

# Full Profile--Conjoint Analysis and Experimental Design Final Project-VBA

By Alex Sakaguchi and Brent Taylor

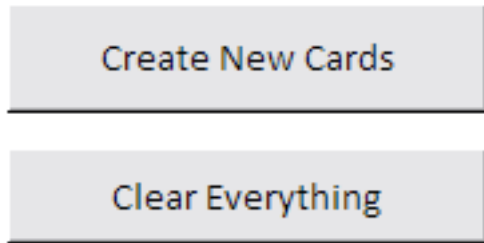
## Executive Summary

One of the most important and useful tools in all of marketing research is the ability to conduct studies using conjoint analysis. Full profile conjoint analysis is a way of using experimental design and full/partial factorials to come up with a set of cards that represents a given product's features and levels (profiles). A person will go through and rank these cards in order of preference, from which the researcher can later derive representational utility for each combination of features and levels. It is not hard to see that as you include more features or levels per feature the more possible combinations there are and the less feasible it becomes to have a person go through and rank the cards. To compensate for this, experimental design narrows the total number of possibilities (factorials) that would be on each card to a manageable set of cards between 8 and 16 in most cases. The model is limited in that the more you use fractional factorials, the less your predictive capability becomes. For example, if you have a product that has four features with four levels per feature the total number of possible combinations is 256 ( $4 \times 4 \times 4 \times 4$ ). The model will use partial factorials to narrow the required number of cards to 16 and gives the researcher predictive capability to estimate utility levels for any combination—even one that has not necessarily been represented on the cards.

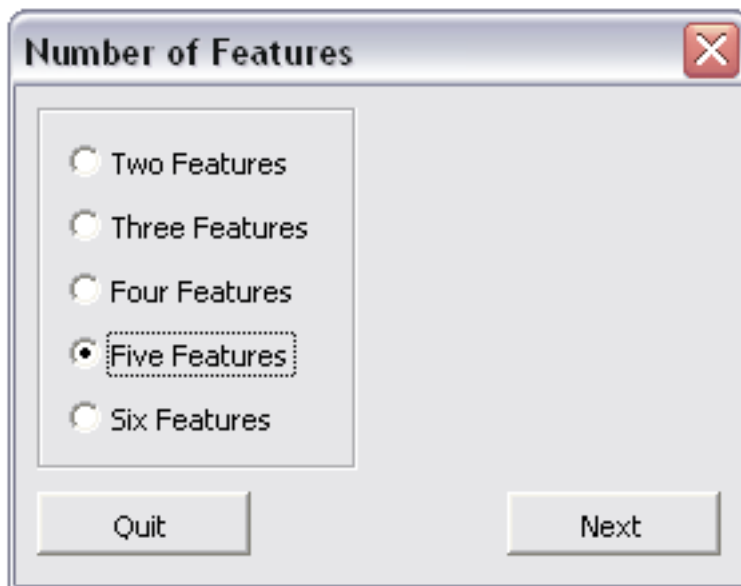
The demand for marketers that are familiar with conjoint analysis is consistently high. Software designed to conduct the necessary computations and algorithms to accurately conduct conjoint studies sells for many thousands of dollars—thus, making it a very expensive tool. This past winter semester, Dr. Smith introduced us to a basic DOS based tool that assisted in setting up full-profile conjoint analysis. He mentioned that the tool was over 20 years old, yet due to the lack of “affordable” alternatives, it was all we could use. Due to our marketing ambitions and desire to learn how to conduct conjoint studies, we thought to write a solution for the problem using VBA. In continued research on the scope of this task, we were disappointed to learn that solutions currently available to help a researcher through the experimental design approach were in excess of 10,000 lines of code. Due to the sheer size of this undertaking, we have simplified the model so that instead of outputting utility levels with empirical support, we can simply use the cards to determine functional direction a firm should undertake when considering a new product offering, change to an existing product, or other line/brand extension. Dr. Smith, owner of Qualtrics, has already communicated interest in our research.

## Implementation Discussion

- When a marketing manager wishes to test for viability and direction a firm should take when considering a new product offering, change to an existing product, or other line/brand extension, he will simply open the excel sheet and click on “Create New Cards”.



- From there, the user will be presented with a form where they can choose the number of product features the test will assume. To make the project more feasible, we have limited the number of features a product can have to 6.

