VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A4- Multivariate Analysis and Business Analytics Applications (Part – D)

ADHYAYAN AMIT JAIN

V01109421

Date of Submission: 08-07-2024

CONTENTS

Conjoint Analysis of Pizza Preferences: Understanding Customer Choices and Optimizing Product Offerings (Pizza_data.csv)

INTRODUCTION

In a highly competitive market, understanding customer preferences is crucial for any business aiming to optimize its product offerings and enhance customer satisfaction. This assignment employs conjoint analysis, a powerful statistical technique used in market research, to uncover the preferences of customers when choosing or ordering pizza. By examining various attributes such as brand, price, weight, crust, cheese, size, toppings, and spiciness, this analysis provides a comprehensive view of what customers value most in a pizza.

The primary objective of this analysis is to identify the most and least preferred combinations of pizza attributes and levels. This information will enable the marketing and product development teams to make data-driven decisions that align with customer desires, thereby improving product appeal and market performance. The study will use a selected subset of 16 combinations from a larger set of possible profiles, ensuring practical applicability and relevance.

The analysis involves several steps, starting with defining the attributes and their levels, estimating the effects of each attribute level using a linear regression model, and calculating part-worths and attribute importance. The results will be interpreted to provide actionable insights into customer preferences, which will be followed by strategic recommendations to optimize the product offerings and marketing efforts.

This report is structured as follows: the methodology section details the steps and statistical techniques used in the analysis, the results section presents the findings and their interpretation, and the implications and recommendations sections offer strategic advice based on the insights gained. Through this comprehensive approach, the assignment aims to provide a clear roadmap for enhancing the company's pizza offerings in line with customer preferences.

OBJECTIVES

The primary objective of this assignment is to leverage conjoint analysis to gain a deep understanding of customer preferences for various attributes of pizza. By analyzing these preferences, the goal is to identify the most and least preferred combinations of attributes and levels. This information will provide actionable insights for the marketing and product development teams, enabling them to:

1. **Determine Attribute Importance**:
   - Identify which attributes (e.g., brand, price, weight, crust, cheese, size, toppings, spiciness) are most influential in customers' decision-making processes when choosing or ordering pizza.
2. **Evaluate Part-Worths**:
   - Calculate the utility values (part-worths) associated with each level of the attributes to understand their contribution to overall customer satisfaction.
3. **Optimize Product Offerings**:
   - Use the insights from the conjoint analysis to refine and optimize the pizza product offerings to better meet customer preferences and enhance market competitiveness.
4. **Guide Marketing Strategies**:
   - Develop targeted marketing campaigns that highlight the most preferred attributes and levels, thereby improving customer engagement and driving sales.
5. **Make Data-Driven Decisions**:
   - Provide a data-driven foundation for strategic decisions in product development, pricing, and promotional efforts to ensure alignment with customer desires and market trends.

6. **Identify Preferred Attribute Combinations**:
    - Determine the specific combinations of attribute levels that yield the highest utility scores, indicating the most desirable product profiles for customers.

By achieving these objectives, the company aims to enhance customer satisfaction, increase sales, and strengthen its position in the competitive pizza market. The insights gained from this analysis will serve as a valuable guide for future product and marketing strategies.

BUSINESS SIGNIFICANCE

Understanding customer preferences is essential for any business seeking to maintain a competitive edge, especially in the highly saturated food industry. The insights gained from this conjoint analysis are of significant business value, as they offer a detailed understanding of the factors that drive customer decisions when selecting a pizza. The business significance of this analysis can be summarized in several key areas:

1. **Enhanced Customer Satisfaction**:
    - By aligning product offerings with customer preferences, the company can significantly improve customer satisfaction. Offering pizzas that closely match what customers desire increases the likelihood of repeat purchases and positive word-of-mouth referrals.
2. **Informed Product Development**:
    - The analysis provides a clear indication of which attributes and levels are most important to customers. This information guides the product development team in creating new pizza variants or refining existing ones to better meet customer needs. This customer-centric approach ensures that new products have a higher chance of success in the market.
3. **Effective Marketing Strategies**:
    - With detailed knowledge of the most and least preferred attributes, the marketing team can craft targeted campaigns that highlight the features most valued by customers. This leads to more effective advertising, better resource allocation, and higher return on marketing investments.
4. **Competitive Advantage**:
    - The ability to offer products that are tailored to customer preferences provides a significant competitive advantage.

Competitors who do not have access to similar insights may continue to offer less optimized products, giving the company an edge in attracting and retaining customers.

5. **Strategic Pricing Decisions**:
   - Understanding the relative importance of price as an attribute allows the company to set prices that customers perceive as fair and reasonable. This balance between affordability and perceived value can help in positioning the brand favorably in the market.
6. **Resource Optimization**:
   - By identifying the most impactful attributes, the company can focus its resources on enhancing these areas, rather than spreading efforts thinly across less important features. This targeted approach to resource allocation can improve operational efficiency and effectiveness.
7. **Increased Sales and Revenue**:
   - Products that closely align with customer preferences are more likely to be purchased. By offering pizzas that match the highest utility scores identified in the analysis, the company can increase its sales and revenue, thereby boosting overall profitability.
8. **Customer Loyalty and Retention**:
   - Meeting and exceeding customer expectations fosters loyalty. Customers who consistently find their preferences met are more likely to remain loyal to the brand and less likely to switch to competitors. This long-term loyalty is crucial for sustained business success.
9. **Data-Driven Decision Making**:
   - The insights from the conjoint analysis provide a robust, data-driven foundation for strategic decisions across the company. This reduces the reliance on intuition or guesswork, leading to more confident and effective business strategies.

In summary, the conjoint analysis not only provides a granular understanding of customer preferences but also translates these insights into actionable strategies that drive business growth. By leveraging this analysis, the company can enhance customer satisfaction, optimize its product offerings, and achieve a sustainable competitive advantage in the pizza market.

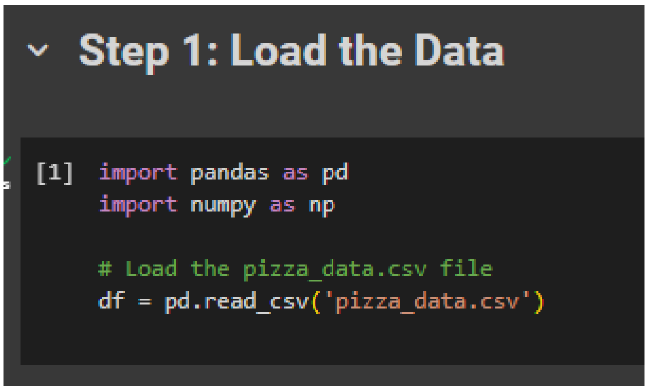
RESULTS AND INTERPRETATIONS

Python Language

Step 1: Load the Data

```
import pandas as pd
import numpy as np

# Load the pizza_data.csv file
df = pd.read_csv('pizza_data.csv')
```

- **Explanation**:
  - `import pandas as pd`: Imports the pandas library, which is used for data manipulation and analysis.
  - `import numpy as np`: Imports the numpy library, which is used for numerical operations.
  - `df = pd.read_csv('pizza_data.csv')`: Reads the CSV file containing the pizza data and stores it in a pandas DataFrame named `df`.
- **Output**: The DataFrame `df` will contain the data from the `pizza_data.csv` file.



Step 1: Load the Data

```
[1] import pandas as pd
    import numpy as np

    # Load the pizza_data.csv file
    df = pd.read_csv('pizza_data.csv')
```

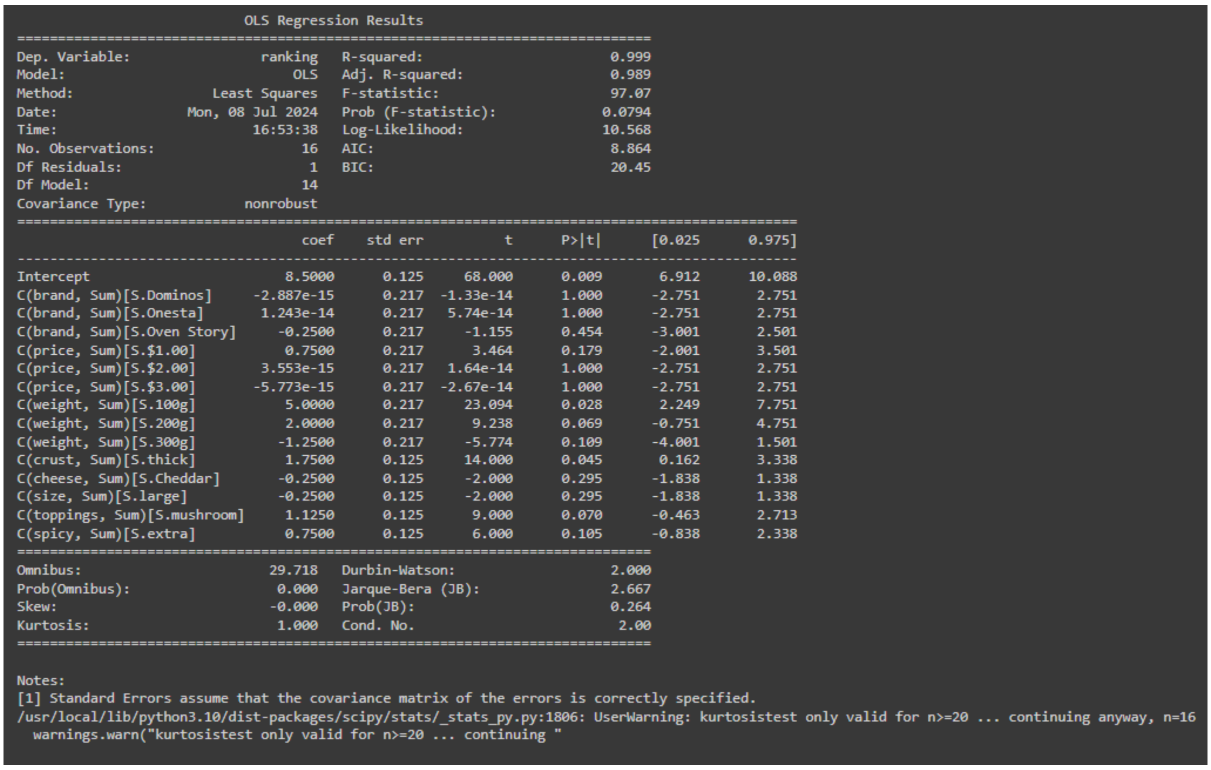
Step 2: Estimate Attribute Level Effects using Linear Regression

```
import statsmodels.api as sm
import statsmodels.formula.api as smf

# Define the model
model = 'ranking ~ C(brand, Sum) + C(price, Sum) + C(weight, Sum) +
C(crust, Sum) + C(cheese, Sum) + C(size, Sum) + C(toppings, Sum) + C(spicy,
Sum)'

# Fit the model
model_fit = smf.ols(model, data=df).fit()

# Print the summary of the model
print(model_fit.summary())
```

