worth utilities.

PSPLINE

expands each variable to a piece-wise polynomial spline basis. By default, `pspline(variables)` uses a cubic spline with no knots. Knots are specified as *t-options*. Specify `pspline(variable / degree=2)` for an attribute variable to fit a quadratic model. For each `pspline` variable, $d + k$ output variables are created, where d is the degree of the polynomial and k is the number of knots. You should not specify `pspline` with the dependent variables.

RANK

performs a rank transformation, with ranks averaged within ties. Rating-scale data can be transformed to ranks by specifying `rank(variables)`. When small data values mean high preference, you will need to use the `reflect` transformation option. Typically, `rank` is only used for dependent variables. For example, if a rating-scale variable has sorted values 1, 1, 1, 2, 3, 3, 4, 5, 5, 5, then the rank transformation is 2, 2, 2, 4, 5.5, 5.5, 7, 9, 9, 9. A conjoint analysis of the original rating-scale variable will not usually be the same as a conjoint analysis of a rank transformation of the ratings. With ordinal-scale-of-measurement data, it is often good to analyze rank transformations instead of the original data. An alternative is to specify `monotone`, which performs a nonmetric conjoint analysis. For real data, `monotone` will always find a better fit than `rank`, but `rank` may lead to better prediction.

SPLINE

smoothly transforms variables. By default, `spline(variables)` fits a cubic spline with no knots. Knots are specified as *t-options*. Like `pspline`, `spline` models nonlinearities in the attributes.

Transformation Options

```
MODEL transform(dependents </ t-options>) =
  transform(independents </ t-options>)
  <transform(independents </ t-options>) ...> </ a-options>;
```

The following are specified in the `model` statement as *t-options*'s.

DEGREE=*n*

specifies the degree of the spline. The defaults are `degree=3` (cubic spline) for `spline` and `pspline`, and `degree=2` (quadratic spline) for `mspline`. For example, to request a quadratic spline, specify `spline(variables / degree=2)`.

EVENLY

is used with the `nknots=` option to evenly space the knots for splines. For example, if `spline(x / nknots=2 evenly)` is specified and `x` has a minimum of 4 and a maximum of 10, then the two interior knots are 6 and 8. Without `evenly`, the `nknots=` option places knots at percentiles, so the knots are not evenly spaced.

KNOTS=*numberlist*

specifies the interior knots or break points for splines. By default, there are no knots. For example, to request knots at 1, 2, 3, 4, 5, specify `spline(variable / knots=1 to 5)`.

NKNOTS=*k*

creates *k* knots for splines: the first at the 100/(*k*+1) percentile, the second at the 200/(*k*+1) percentile, and so on. Unless **evenly** is specified, knots are placed at data values; there is no interpolation. For example, with **spline(variable / NKNOTS=3)**, knots are placed at the twenty-fifth percentile, the median, and the seventy-fifth percentile. By default, **nknots=0**.

REFLECT

reflects the transformation around its mean, $Y = -(Y - \bar{Y}) + \bar{Y}$, after the iterations are completed and before the final standardization and results calculations. This option is particularly useful with the dependent variable. When the dependent variable consists of ranks with the most preferred combination assigned 1.0, **identity(variable / reflect)** will reflect the transformation so that positive utilities mean high preference.

ZERO=SUM

constrains the part-worth utilities to sum to zero within each attribute. The specification **class(variables / zero=sum)** creates a less than full rank model, but the coefficients are uniquely determined due to the sum-to-zero constraint.

BY Statement

BY variables;

A **by** statement can be used with PROC TRANSREG to obtain separate analyses on observations in groups defined by the **by** variables. When a **by** statement appears, the procedure expects the input data set to be sorted in order of the **by** variables.

If the input data set is not sorted in ascending order, use one of the following alternatives:

- Use the SORT procedure with a similar **by** statement to sort the data.
- Use the **by** statement options **notsorted** or **descending** in the **by** statement for the TRANSREG procedure. As a cautionary note, the **notsorted** option does not mean that the data are unsorted. It means that the data are arranged in groups (according to values of the **by** variables), and these groups are not necessarily in alphabetical or increasing numeric order.
- Use the DATASETS procedure (in base SAS software) to create an index on the **by** variables.