A standard Windows-style dialog box with a title bar that says "Number of Features" and a red close button (X) in the top right corner. The main area contains a list of five radio button options: "Two Features", "Three Features", "Four Features", "Five Features", and "Six Features". The "Five Features" option is selected, indicated by a black dot inside the radio button and a dashed rectangular border around the text. At the bottom of the dialog, there are two buttons: "Quit" on the left and "Next" on the right.

- Once the number of features is decided, the user is presented with a form where they will name each individual feature. These names will be used dynamically on the next user form, as well as on the final deliverable for the user. It should be noted, an error trap will not allow the user to use a zero length string for the name of a feature

**Features' Names**

Feature 1:

Feature 2:

Feature 3:

Feature 4:

Feature 5:

- Next, the user will be asked to input the name of each level per feature. We have limited the number of levels to two levels per feature in order to make the task a little less complicated.
- It's important to note that the total maximum possibilities allowed in this model is  $2^6$  or  $2 \times 2 \times 2 \times 2 \times 2 \times 2$  which equals 64 combinations.
- Typically there is a rank order to the features. That is to say, one level is inherently preferred over the other (hence the naming of 'high' and 'low' features). However, this does not always have to be the case. It may be that something like color is simply a preference. This does not have any negative impact on the analysis of the researcher conducting the conjoint. Rather, the extent to which these 'lateral' moves are used appropriately, it can add depth and understanding.

**Feature High Low Names**

	Low	High
Engine Size	<input type="text" value="4 cyl"/>	<input type="text" value="6 cyl"/>
MPG	<input type="text" value="20 mpg"/>	<input type="text" value="25 mpg"/>
Make	<input type="text" value="Deawo"/>	<input type="text" value="Kia"/>
Interior	<input type="text" value="..."/>	<input type="text"/>
Color	<input type="text"/>	<input type="text"/>

- Once the user has successfully navigated his way through selecting product features and levels, the program is now ready to generate the cards that can be given to participants to rank. This represents the most difficult part of the project. Below is a screen shot of the code that assigns the product feature level in order to accurately represent each an equal number of times.

16	7	0						
2	2	1	1	2	0	2		
0	0	0	0	0	0	2		
1	0	0	1	2	1	1		
0	3	1	0	2	1	3		
1	3	1	1	0	0	0		
1	2	0	0	3	0	3		
3	2	1	0	0	1	1		
0	2	0	1	1	1	0		
1	1	1	0	1	1	2		
2	1	0	1	0	1	3		
3	0	1	1	1	0	3		
2	0	1	0	3	1	0		
3	1	0	0	2	0	0		
2	3	0	0	1	0	1		
0	1	1	1	3	0	1		
3	3	0	1	3	1	2		

The columns represent the levels in each feature. Note column 1—this feature, which happens to be price, includes 4 levels: \$3k, \$5k, \$15k, \$30k. Each one is assigned a value, 0, 1, 2, and 3 respectively. The rows are the profiles generated through experimental design.

Dr. Scott Smith provided this data from an actual study conducted.

- Additionally, we encountered the following problems, and overcame them with the subsequent solution:
  - Problem: Although we can use counters to be sure our random generation of cards is equally weighted, we soon realized there was nothing to prevent the exact same card from being randomly generated twice. This was especially a problem with 3 features since there are only 8 possible combinations and we are using 8 cards. There is only a 0.24% chance that all 8 cards would be unique in that case.
    - Solution: We added functionality that compares each card to every other card, looking for matches. We loop until nothing matches. Initially we were concerned that perhaps this would consume a large amount of processing time, but that was not the case.
  - Problem: Solution to the problem above created another problem. If there are only 2 features selected, there are only 4 possible combinations. With 8 cards, each solution needs to be represented exactly twice.
    - Solution: We built in functionality that checks the first four cards against each other, and the last four cards against each other if there are only 2 features.
  - Problem: If a user begins inputting information but decides to quit, other procedures called from our control sub still tried to execute.
    - Solution: If the user ever decides to quit, we added a global Boolean variable that indicates whether or not the program should continue running after the user forms are finished.
- In order to keep the program understandable, we decided to use a 'control' procedure that functions mostly to call other procedures as needed. A screen shot of this control procedure is included here

```

Sub ControlProcedure()

    ShouldWeContinue = True
    FeaturesForm.Show
    If ShouldWeContinue = False Then Exit Sub

    Do
        PickFeaturesForCards           'generates random cards
        CheckDupCards                 'checks to make sure random cards aren't duplicates
    Loop Until RedoCards = False

    TextualizeCards
    WriteOutCards
    CleanOutForms
    RandomizeCardsToSheet2
    FormatToPrint

    Sheet2.Activate
    MsgBox "Your cards have been generated in an random order." & Chr(10) & _
        "Feel free to print, cut, and use the cards as shown here."

End Sub