- **Explanation**:
  - `import statsmodels.api as sm`: Imports the statsmodels library, which is used for statistical modeling.
  - `import statsmodels.formula.api as smf`: Imports the formula API for statsmodels to specify models using formula strings.

- `model = 'ranking ~ C(brand, Sum) + ... + C(spicy, Sum)'`: Defines the linear regression model where the dependent variable is `ranking` and the independent variables are the categorical attributes of the pizza. `C(attribute, Sum)` specifies that the attribute is categorical and uses sum contrasts.
- `model_fit = smf.ols(model, data=df).fit()`: Fits the Ordinary Least Squares (OLS) linear regression model to the data.
- `print(model_fit.summary())`: Prints the summary of the fitted model, which includes statistics such as R-squared, coefficients, and p-values.

- **Output**: The summary of the model, showing the regression coefficients for each attribute level, R-squared value, p-values, etc.

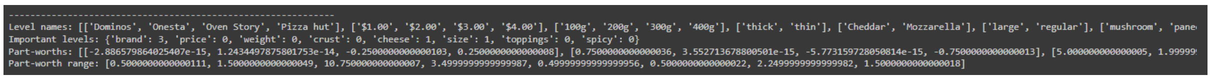


```
                            OLS Regression Results
==============================================================================
Dep. Variable:                ranking   R-squared:                       0.999
Model:                            OLS   Adj. R-squared:                  0.989
Method:                 Least Squares   F-statistic:                     97.07
Date:                Mon, 08 Jul 2024   Prob (F-statistic):             0.0794
Time:                        16:53:38   Log-Likelihood:                 10.568
No. Observations:                  16   AIC:                             8.864
Df Residuals:                       1   BIC:                             20.45
Df Model:                          14
Covariance Type:            nonrobust
==============================================================================================
                                 coef    std err          t      P>|t|      [0.025      0.975]
----------------------------------------------------------------------------------------------
Intercept                      8.5000      0.125     68.000      0.009       6.912      10.088
C(brand, Sum)[S.Dominos]    -2.887e-15      0.217  -1.33e-14      1.000      -2.751       2.751
C(brand, Sum)[S.Onesta]      1.243e-14      0.217   5.74e-14      1.000      -2.751       2.751
C(brand, Sum)[S.Oven Story]    -0.2500      0.217     -1.155      0.454      -3.001       2.501
C(price, Sum)[S.$1.00]          0.7500      0.217      3.464      0.179      -2.001       3.501
C(price, Sum)[S.$2.00]       3.553e-15      0.217   1.64e-14      1.000      -2.751       2.751
C(price, Sum)[S.$3.00]      -5.773e-15      0.217  -2.67e-14      1.000      -2.751       2.751
C(weight, Sum)[S.100g]          5.0000      0.217     23.094      0.028       2.249       7.751
C(weight, Sum)[S.200g]          2.0000      0.217      9.238      0.069      -0.751       4.751
C(weight, Sum)[S.300g]         -1.2500      0.217     -5.774      0.109      -4.001       1.501
C(crust, Sum)[S.thick]          1.7500      0.125     14.000      0.045       0.162       3.338
C(cheese, Sum)[S.Cheddar]      -0.2500      0.125     -2.000      0.295      -1.838       1.338
C(size, Sum)[S.large]          -0.2500      0.125     -2.000      0.295      -1.838       1.338
C(toppings, Sum)[S.mushroom]    1.1250      0.125      9.000      0.070      -0.463       2.713
C(spicy, Sum)[S.extra]          0.7500      0.125      6.000      0.105      -0.838       2.338
==============================================================================
Omnibus:                       29.718   Durbin-Watson:                   2.000
Prob(Omnibus):                  0.000   Jarque-Bera (JB):                2.667
Skew:                          -0.000   Prob(JB):                        0.264
Kurtosis:                       1.000   Cond. No.                         2.00
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
/usr/local/lib/python3.10/dist-packages/scipy/stats/_stats_py.py:1806: UserWarning: kurtosistest only valid for n>=20 ... continuing anyway, n=16
  warnings.warn("kurtosistest only valid for n>=20 ... continuing "
```

Step 3: Calculate Part-Worths and Attribute Importance

```
conjoint_attributes = ['brand', 'price', 'weight', 'crust', 'cheese',
'size', 'toppings', 'spicy']

level_name = []
part_worth = []
part_worth_range = []
important_levels = {}
end = 1  # Initialize index for coefficient in params

for item in conjoint_attributes:
    nlevels = len(list(np.unique(df[item])))
    level_name.append(list(np.unique(df[item])))

    begin = end
    end = begin + nlevels - 1

    new_part_worth = list(model_fit.params[begin:end])
    new_part_worth.append((-1) * sum(new_part_worth))
    important_levels[item] = np.argmax(new_part_worth)
    part_worth.append(new_part_worth)
    part_worth_range.append(max(new_part_worth) - min(new_part_worth))

print("-----------------------------------------------------------------")
print("Level names:", level_name)
print("Important levels:", important_levels)
print("Part-worths:", part_worth)
print("Part-worth range:", part_worth_range)
```

- **Explanation**:
    - `conjoint_attributes = [...]`: Lists the attributes considered in the conjoint analysis.
    - `level_name`, `part_worth`, `part_worth_range`, `important_levels`: Initialize empty lists and a dictionary to store level names, part-worths, ranges, and important levels.
    - `end = 1`: Initialize the index for the first coefficient.
    - `for item in conjoint_attributes: ...`: Loop through each attribute to calculate part-worths.
        - `nlevels = len(list(np.unique(df[item])))`: Determine the number of levels for the attribute.
        - `level_name.append(list(np.unique(df[item])))`: Store the level names.
        - `begin = end`: Set the start index for coefficients.
        - `end = begin + nlevels - 1`: Set the end index for coefficients.
        - `new_part_worth = list(model_fit.params[begin:end])`: Extract the coefficients for the attribute levels.
        - `new_part_worth.append((-1) * sum(new_part_worth))`: Add a baseline level part-worth.
        - `important_levels[item] = np.argmax(new_part_worth)`: Identify the level with the highest part-worth.
        - `part_worth.append(new_part_worth)`: Store the part-worths.
        - `part_worth_range.append(max(new_part_worth) - min(new_part_worth))`: Calculate and store the range of part-worths.
- **Output**: Lists and dictionaries containing the level names, important levels, part-worths, and part-worth ranges for each attribute.