For more information on the **by** statement, refer to the discussion in *SAS Language: Reference*. For more information on the DATASETS procedure, refer to the discussion in *SAS Procedures Guide*.

ID Statement

ID variables;

The **id** statement includes additional character or numeric variables from the input data set in the **out=** data set.

WEIGHT *Statement*

`WEIGHT variable;`

A `weight` statement can be used in conjoint analysis to distinguish ordinary active observations, holdouts, and simulation observations. When a `weight` statement is used, a weighted residual sum of squares is minimized. The observation is used in the analysis only if the value of the `weight` statement variable is greater than zero. For observations with positive weight, the `weight` statement has no effect on *df* or number of observations, but the weights affect most other calculations.

Assign each active observation a weight of 1. Assign each holdout observation a weight that excludes it from the analysis, such as missing. Assign each simulation observation a different weight that excludes it from the analysis, such as zero. Holdouts are rated by the subjects and so have nonmissing values in the dependent variables. Simulation observations are not rated and so have missing values in the dependent variable. It is useful to create a format for the `weight` variable that distinguishes the three types of observations in the input and output data sets.

```
proc format;
  value wf 1 = 'Active'
        . = 'Holdout'
        0 = 'Simulation';
run;
```

PROC TRANSREG does not distinguish between weights that are zero, missing, or negative. All non-positive weights exclude the observations from the analysis. The holdout and simulation observations are given different nonpositive values and a format to make them easy to distinguish in subsequent analyses and listings. The part-worth utilities for each attribute are computed using only those observations with positive weight. The predicted utility is computed for all products, even those with nonpositive weights.

Monotone, Spline, and Monotone Spline Comparisons

When you choose the transformation of the ratings or rankings, you choose among

`identity` - model the data directly

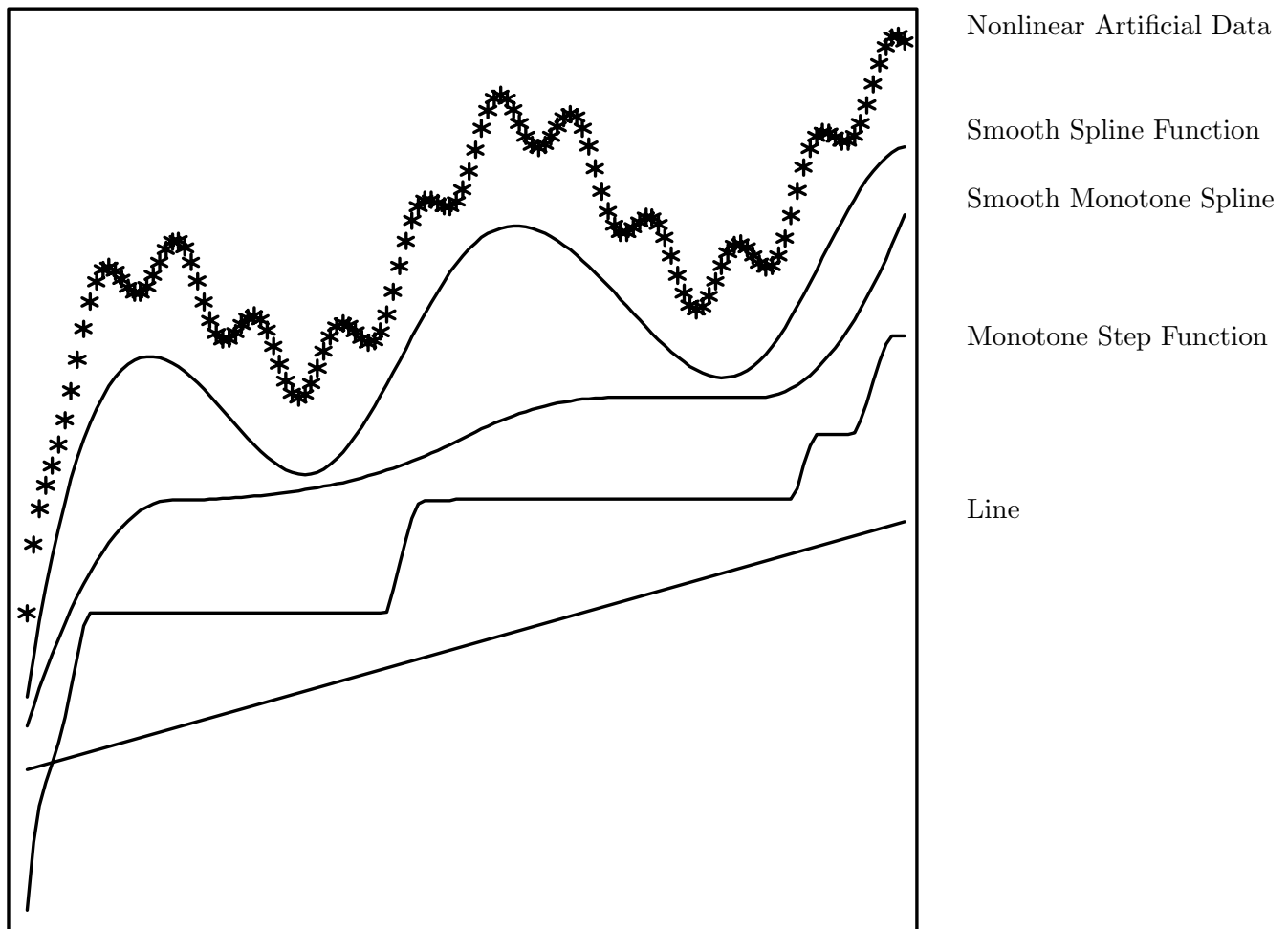
`monotone` - model an increasing step function of the data