```

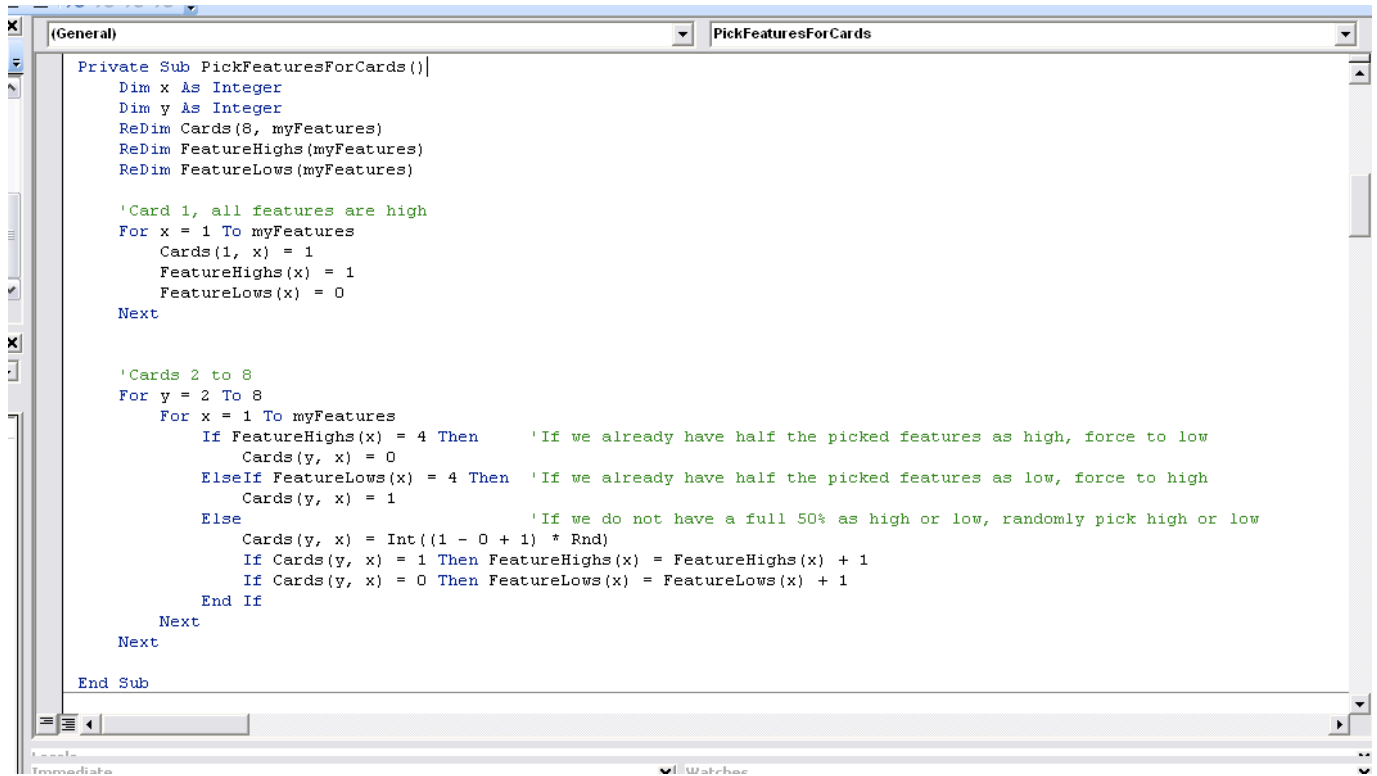
---

## Learning Discussion

First of all, one big learning point is that there is a reason why there are no simple solutions to conduct conjoint analysis and that those that are available are certainly justified in their price points. This project was not easy. Now that it works, though in a simplified manner, we believe we've been able to create a very useful solution for firms that might be considering new product offerings and would like to test the "validity" of direction moving forward. Though exact utility levels per feature and level is beyond the scope of this assignment, the tool does provide a starting point for marketing managers to determine whether or not they should move forward in a particular direction. The object of this type of automation is to make processes more simple and consistent. This model does exactly that. The cards it produces are valid and consistent with experimental design and are presented in a format that is ready to print.

