Table of Contents

[References 1](#_Toc120256579)

[Conjoint Analysis Revisited 1](#_Toc120256580)

[Possible Extensions to the Model 5](#_Toc120256581)

## References

<https://www.linkedin.com/pulse/conjoint-analysis-simple-python-implementation-prajwal-sreenivas>

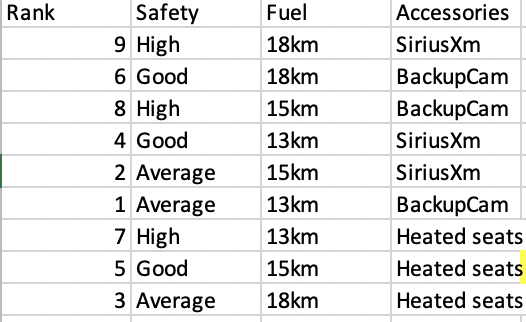
<https://www.informit.com/articles/article.aspx?p=2350028>

## Conjoint Analysis Revisited

Earlier we discussed how to build and evaluate individual preferences with a conjoint analysis survey.

Example 1: Calculating Level Part-Worths and Attribute Importances

This example shows how to take survey results and generate level part-worths and attribute importances from rankings in a CSV file ('CarRanking\_train.csv'):



The calculations are stored in a JSON object.

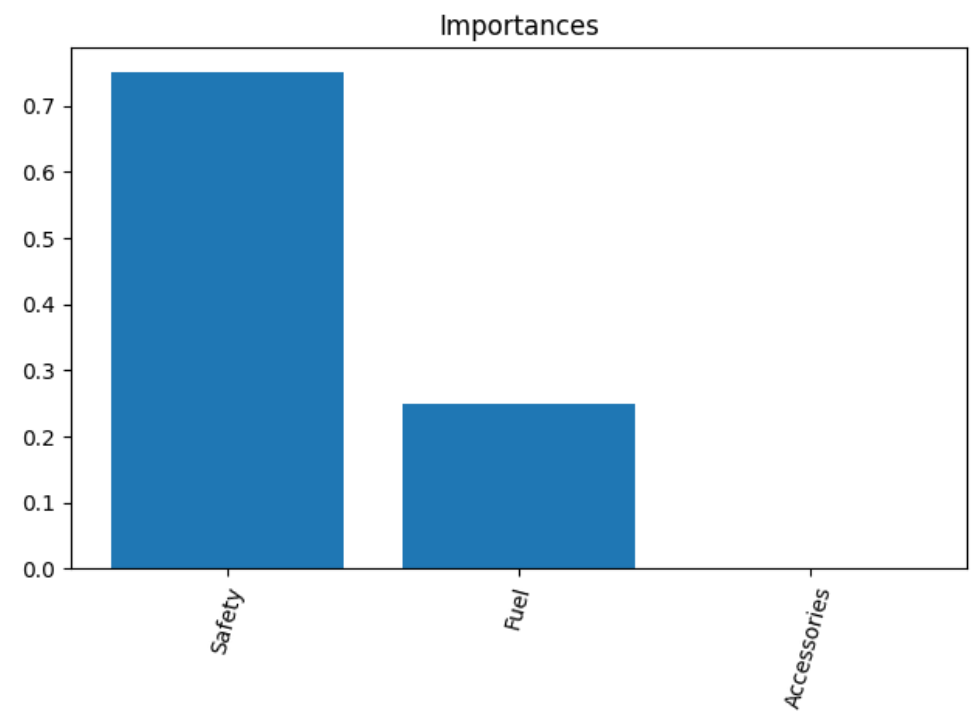
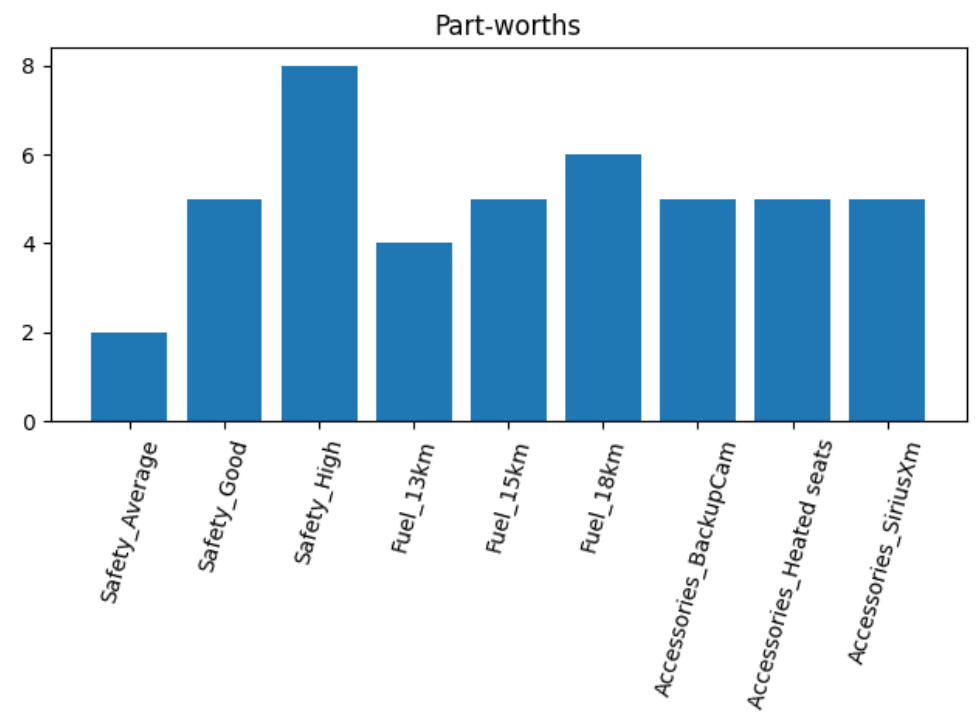
|  |
| --- |
| [{'attribute': 'Safety', 'min': 2.0, 'max': 8.0, 'range': 6.0,  'partWorths': [{'Average': 2.0}, {'Good': 5.0}, {'High': 8.0}],  'importance': 0.75},  {'attribute': 'Fuel', 'min': 4.0, 'max': 6.0, 'range': 2.0,  'partWorths': [{'13km': 4.0}, {'15km': 5.0}, {'18km': 6.0}],  'importance': 0.25},  {'attribute': 'Accessories', 'min': 5.0, 'max': 5.0, 'range': 0.0,  'partWorths': [{'BackupCam': 5.0}, {'Heated seats': 5.0}, {'SiriusXm': 5.0}],  'importance': 0.0}] |