`mspline` - model a nonlinear but smooth and increasing function of the data

`spline` - model a smooth function of the data

The following plot shows examples of the different types of functions you can fit in PROC TRANSREG. At the top of the plot are some artificial nonlinear data. Below that is a spline function, created by `spline`. It is smooth and nonlinear. It follows the overall shape of the data, but smoothes out the smaller bumps. Below that is a monotone spline function, created by `mspline`. Like the spline function, it is smooth and nonlinear. Unlike the spline function, it is monotonic. The function never decreases; it always rises or stays flat. The monotone spline function follows the overall upward trend in the data, and it shows the changes in upward trend, but it smoothes out all the dips and bumps in the function. Below the monotone spline function is a monotone step function, created by `monotone`. It is not smooth, but it is monotonic. Like the monotone spline, the monotone step function follows the

Functions Available in PROC TRANSREG



overall upward trend in the data, and it smoothes out all the dips and bumps in the function. However, the function is not smooth, and it typically requires many more parameters be fit than with monotone splines. Below the monotone step function is a line, created by `identity`. It is smooth and linear. It follows the overall upward trend in the data, but it smoothes over all the dips, bumps, and changes in upward trend.

Typical conjoint analyses are metric (using `identity`) or nonmetric (using `monotone`). While not often used in practice, monotone splines have a lot to recommend them. They allow for nonlinearities in the transformation of preference, but unlike `monotone`, they are smooth and do not use up all of your error *df*. One would typically never use `spline` on the ratings or rankings in a conjoint analysis, but if for some reason, you had a lot of price points,[¶] you could fit a spline function of the price attribute. This would allow for nonlinearities in preferences for different prices while constraining the part-worth utility function to be smooth.

[¶]For design efficiency reasons, you typically should not.

Samples of PROC TRANSREG Usage

Conjoint analysis can be performed in many ways with PROC TRANSREG. This section provides sample specifications for some typical and some more esoteric conjoint analyses. The dependent variables typically contain ratings or rankings of products by a number of subjects. The independent variables, `x1-x5`, are the attributes. For metric conjoint analysis, the dependent variable is designated `identity`. For nonmetric conjoint analysis, `monotone` is used. Attributes are usually designated as `class` variables with the restriction that the part-worth utilities within each attribute sum to zero.