Working with so many small pieces of data, this particular project greatly enhanced our ability use loops and arrays effectively. While there are some places we would have liked to use them more, a very large part of the code is looping similar lines of code over and over. Without loops, coding this would not have been an enjoyable experience.

Code that we are proud of:



```
Private Sub PickFeaturesForCards()  
    Dim x As Integer  
    Dim y As Integer  
    ReDim Cards(8, myFeatures)  
    ReDim FeatureHighs(myFeatures)  
    ReDim FeatureLows(myFeatures)  
  
    'Card 1, all features are high  
    For x = 1 To myFeatures  
        Cards(1, x) = 1  
        FeatureHighs(x) = 1  
        FeatureLows(x) = 0  
    Next  
  
    'Cards 2 to 8  
    For y = 2 To 8  
        For x = 1 To myFeatures  
            If FeatureHighs(x) = 4 Then 'If we already have half the picked features as high, force to low  
                Cards(y, x) = 0  
            ElseIf FeatureLows(x) = 4 Then 'If we already have half the picked features as low, force to high  
                Cards(y, x) = 1  
            Else 'If we do not have a full 50% as high or low, randomly pick high or low  
                Cards(y, x) = Int((1 - 0 + 1) * Rnd)  
                If Cards(y, x) = 1 Then FeatureHighs(x) = FeatureHighs(x) + 1  
                If Cards(y, x) = 0 Then FeatureLows(x) = FeatureLows(x) + 1  
            End If  
        Next  
    Next  
  
End Sub
```

```
Private Sub TextualizeCards()  
    ReDim TextCards(8, myFeatures)  
    Dim x As Integer  
    Dim y As Integer  
    Dim z As Integer  
  
    For z = 1 To myFeatures 'z is for features  
        For y = 1 To 8 'y is for cards  
            For x = 0 To 1 'x is for high/low  
                If Cards(y, z) = x Then TextCards(y, z) = myFeaturesHighLow(z, x)  
                If Cards(y, z) = x Then TextCards(y, z) = myFeaturesHighLow(z, x)  
            Next  
        Next  
    Next  
  
End Sub
```

```

Private Sub RandomizeCardsToSheet2 ()
    Dim MoveCards(8)
    Dim UsedY(8) As Boolean
    Dim x As Integer
    Dim y As Integer

    For x = 1 To 8
        Do
            y = Int((8 - 1 + 1) * Rnd + 1)
            If UsedY(y) = False Then
                UsedY(y) = False
            Else
                UsedY(y) = True
            End If
        Loop Until UsedY(y) = False
        MoveCards(y) = Sheet1.Range(Sheet1.Cells(2, x), Sheet1.Cells(7, x)).Value
        UsedY(y) = True
    Next

    Sheet2.Range(Sheet2.Cells(2, 1), Sheet2.Cells(7, 1)).Value = MoveCards(1)
    Sheet2.Range(Sheet2.Cells(2, 3), Sheet2.Cells(7, 3)).Value = MoveCards(2)
    Sheet2.Range(Sheet2.Cells(2, 5), Sheet2.Cells(7, 5)).Value = MoveCards(3)
    Sheet2.Range(Sheet2.Cells(2, 7), Sheet2.Cells(7, 7)).Value = MoveCards(4)
    Sheet2.Range(Sheet2.Cells(11, 1), Sheet2.Cells(16, 1)).Value = MoveCards(5)
    Sheet2.Range(Sheet2.Cells(11, 3), Sheet2.Cells(16, 3)).Value = MoveCards(6)
    Sheet2.Range(Sheet2.Cells(11, 5), Sheet2.Cells(16, 5)).Value = MoveCards(7)
    Sheet2.Range(Sheet2.Cells(11, 7), Sheet2.Cells(16, 7)).Value = MoveCards(8)

```