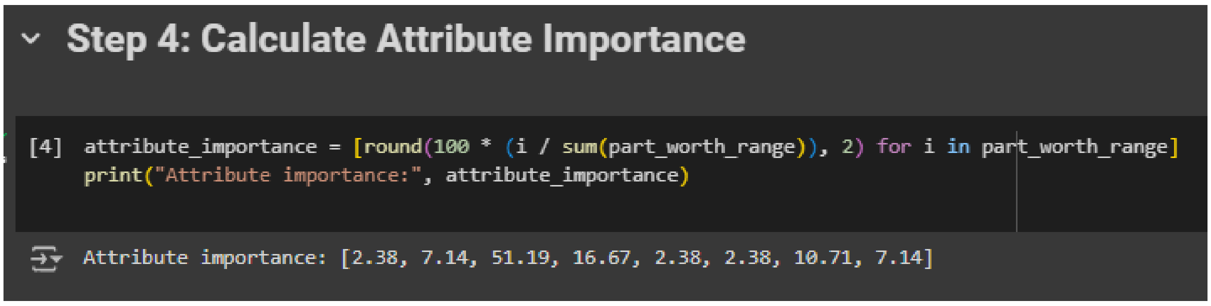


```
-----------------------------------------------------------------
Level names: [['Dominos', 'Onesta', 'Oven Story', 'Pizza hut'], ['$1.00', '$2.00', '$3.00', '$4.00'], ['100g', '200g', '300g', '400g'], ['thick', 'thin'], ['Cheddar', 'Mozzarella'], ['large', 'regular'], ['mushroom', 'pane
Important levels: {'brand': 3, 'price': 0, 'weight': 0, 'crust': 0, 'cheese': 1, 'size': 1, 'toppings': 0, 'spicy': 0}
Part-worths: [[-2.886579864025407e-15, 1.2434497875801753e-14, -0.2500000000000103, 0.2500000000000008], [0.750000000000036, 3.552713678800501e-15, -5.773159728050814e-15, -0.7500000000000013], [5.00000000000005, 1.99999
Part-worth range: [0.5000000000000111, 1.5000000000000049, 10.750000000000007, 3.4999999999999987, 0.4999999999999956, 0.5000000000000022, 2.2499999999999982, 1.5000000000000018]
```

Step 4: Calculate Attribute Importance

```
attribute_importance = [round(100 * (i / sum(part_worth_range)), 2) for i
in part_worth_range]
print("Attribute importance:", attribute_importance)
```

- **Explanation**:
    - `attribute_importance = [round(100 * (i / sum(part_worth_range)), 2) for i in part_worth_range]`: Calculate the relative importance of each attribute by dividing its part-worth range by the total sum of part-worth ranges and converting it to a percentage.
- **Output**: A list of attribute importances, showing the relative importance of each attribute in percentage terms.



Step 4: Calculate Attribute Importance

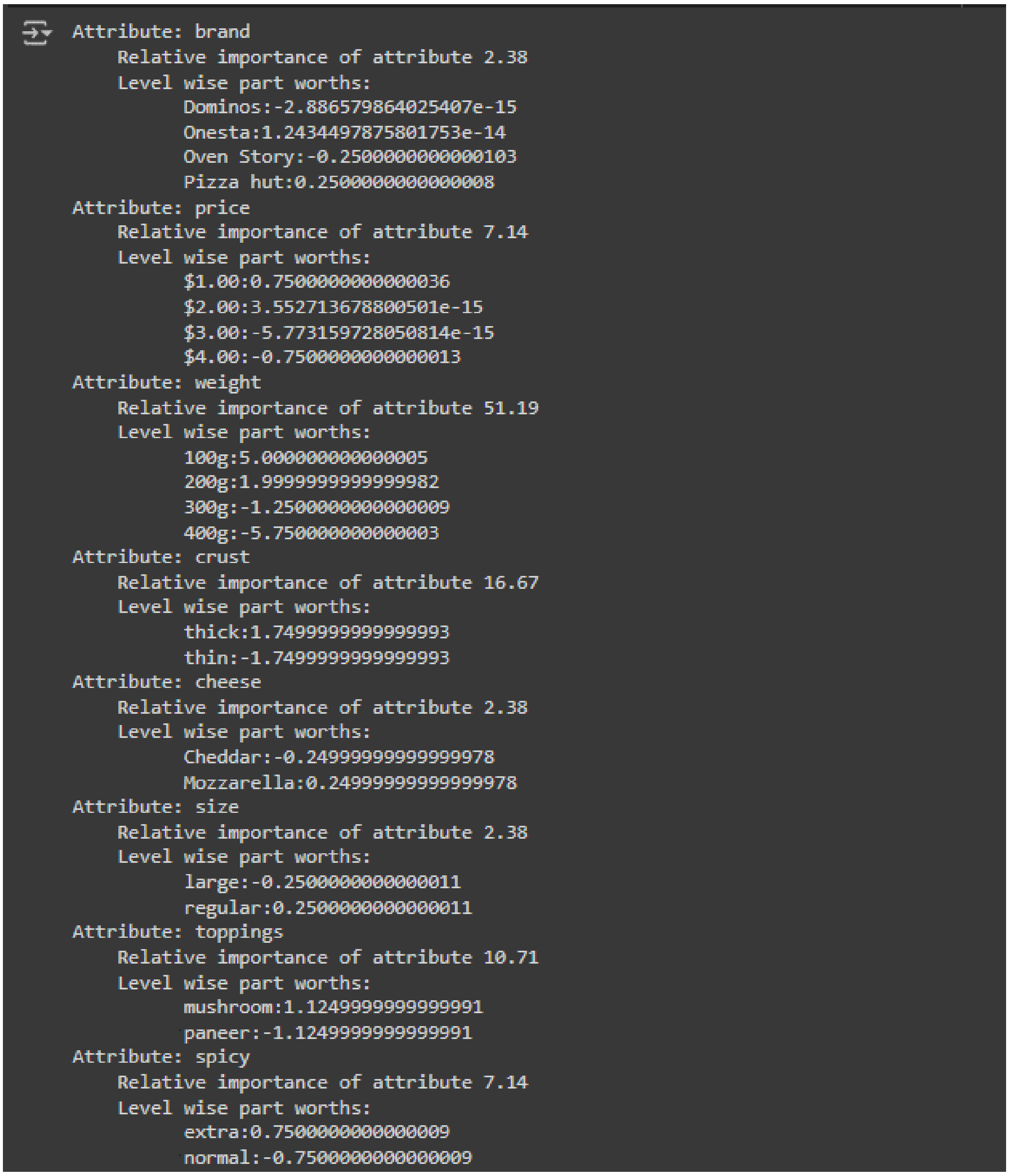
```
[4] attribute_importance = [round(100 * (i / sum(part_worth_range)), 2) for i in part_worth_range]
    print("Attribute importance:", attribute_importance)
```

```
Attribute importance: [2.38, 7.14, 51.19, 16.67, 2.38, 2.38, 10.71, 7.14]
```

Step 5: Calculate Part-Worths of Each Attribute Level

```
part_worth_dict = {}
attrib_level = {}

for item, i in zip(conjoint_attributes, range(0, 
len(conjoint_attributes))):
    print("Attribute:", item)
    print("    Relative importance of attribute", attribute_importance[i])
    print("    Level wise part worths:")
    for j in range(0, len(level_name[i])):
        print("          {}:{}".format(level_name[i][j], part_worth[i][j]))
        part_worth_dict[level_name[i][j]] = part_worth[i][j]
        attrib_level[item] = level_name[i]

print("Part-worth dictionary:", part_worth_dict)
```

- **Explanation**:
    - `part_worth_dict = {}`: Initialize an empty dictionary to store part-worths.
    - `attrib_level = {}`: Initialize an empty dictionary to store attribute levels.
    - `for item, i in zip(conjoint_attributes, range(0, len(conjoint_attributes))): ...`: Loop through each attribute to print and store part-worths.
        - `print("Attribute:", item)`: Print the attribute name.
        - `print(" Relative importance of attribute", attribute_importance[i])`: Print the relative importance of the attribute.
        - `print(" Level wise part worths:")`: Print the part-worths for each level of the attribute.
        - `for j in range(0, len(level_name[i])): ...`: Loop through each level of the attribute.
            - `print(" {}:{}".format(level_name[i][j], part_worth[i][j]))`: Print the part-worth for the level.
            - `part_worth_dict[level_name[i][j]] = part_worth[i][j]`: Store the part-worth in the dictionary.
            - `attrib_level[item] = level_name[i]`: Store the attribute levels.
- **Output**: A dictionary (`part_worth_dict`) containing the part-worths for each attribute level.