Here is the code:

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  PATH = "/Users/pm/Desktop/DayDocs/data/"  df = pd.read\_csv('/Users/pm/Downloads/CarRanking\_train.csv')  pd.set\_option('display.max\_columns', None)  pd.set\_option('display.width', 1000)  print(df)  def getSummary(df, attribute):  averagesSeries = df.groupby([attribute])['Rank'].mean()  attributeDict = {}  min = averagesSeries.min()  max = averagesSeries.max()  attributeDict['attribute'] = attribute  attributeDict['min'] = min  attributeDict['max'] = max  attributeDict['range'] = max - min  averagesDf = averagesSeries.to\_frame()  levels = list(averagesDf.index)  levelPartWorths = []  for i in range(0, len(levels)):  averagePartWorth = averagesSeries[i]  levelName = levels[i]  levelPartWorths.append({levelName:averagePartWorth})  attributeDict['partWorths'] = levelPartWorths  return attributeDict  def getImportances(attributeSummaries):  ranges = []  for i in range(0, len(attributeSummaries)):  ranges.append(attributeSummaries[i]['range'])  rangeSum = sum(ranges)  for i in range(0, len(attributeSummaries)):  importance = attributeSummaries[i]['range']/rangeSum  attributeSummaries[i]['importance'] = importance  return attributeSummaries  attributeNames = ['Safety','Fuel','Accessories']  attributeSummaries = []  for i in range(0, len(attributeNames)):  attributeInfo = getSummary(df, attributeNames[i])  attributeSummaries.append(attributeInfo)  attributeSummaries = getImportances(attributeSummaries)  print(attributeSummaries) |

Example 2: Plotting Level Part-Worths and Attribute Importances

This example shows how to plot level part-worths and importances.