The `utilities` option requests an overall ANOVA table, a table of part-worth utilities, their standard errors, and the importance of each attribute. The `p` (predicted values) option outputs to a data set the predicted utility for each product. The `ireplace` option suppresses the separate output of transformed independent variables since the independent variable transformations are the same as the raw independent variables. The `weight` variable is used to distinguish active observations from holdouts and simulation observations. The `reflect` transformation option reflects the transformation of the ranking so that large transformed values, positive utility, and positive evaluation will all correspond.

Today, metric conjoint analysis is used more often than nonmetric conjoint analysis, and rating-scale data are collected more often than rankings.

Metric Conjoint Analysis with Rating-Scale Data

This is a metric conjoint analysis with rating-scale data.

```
ods exclude notes mvanova anova;
proc transreg data=a utilities short method=morals;
  model identity(rating1-rating100) = class(x1-x5 / zero=sum);
  output p ireplace;
  weight w;
run;
```

Nonmetric Conjoint Analysis

This is a nonmetric conjoint analysis specification, which has *many* parameters for the transformations.

```
ods exclude notes anova liberalanova conservanova
  mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
  model monotone(ranking1-ranking100 / reflect) = class(x1-x5 / zero=sum);
  output p ireplace;
  weight w;
run;
```

Monotone Splines

This is a conjoint analysis that is more restrictive than a nonmetric analysis but less restrictive than a metric conjoint analysis. By default, the monotone spline transformation has two parameters (degree two with no knots).

```
ods exclude notes anova liberalanova conservanova
      mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
  model mspline(ranking1-ranking100 / reflect) =
    class(x1-x5 / zero=sum);
  output p ireplace;
  weight w;
run;
```

If less smoothness is desired, specify knots. For example:

```
ods exclude notes anova liberalanova conservanova
      mvanova liberalmvanova conservmvanova;
proc transreg data=a utilities short maxiter=500 method=morals;
  model mspline(ranking1-ranking100 / reflect nknots=3) =
    class(x1-x5 / zero=sum);
  output p ireplace;
  weight w;
run;
```

Each knot requires estimation of an additional parameter.

Constraints on the Utilities

Here is a metric conjoint analysis specification with linearity constraints imposed on `x4` and monotonicity constraints imposed on `x5`.

```
ods exclude notes anova liberalanova conservanova
      mvanova liberalmvanova conservmvanova
      liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
  model identity(rating1-rating100) = class(x1-x3 / zero=sum)
    identity(x4) monotone(x5);
  output p ireplace;
  weight w;
run;
```

With the monotonic constraints on the part-worth utilities, PROC TRANSREG prints some extra information, liberal and conservative part-worth utility and fit statistics tables. These tables report the same part-worth utilities, but are based on different methods of counting the number of parameters estimated. The liberal test tables can be suppressed by adding `liberalutilities liberalfitstatistics` to the `ods exclude` statement.

Here is another example, specifying monotonic step-function constraints on `x1-x5` and a smooth, monotonic transformation of price:

```
ods exclude notes anova liberalanova conservanova
      mvanova liberalmvanova conservmvanova
      liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
  model identity(rating1-rating100) = monotone(x1-x5) mspline(price);
  output p ireplace;
  weight w;
run;
```

A Discontinuous Price Function

The utility of price may not be a continuous function of price. It has been frequently found that utility is discontinuous at *round* numbers such as \$1.00, \$2.00, \$100, \$1000, and so on. If `price` has many values in the data set, say over the range \$1.05 to \$3.95, then a monotone function of price with discontinuities at \$2.00 and \$3.00 can be requested as follows.

```
ods exclude notes anova liberalanova conservanova
      mvanova liberalmvanova conservmvanova
      liberalutilities liberalfitstatistics;
proc transreg data=a utilities short maxiter=500 method=morals;
  model identity(rating1-rating100) =
    class(x1-x5 / zero=sum)
    mspline(price / knots=2 2 2 3 3 3);
  output p ireplace;
  weight w;
run;
```

The monotone spline is degree two. The *order* of the spline is one greater than the degree; in this case the order is three. When the same knot value is specified *order* times, the transformation is discontinuous at the knot. Refer to page 785, for some applications of splines to conjoint analysis.