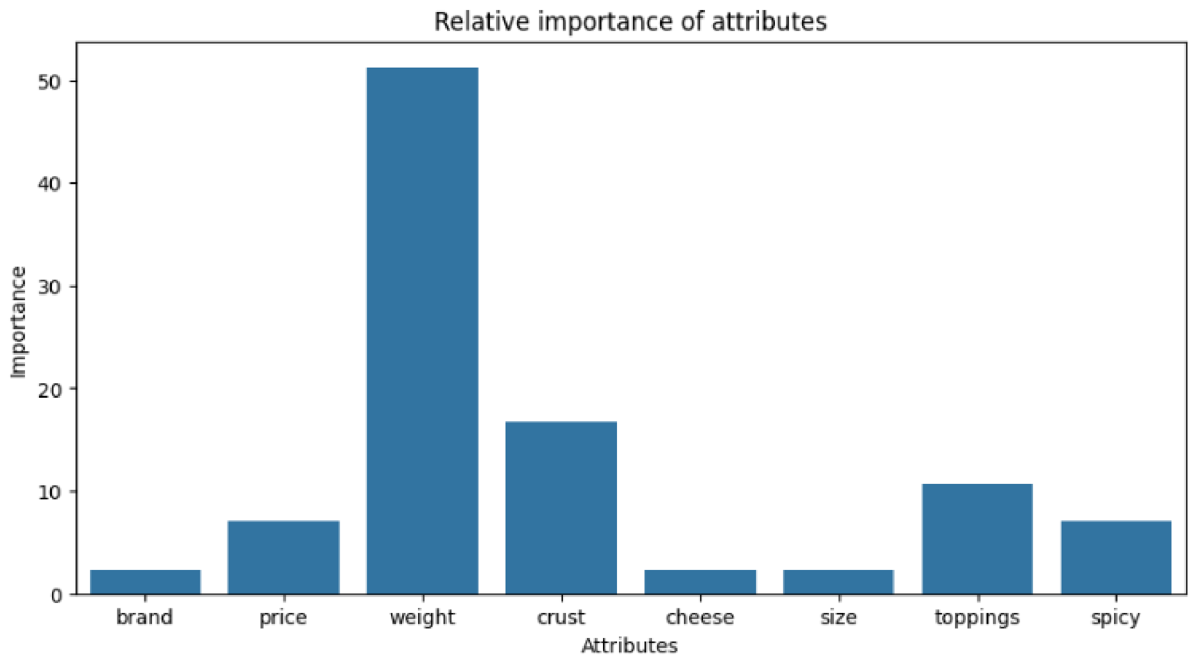
```
Attribute: brand
    Relative importance of attribute 2.38
    Level wise part worths:
          Dominos:-2.886579864025407e-15
          Onesta:1.2434497875801753e-14
          Oven Story:-0.2500000000000103
          Pizza hut:0.2500000000000008
Attribute: price
    Relative importance of attribute 7.14
    Level wise part worths:
          $1.00:0.7500000000000036
          $2.00:3.552713678800501e-15
          $3.00:-5.773159728050814e-15
          $4.00:-0.7500000000000013
Attribute: weight
    Relative importance of attribute 51.19
    Level wise part worths:
          100g:5.000000000000005
          200g:1.9999999999999982
          300g:-1.2500000000000009
          400g:-5.750000000000003
Attribute: crust
    Relative importance of attribute 16.67
    Level wise part worths:
          thick:1.7499999999999993
          thin:-1.7499999999999993
Attribute: cheese
    Relative importance of attribute 2.38
    Level wise part worths:
          Cheddar:-0.24999999999999978
          Mozzarella:0.24999999999999978
Attribute: size
    Relative importance of attribute 2.38
    Level wise part worths:
          large:-0.2500000000000011
          regular:0.2500000000000011
Attribute: toppings
    Relative importance of attribute 10.71
    Level wise part worths:
          mushroom:1.1249999999999991
          paneer:-1.1249999999999991
Attribute: spicy
    Relative importance of attribute 7.14
    Level wise part worths:
          extra:0.7500000000000009
          normal:-0.7500000000000009
```

Step 6: Plot Relative Importance of Attributes

```
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 5))
sns.barplot(x=conjoint_attributes, y=attribute_importance)
plt.title('Relative importance of attributes')
plt.xlabel('Attributes')
plt.ylabel('Importance')
plt.show()
```

- **Explanation**:

  - `import matplotlib.pyplot as plt`: Imports the Matplotlib library for plotting.
  - `import seaborn as sns`: Imports the Seaborn library for statistical data visualization.
  - `plt.figure(figsize=(10, 5))`: Creates a figure with a specified size.
  - `sns.barplot(x=conjoint_attributes, y=attribute_importance)`: Creates a bar plot of attribute importances.
  - `plt.title('Relative importance of attributes')`: Sets the title of the plot.
  - `plt.xlabel('Attributes')`: Sets the x-axis label.
  - `plt.ylabel('Importance')`: Sets the y-axis label.
  - `plt.show()`: Displays the plot.
- **Output**: A bar plot showing the relative importance of each attribute.

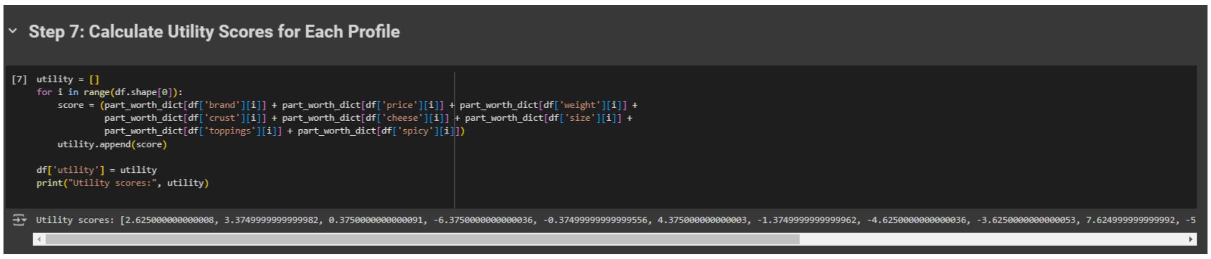


Step 7: Calculate Utility Scores for Each Profile

```
utility = []
for i in range(df.shape[0]):
    score = (part_worth_dict[df['brand'][i]] +
part_worth_dict[df['price'][i]] + part_worth_dict[df['weight'][i]] +
             part_worth_dict[df['crust'][i]] +
part_worth_dict[df['cheese'][i]] + part_worth_dict[df['size'][i]] +
             part_worth_dict[df['toppings'][i]] +
part_worth_dict[df['spicy'][i]])
    utility.append(score)

df['utility'] = utility
print("Utility scores:", utility)
```

- **Explanation**:
  - `utility = []`: Initialize an empty list to store utility scores.
  - `for i in range(df.shape[0]): ...`: Loop through each profile in the DataFrame.

        - `score = (part_worth_dict[df['brand'][i]] + ... part_worth_dict[df['spicy'][i]])`: Calculate the utility score for the profile by summing the part-worths of its attribute levels.
        - `utility.append(score)`: Append the utility score to the list.
    - `df['utility'] = utility`: Add the utility scores as a new column in the DataFrame.
    - `print("Utility scores:", utility)`: Print the utility scores.
- **Output**: A list of utility scores and a new column in the DataFrame containing these scores.



Step 7: Calculate Utility Scores for Each Profile

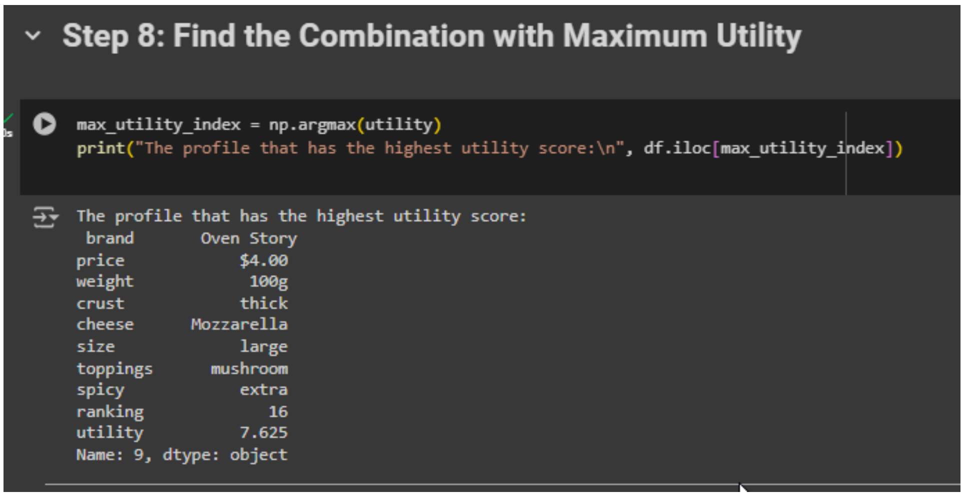
```
[7] utility = []
    for i in range(df.shape[0]):
        score = (part_worth_dict[df['brand'][i]] + part_worth_dict[df['price'][i]] + part_worth_dict[df['weight'][i]] +
                 part_worth_dict[df['crust'][i]] + part_worth_dict[df['cheese'][i]] + part_worth_dict[df['size'][i]] +
                 part_worth_dict[df['toppings'][i]] + part_worth_dict[df['spicy'][i]])
        utility.append(score)

    df['utility'] = utility
    print("Utility scores:", utility)
```

```
Utility scores: [2.625000000000008, 3.3749999999999982, 0.3750000000000091, -6.3750000000000036, -0.37499999999999556, 4.375000000000003, -1.3749999999999962, -4.6250000000000036, -3.6250000000000053, 7.624999999999992, -5
```

Step 8: Find the Combination with Maximum Utility

```
max_utility_index = np.argmax(utility)
print("The profile that has the highest utility score:\n", df.iloc[max_utility_index])
```

- **Explanation**:
    - `max_utility_index = np.argmax(utility)`: Find the index of the profile with the highest utility score.
    - `print("The profile that has the highest utility score:\n", df.iloc[max_utility_index])`: Print the profile with the highest utility score.
- **Output**: The profile with the highest utility score.