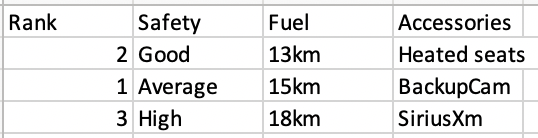
Here we can see how each attribute (safety, fuel economy and accessories) rank with each other in terms of importance. The relative level part-worths for each feature are displayed in the plot on the right.

To complete this solution, add this code to the end of Example 1:

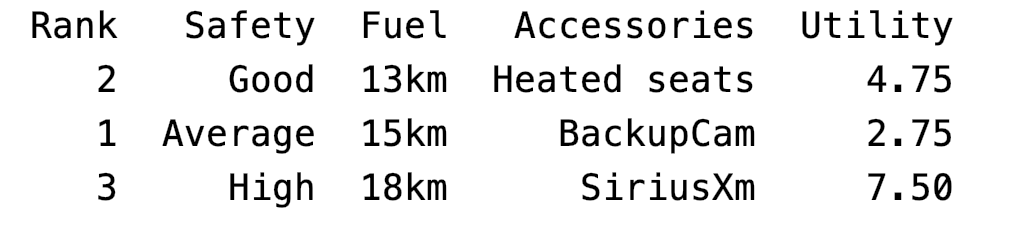
|  |
| --- |
| def plotImportances(attributeSummaries):  X = []  y = []  for i in range(0, len(attributeSummaries)):  X.append(attributeSummaries[i]['attribute'])  y.append(attributeSummaries[i]['importance'])  plt.bar(X, y)  plt.title("Importances")  plt.xticks(rotation=75)  plt.show()  def plotLevels(attributeSummaries):  X = []  y = []  for i in range(0, len(attributeSummaries)):  attribute = attributeSummaries[i]['attribute']  partWorths = attributeSummaries[i]['partWorths']  for j in range(0, len(partWorths)):  obj = partWorths[j]  key = list(obj.keys())[0]  val = list(obj.values())[0]  label = attribute + "\_" + key  X.append(label)  y.append(val)  plt.bar(X, y)  plt.title("Part-worths")  plt.xticks(rotation=75)  plt.show()  plotImportances(attributeSummaries)  plotLevels(attributeSummaries) |

Example 3: Predicting Utility for Different Options

These are new options. The first two in this new list were not presented in the original survey.



Based on an understanding of attribute importance and level worth for each attribute, the model can predict the respondent’s ranking for each car model by utility level.



To complete this solution, add this code to the end of Example 2:

|  |
| --- |
| dfTest = pd.read\_csv('/Users/pm/Downloads/CarRanking\_test.csv')  utilities = []  for i in range(0, len(dfTest)):  utilitySum = 0  for j in range(0, len(attributeNames)):  attribute = attributeNames[j]  level = dfTest.iloc[i][attribute]  utility = getUtility(attributeSummaries, attribute, level)  utilitySum += utility  utilities.append(utilitySum)  dfTest['Utility'] = utilities  print(dfTest) |

Exercise 1 (4 marks)

Edit **CarRanking\_train.csv** so the rankings reflect your preference. Then save the file. Show a screenshot of the edited spreadsheet table here:

|  |
| --- |
|  |

Next, edit **CarRanking\_test.csv** so the rankings reflect your preference. Then save the file. Show a screenshot of the edited spreadsheet table here:

|  |
| --- |
|  |

Next run the code. Show a screenshot of the ranking and calculated utility (quantified preference) for each car option.

|  |
| --- |
|  |

How well did the utility calculation do for ranking your preferred car model where largest utility is for the highest preferred model?

|  |
| --- |
| Nope! Largest utility is not the top preferred model…. ☹ |

## Possible Extensions to the Model

If many people are surveyed, each person’s utility for each feature such as 18KM/L could be scaled relative to the sum of all of their feature utilities. Then people’s relative utility for each feature could be compared by demographic. In this case, a car manufacturer could determine which car options would satisfy each demographic and which car options would be most popular overall.