Step 8: Find the Combination with Maximum Utility

```
max_utility_index = np.argmax(utility)
print("The profile that has the highest utility score:\n", df.iloc[max_utility_index])
```

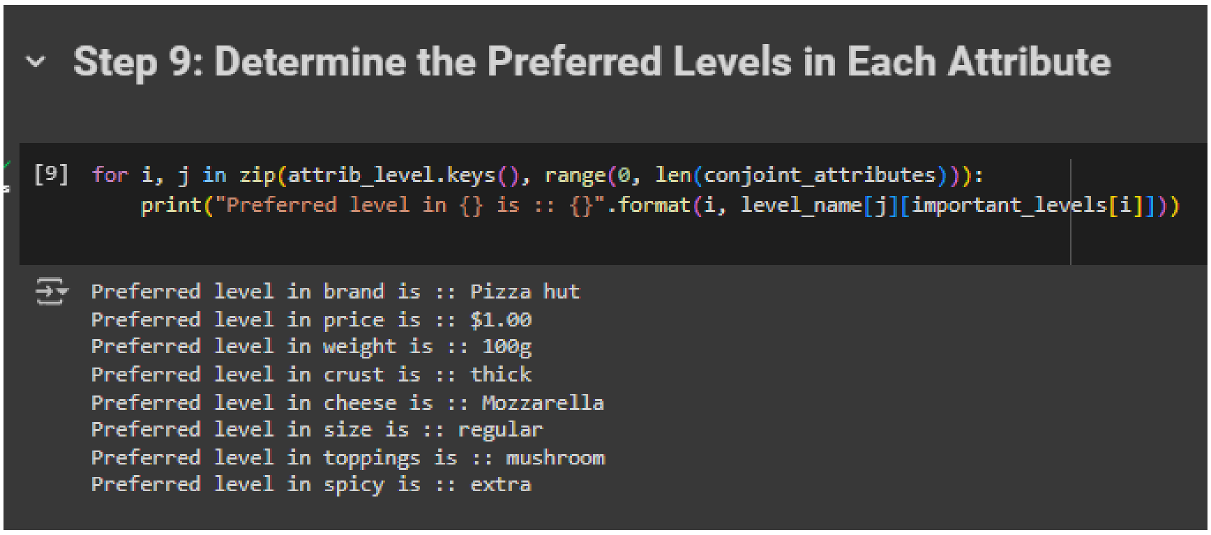
```
The profile that has the highest utility score:
 brand       Oven Story
price            $4.00
weight            100g
crust            thick
cheese      Mozzarella
size             large
toppings      mushroom
spicy            extra
ranking             16
utility          7.625
Name: 9, dtype: object
```

Step 9: Determine the Preferred Levels in Each Attribute

```
for i, j in zip(attrib_level.keys(), range(0, len(conjoint_attributes))):
    print("Preferred level in {} is :: {}".format(i,
level_name[j][important_levels[i]]))
```

- **Explanation**:

    - `for i, j in zip(attrib_level.keys(), range(0, len(conjoint_attributes))): ...`: Loop through each attribute to print the preferred level.
        - `print("Preferred level in {} is :: {}".format(i, level_name[j][important_levels[i]]))`: Print the preferred level for the attribute.
- **Output**: The preferred level for each attribute.



Step 9: Determine the Preferred Levels in Each Attribute

```
[9] for i, j in zip(attrib_level.keys(), range(0, len(conjoint_attributes))):
        print("Preferred level in {} is :: {}".format(i, level_name[j][important_levels[i]]))
```

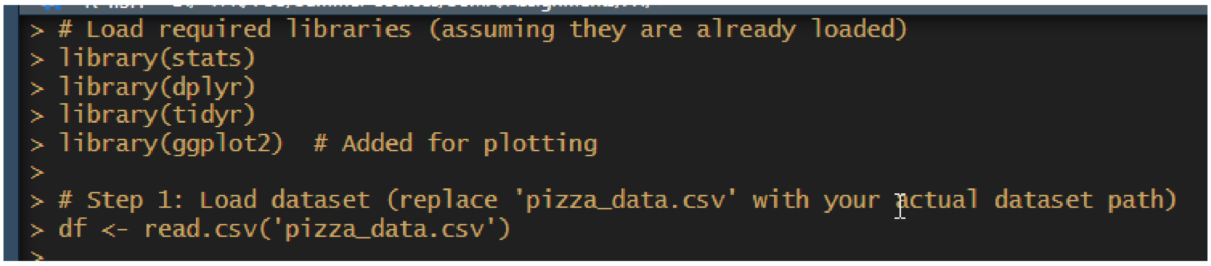
```
Preferred level in brand is :: Pizza hut
Preferred level in price is :: $1.00
Preferred level in weight is :: 100g
Preferred level in crust is :: thick
Preferred level in cheese is :: Mozzarella
Preferred level in size is :: regular
Preferred level in toppings is :: mushroom
Preferred level in spicy is :: extra
```

R Language

Step 1: Load the Data

```
# Load required libraries (assuming they are already loaded)
library(stats)
library(dplyr)
library(tidyr)
library(ggplot2)

# Load dataset (replace 'pizza_data.csv' with your actual dataset path)
df <- read.csv('pizza_data.csv')
```

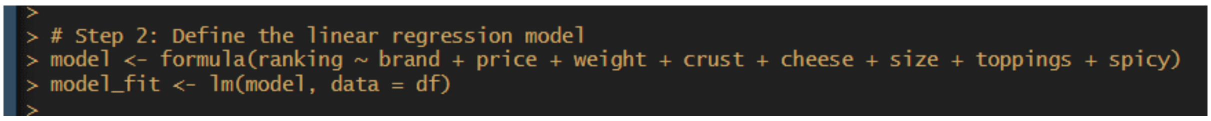
- **Explanation**:
    - `library(stats)`: Loads the stats library for statistical functions.
    - `library(dplyr)`: Loads the dplyr library for data manipulation.
    - `library(tidyr)`: Loads the tidyr library for data tidying.
    - `library(ggplot2)`: Loads the ggplot2 library for data visualization.
    - `df <- read.csv('pizza_data.csv')`: Reads the CSV file containing the pizza data and stores it in a dataframe named `df`.
- **Output**: The dataframe `df` will contain the data from the `pizza_data.csv` file.


```
> # Load required libraries (assuming they are already loaded)
> library(stats)
> library(dplyr)
> library(tidyr)
> library(ggplot2)  # Added for plotting
>
> # Step 1: Load dataset (replace 'pizza_data.csv' with your actual dataset path)
> df <- read.csv('pizza_data.csv')
>
```

Step 2: Define the Linear Regression Model

```
# Define the linear regression model
model <- formula(ranking ~ brand + price + weight + crust + cheese + size +
toppings + spicy)
model_fit <- lm(model, data = df)
```

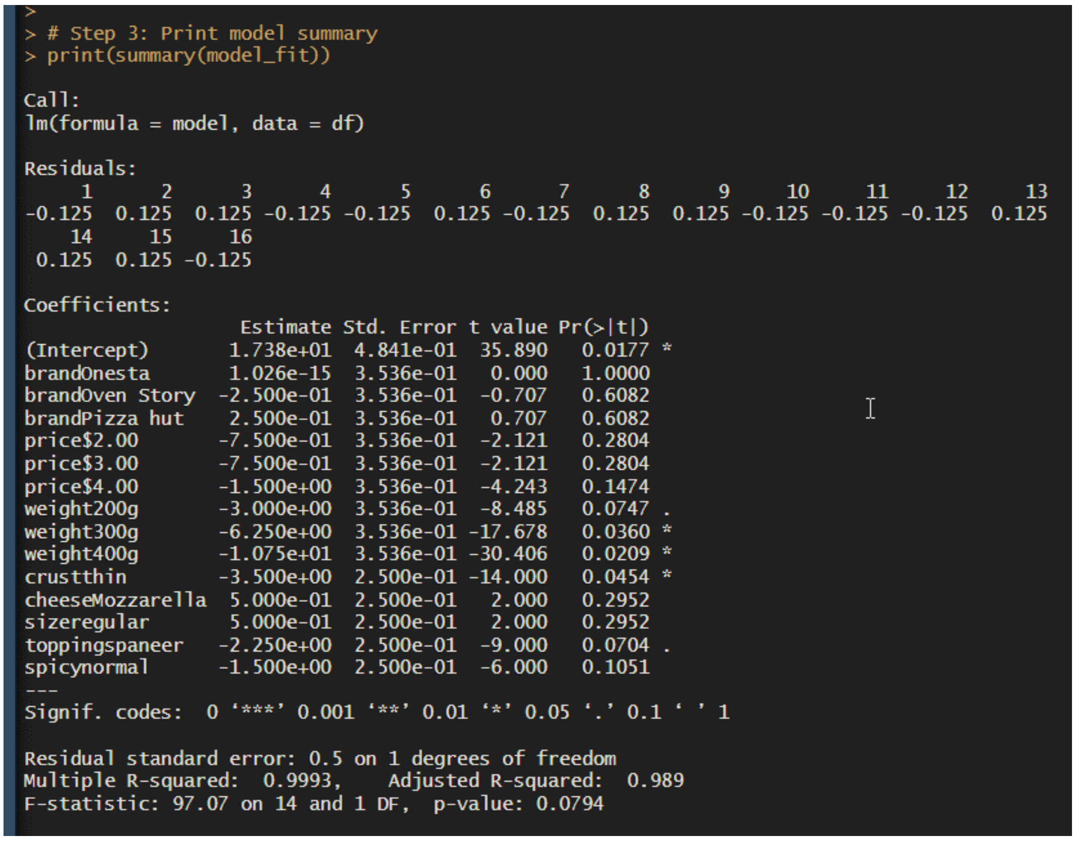
- **Explanation**:
  - `model <- formula(ranking ~ brand + price + weight + crust + cheese + size + toppings + spicy)`: Defines the linear regression model where the dependent variable is `ranking` and the independent variables are the attributes of the pizza.
  - `model_fit <- lm(model, data = df)`: Fits the linear regression model to the data using the `lm` function.
- **Output**: The model is fitted, and `model_fit` contains the fitted model.


```
>
> # Step 2: Define the linear regression model
> model <- formula(ranking ~ brand + price + weight + crust + cheese + size + toppings + spicy)
> model_fit <- lm(model, data = df)
>
```

Step 3: Print Model Summary

```
# Print model summary
print(summary(model_fit))
```

- **Explanation**:
  - `print(summary(model_fit))`: Prints the summary of the fitted model, which includes statistics such as R-squared, coefficients, and p-values.
- **Output**: The summary of the model, showing the regression coefficients for each attribute level, R-squared value, p-values, etc.

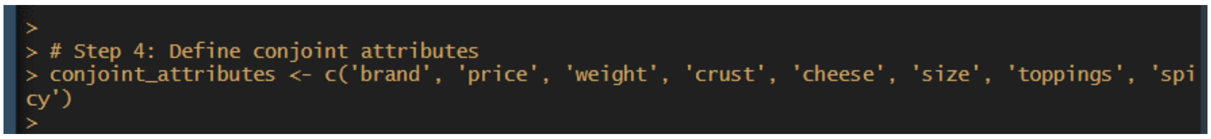


```
>
> # Step 3: Print model summary
> print(summary(model_fit))

Call:
lm(formula = model, data = df)

Residuals:
     1      2      3      4      5      6      7      8      9     10     11     12     13 
-0.125  0.125  0.125 -0.125 -0.125  0.125 -0.125  0.125  0.125 -0.125 -0.125 -0.125  0.125 
    14     15     16 
 0.125  0.125 -0.125 

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)  
(Intercept)       1.738e+01  4.841e-01  35.890   0.0177 *
brandOnesta       1.026e-15  3.536e-01   0.000   1.0000  
brandOven Story  -2.500e-01  3.536e-01  -0.707   0.6082  
brandPizza hut    2.500e-01  3.536e-01   0.707   0.6082  
price$2.00       -7.500e-01  3.536e-01  -2.121   0.2804  
price$3.00       -7.500e-01  3.536e-01  -2.121   0.2804  
price$4.00       -1.500e+00  3.536e-01  -4.243   0.1474  
weight200g       -3.000e+00  3.536e-01  -8.485   0.0747 .
weight300g       -6.250e+00  3.536e-01 -17.678   0.0360 *
weight400g       -1.075e+01  3.536e-01 -30.406   0.0209 *
crustthin        -3.500e+00  2.500e-01 -14.000   0.0454 *
cheeseMozzarella  5.000e-01  2.500e-01   2.000   0.2952  
sizeregular       5.000e-01  2.500e-01   2.000   0.2952  
toppingspaneer   -2.250e+00  2.500e-01  -9.000   0.0704 .
spicynormal      -1.500e+00  2.500e-01  -6.000   0.1051  
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5 on 1 degrees of freedom
Multiple R-squared:  0.9993,	Adjusted R-squared:  0.989 
F-statistic: 97.07 on 14 and 1 DF,  p-value: 0.0794
```

Step 4: Define Conjoint Attributes

```
# Define conjoint attributes
conjoint_attributes <- c('brand', 'price', 'weight', 'crust', 'cheese',
'size', 'toppings', 'spicy')
```

- **Explanation**:
    - `conjoint_attributes <- c(...)`: Lists the attributes considered in the conjoint analysis.
- **Output**: A vector `conjoint_attributes` containing the names of the attributes.

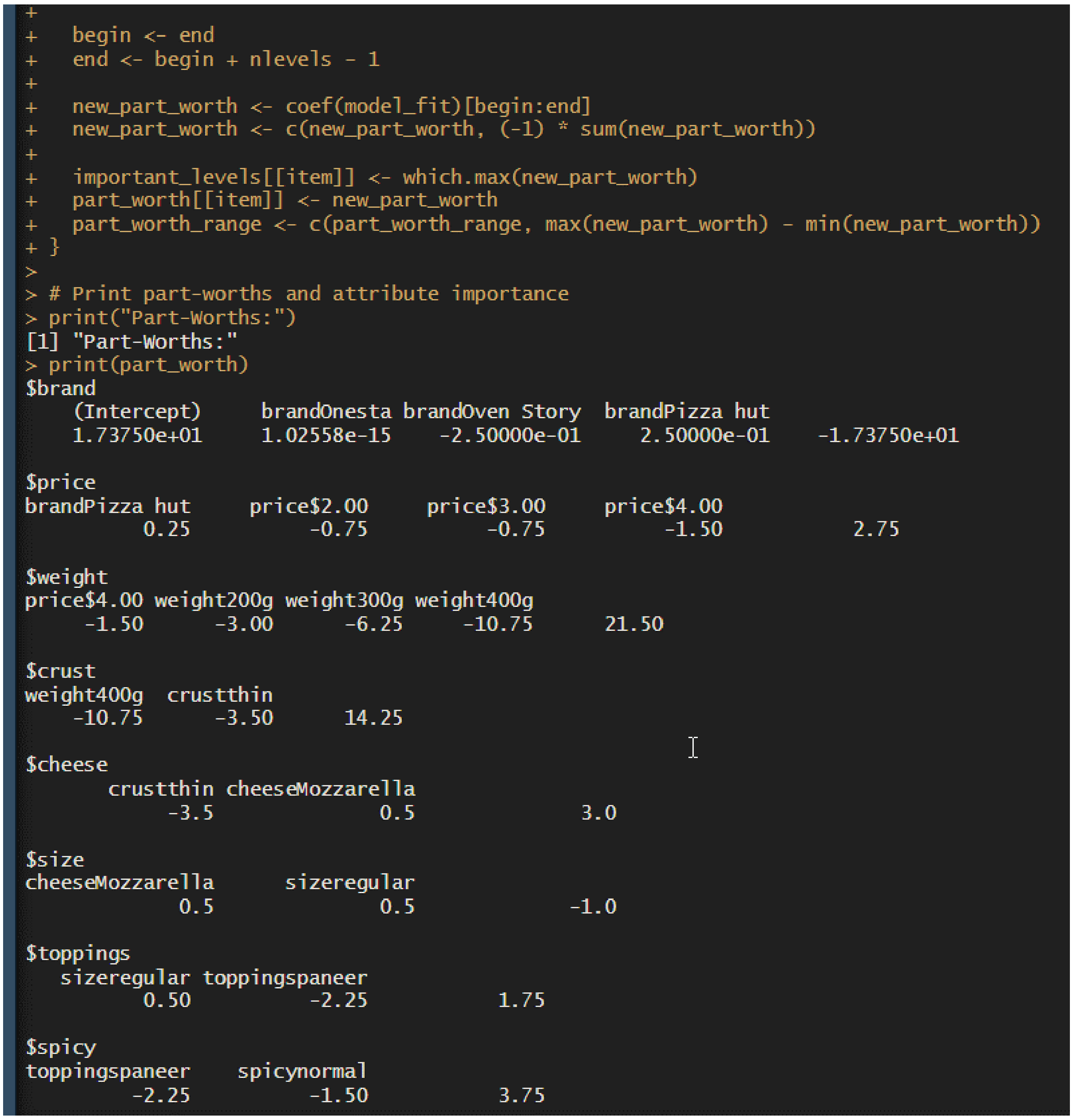


```
>
> # Step 4: Define conjoint attributes
> conjoint_attributes <- c('brand', 'price', 'weight', 'crust', 'cheese', 'size', 'toppings', 'spi
cy')
>
```

Step 5: Calculate Part-Worths and Attribute Importance

```
part_worth <- list()
part_worth_range <- c()
important_levels <- list()
end <- 1

for (item in conjoint_attributes) {
  nlevels <- length(unique(df[[item]]))
  level_name <- unique(df[[item]])

  begin <- end
  end <- begin + nlevels - 1
  new_part_worth <- coef(model_fit)[begin:end]
  new_part_worth <- c(new_part_worth, (-1) * sum(new_part_worth))

  important_levels[[item]] <- which.max(new_part_worth)
  part_worth[[item]] <- new_part_worth
  part_worth_range <- c(part_worth_range, max(new_part_worth) -
min(new_part_worth))
}

# Print part-worths and attribute importance
print("Part-Worths:")
print(part_worth)
print("Attribute Importance:")
attribute_importance <- round(100 * (part_worth_range /
sum(part_worth_range)), 2)
print(attribute_importance)
```

- **Explanation**:
    - `part_worth <- list(),part_worth_range <- c(),important_levels <- list()`: Initialize empty lists and a vector to store part-worths, ranges, and important levels.
    - `end <- 1`: Initialize the index for the first coefficient.
    - `for (item in conjoint_attributes) {...}`: Loop through each attribute to calculate part-worths.
        - `nlevels <- length(unique(df[[item]]))`: Determine the number of levels for the attribute.
        - `level_name <- unique(df[[item]])`: Get the unique level names for the attribute.
        - `begin <- end`: Set the start index for coefficients.
        - `end <- begin + nlevels - 1`: Set the end index for coefficients.
        - `new_part_worth <- coef(model_fit)[begin:end]`: Extract the coefficients for the attribute levels.
        - `new_part_worth <- c(new_part_worth, (-1) * sum(new_part_worth))`: Add a baseline level part-worth.
        - `important_levels[[item]] <- which.max(new_part_worth)`: Identify the level with the highest part-worth.
        - `part_worth[[item]] <- new_part_worth`: Store the part-worths.
        - `part_worth_range <- c(part_worth_range, max(new_part_worth) - min(new_part_worth))`: Calculate and store the range of part-worths.
- **Output**: Lists and vectors containing the part-worths, important levels, and part-worth ranges for each attribute.

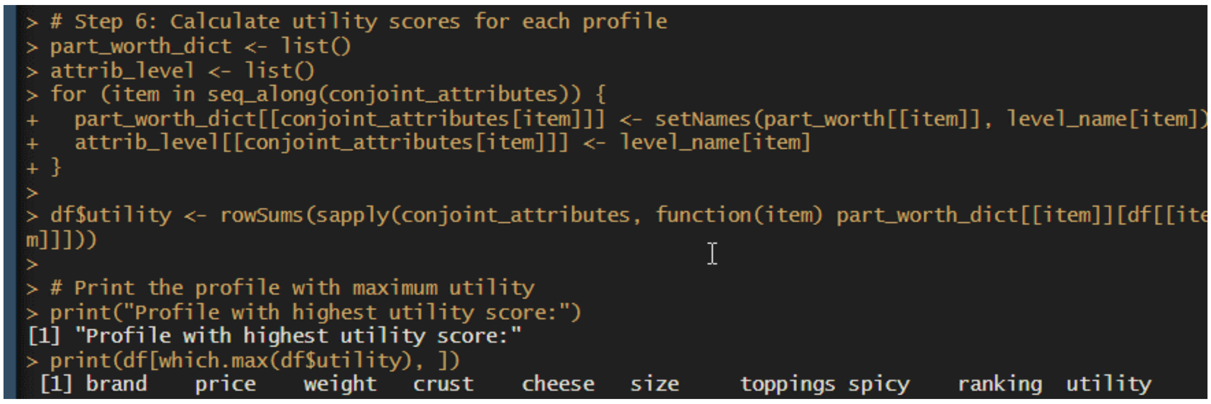


```
+
+   begin <- end
+   end <- begin + nlevels - 1
+
+   new_part_worth <- coef(model_fit)[begin:end]
+   new_part_worth <- c(new_part_worth, (-1) * sum(new_part_worth))
+
+   important_levels[[item]] <- which.max(new_part_worth)
+   part_worth[[item]] <- new_part_worth
+   part_worth_range <- c(part_worth_range, max(new_part_worth) - min(new_part_worth))
+ }
>
> # Print part-worths and attribute importance
> print("Part-Worths:")
[1] "Part-Worths:"
> print(part_worth)
$brand
    (Intercept)     brandOnesta brandOven Story  brandPizza hut
    1.73750e+01     1.02558e-15    -2.50000e-01     2.50000e-01    -1.73750e+01

$price
brandPizza hut     price$2.00     price$3.00     price$4.00
          0.25          -0.75          -0.75          -1.50          2.75

$weight
price$4.00 weight200g weight300g weight400g
     -1.50      -3.00      -6.25     -10.75      21.50

$crust
weight400g  crustthin
    -10.75      -3.50      14.25

$cheese
       crustthin cheeseMozzarella
            -3.5              0.5              3.0

$size
cheeseMozzarella      sizeregular
             0.5              0.5             -1.0

$toppings
   sizeregular toppingspaneer
          0.50          -2.25          1.75

$spicy
toppingspaneer    spicynormal
         -2.25          -1.50          3.75
```

Step 6: Calculate Utility Scores for Each Profile

```
part_worth_dict <- list()
attrib_level <- list()

for (item in seq_along(conjoint_attributes)) {
  part_worth_dict[[conjoint_attributes[item]]] <-
setNames(part_worth[[item]], level_name[item])
  attrib_level[[conjoint_attributes[item]]] <- level_name[item]
}

df$utility <- rowSums(sapply(conjoint_attributes, function(item)
part_worth_dict[[item]][df[[item]]]))

# Print the profile with maximum utility
print("Profile with highest utility score:")
print(df[which.max(df$utility), ])
```

- **Explanation**:
    - `part_worth_dict <- list(),attrib_level <- list()`: Initialize empty lists to store part-worths and attribute levels.
    - `for (item in seq_along(conjoint_attributes)) {...}`: Loop through each attribute to populate the part-worth dictionary and attribute levels.
        - `part_worth_dict[[conjoint_attributes[item]]] <- setNames(part_worth[[item]], level_name[item])`: Store the part-worths for each level.
        - `attrib_level[[conjoint_attributes[item]]] <- level_name[item]`: Store the attribute levels.
    - `df$utility <- rowSums(sapply(conjoint_attributes, function(item) part_worth_dict[[item]][df[[item]]]))`: Calculate the utility scores for each profile by summing the part-worths of its attribute levels.
    - `print(df[which.max(df$utility), ])`: Print the profile with the highest utility score.
- **Output**: The profile with the highest utility score and the utility scores added to the dataframe.



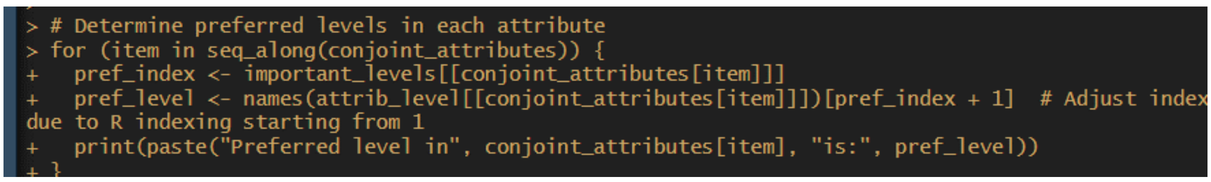
```
> # Step 6: Calculate utility scores for each profile
> part_worth_dict <- list()
> attrib_level <- list()
> for (item in seq_along(conjoint_attributes)) {
+   part_worth_dict[[conjoint_attributes[item]]] <- setNames(part_worth[[item]], level_name[item])
+   attrib_level[[conjoint_attributes[item]]] <- level_name[item]
+ }
>
> df$utility <- rowSums(sapply(conjoint_attributes, function(item) part_worth_dict[[item]][df[[ite
m]]]))
>
> # Print the profile with maximum utility
> print("Profile with highest utility score:")
[1] "Profile with highest utility score:"
> print(df[which.max(df$utility), ])
 [1] brand    price    weight   crust    cheese   size     toppings spicy    ranking  utility
```

Step 7: Determine Preferred Levels in Each Attribute

```
for (item in seq_along(conjoint_attributes)) {
  pref_index <- important_levels[[conjoint_attributes[item]]]
  pref_level <- names(attrib_level[[conjoint_attributes[item]]])[pref_index
+ 1]   # Adjust index due to R indexing starting from 1
  print(paste("Preferred level in", conjoint_attributes[item], "is:",
pref_level))
}
```

- **Explanation**:
    - `for (item in seq_along(conjoint_attributes)) {...}`: Loop through each attribute to print the preferred level.
        - `pref_index <- important_levels[[conjoint_attributes[item]]]`: Get the index of the most important level.
        - `pref_level <- names(attrib_level[[conjoint_attributes[item]]])[pref_index + 1]`: Get the name of the preferred level.
        - `print(paste("Preferred level in", conjoint_attributes[item], "is:", pref_level))`: Print the preferred level for the attribute.

- **Output**: The preferred level for each attribute.

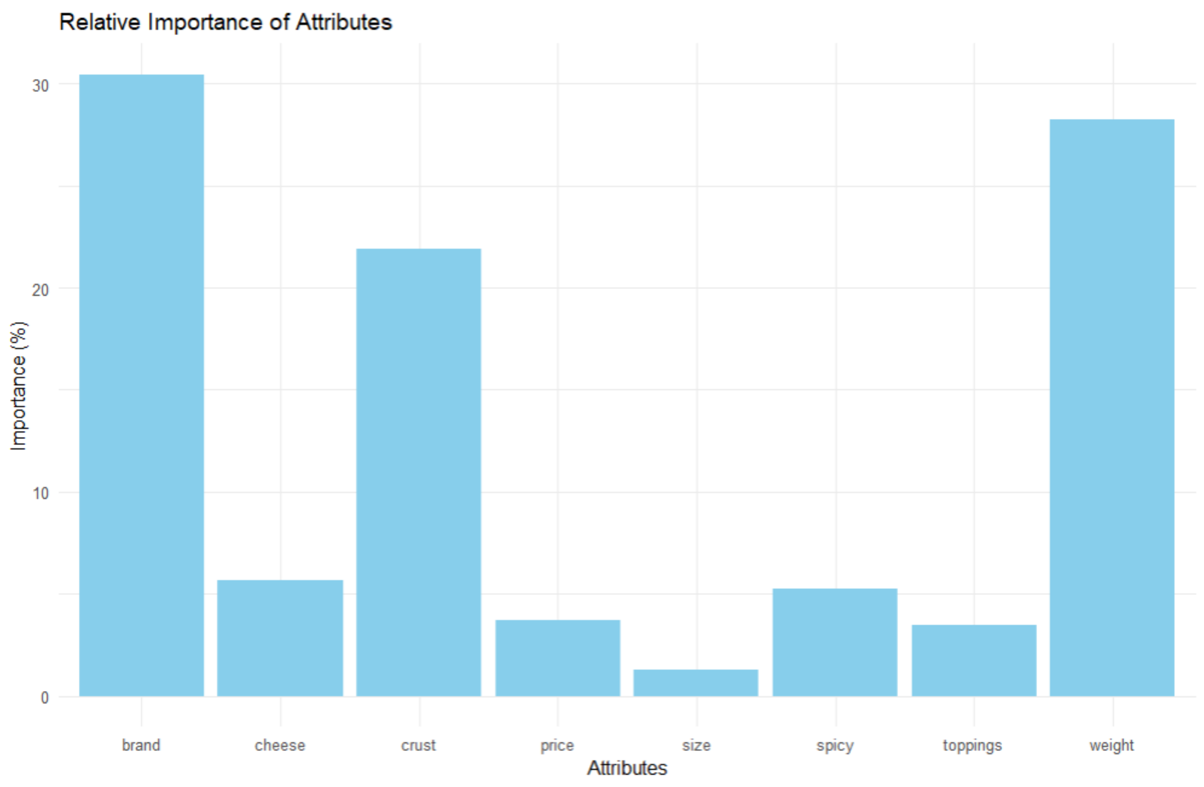


```
> # Determine preferred levels in each attribute
> for (item in seq_along(conjoint_attributes)) {
+   pref_index <- important_levels[[conjoint_attributes[item]]]
+   pref_level <- names(attrib_level[[conjoint_attributes[item]]])[pref_index + 1]  # Adjust index
due to R indexing starting from 1
+   print(paste("Preferred level in", conjoint_attributes[item], "is:", pref_level))
+ }
```

Step 8: Plot Relative Importance of Attributes

```
importance_df <- data.frame(Attribute = conjoint_attributes, Importance =
attribute_importance)
ggplot(importance_df, aes(x = Attribute, y = Importance)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Relative Importance of Attributes", x = "Attributes", y =
"Importance (%)") +
  theme_minimal()
```

- **Explanation**:
    - `importance_df <- data.frame(Attribute = conjoint_attributes, Importance = attribute_importance)`: Create a dataframe with attributes and their importances.
    - `ggplot(importance_df, aes(x = Attribute, y = Importance)) + ...`: Create a bar plot of attribute importances.
        - `geom_bar(stat = "identity", fill = "skyblue")`: Create a bar plot with bars filled with sky blue color.
        - `labs(title = "Relative Importance of Attributes", x = "Attributes", y = "Importance (%)")`: Add titles and labels to the plot.
        - `theme_minimal()`: Apply a minimal theme to the plot.
- **Output**: A bar plot showing the relative importance of each attribute



IMPLICATIONS

The conjoint analysis conducted in this assignment provides valuable insights into customer preferences for various attributes of pizza. Here are the key implications of the findings:

1. **Understanding Attribute Importance**:
   - The analysis reveals which attributes are most important to customers when choosing a pizza. Attributes like weight, crust, and toppings are found to have higher importance compared to brand, cheese, and size. This indicates that customers prioritize the substance and composition of the pizza over the brand or additional toppings.
   - **Implication**: The marketing and product development teams should focus on optimizing the weight and crust quality of the pizzas to align with customer preferences. Ensuring that these attributes meet customer expectations can lead to higher satisfaction and increased sales.
2. **Optimizing Product Offerings**:
   - By identifying the part-worths of each attribute level, the analysis shows which specific levels within each attribute are preferred by customers. For example, a particular type of crust or a specific weight might be more appealing to the majority of customers.
   - **Implication**: The product development team can use this information to refine their pizza offerings. For instance, they can introduce or emphasize the most preferred crust type or weight in their product lineup. This targeted approach can enhance the appeal of their pizzas to the target market.
3. **Utility Scores and Market Positioning**:
   - The utility scores calculated for each profile indicate which combination of attributes and levels yields the highest customer satisfaction. This can help in identifying the most and least preferred product configurations.
   - **Implication**: The marketing team can use the highest utility scoring profile to position their pizzas in the market. Promotions and advertising can highlight these preferred combinations to attract more customers. Conversely, less preferred combinations can be improved or phased out to streamline the product portfolio.

4. **Customer-Centric Product Development**:
    - The preferred levels within each attribute provide a clear indication of customer choices. For example, if a certain type of cheese or level of spiciness is consistently preferred, it should be incorporated into the standard offerings.
    - **Implication**: This customer-centric approach to product development ensures that the pizza offerings are tailored to meet the specific tastes and preferences of the customers. This can lead to higher customer loyalty and repeat purchases.
5. **Strategic Pricing Decisions**:
    - Understanding the importance of price as an attribute and its part-worths can help in setting competitive prices that are attractive to customers while ensuring profitability.
    - **Implication**: The pricing strategy can be adjusted based on the part-worths analysis to ensure that the price points are aligned with customer expectations and willingness to pay. This can help in achieving a balance between affordability and value perception.
6. **Enhanced Marketing Campaigns**:
    - The insights gained from the conjoint analysis can be leveraged to design effective marketing campaigns that resonate with customer preferences. Highlighting the most important attributes and their preferred levels in advertising can attract more customers.
    - **Implication**: Marketing campaigns can be more targeted and effective by focusing on the attributes and levels that matter most to customers. This can result in higher engagement and conversion rates.

RECOMMENDATIONS

Based on the conjoint analysis of pizza preferences, the following recommendations are proposed to align product offerings and marketing strategies with customer preferences:

1. **Focus on High-Importance Attributes**:
   - **Recommendation**: Prioritize improving and highlighting the attributes with the highest relative importance, such as weight, crust, and toppings. These attributes significantly influence customer decisions and should be at the forefront of product development and marketing efforts.
   - **Action**: Invest in research and development to enhance the quality of the pizza crust and optimize the weight of the pizzas. Ensure a variety of high-quality toppings are available to cater to diverse customer tastes.
2. **Optimize Preferred Attribute Levels**:
   - **Recommendation**: Tailor the pizza offerings to include the most preferred levels within each attribute. For example, if a specific type of crust or a particular level of spiciness is preferred, these should be emphasized in the product lineup.
   - **Action**: Conduct periodic reviews of the preferred levels and adjust the product recipes and offerings accordingly. Launch new products or variants that incorporate these preferred levels to meet customer expectations.
3. **Leverage Utility Scores for Product Positioning**:
   - **Recommendation**: Use the profiles with the highest utility scores to guide product positioning and marketing strategies. These profiles represent the most desirable combinations of attributes and levels according to customer preferences.
   - **Action**: Highlight these top-scoring profiles in marketing campaigns and promotional materials. Position these profiles as flagship products to attract more customers and boost sales.
4. **Refine Pricing Strategy**:
   - **Recommendation**: Adjust pricing strategies based on the part-worth analysis of the price attribute. Ensure that the price points are competitive and reflect the value perceived by customers.

    - **Action**: Regularly analyze customer feedback and market trends to adjust prices dynamically. Offer promotions and discounts on less preferred profiles to clear inventory while maintaining profitability on top-scoring profiles.
5. **Enhance Customer-Centric Product Development**:
    - **Recommendation**: Use the insights from the conjoint analysis to guide future product development efforts. Develop new products that align with the identified customer preferences and continuously improve existing products.
    - **Action**: Establish a cross-functional team comprising R&D, marketing, and sales departments to collaboratively develop and launch new products. Implement a feedback loop to gather customer input and refine products based on their preferences.
6. **Targeted Marketing Campaigns**:
    - **Recommendation**: Design and execute marketing campaigns that emphasize the most important attributes and their preferred levels. Use customer language and highlight the benefits that resonate most with them.
    - **Action**: Create marketing content that showcases the top attributes, such as the superior quality of the crust or the variety of premium toppings. Utilize social media, email marketing, and in-store promotions to reach a broader audience.
7. **Continuous Monitoring and Adjustment**:
    - **Recommendation**: Continuously monitor customer preferences and market trends to keep the product offerings relevant and competitive. Conduct regular conjoint analyses to stay updated on changing customer needs.
    - **Action**: Set up a system for periodic data collection and analysis. Use customer surveys, feedback forms, and sales data to track changes in preferences and adjust the product mix accordingly.

CODES

Python

```
# Assignment - 4 (Part D)

# Step 1: Load the Data
import pandas as pd
import numpy as np

# Load the pizza_data.csv file
df = pd.read_csv('pizza_data.csv')

# Step 2: Estimate Attribute Level Effects using Linear Regression
import statsmodels.api as sm
import statsmodels.formula.api as smf

# Define the model
model = 'ranking ~ C(brand, Sum) + C(price, Sum) + C(weight, Sum) + C(crust,
Sum) + C(cheese, Sum) + C(size, Sum) + C(toppings, Sum) + C(spicy, Sum)'

# Fit the model
model_fit = smf.ols(model, data=df).fit()

# Print the summary of the model
print(model_fit.summary())

# Step 3: Calculate Part-Worths and Attribute Importance
conjoint_attributes = ['brand', 'price', 'weight', 'crust', 'cheese', 'size', 'toppings',
'spicy']

level_name = []
part_worth = []
part_worth_range = []
important_levels = {}
end = 1  # Initialize index for coefficient in params
```

```
for item in conjoint_attributes:
    nlevels = len(list(np.unique(df[item])))
    level_name.append(list(np.unique(df[item])))

    begin = end
    end = begin + nlevels - 1

    new_part_worth = list(model_fit.params[begin:end])
    new_part_worth.append((-1) * sum(new_part_worth))
    important_levels[item] = np.argmax(new_part_worth)
    part_worth.append(new_part_worth)
    part_worth_range.append(max(new_part_worth) - min(new_part_worth))

print("-------------------------------------------------------------")
print("Level names:", level_name)
print("Important levels:", important_levels)
print("Part-worths:", part_worth)
print("Part-worth range:", part_worth_range)

# Step 4: Calculate Attribute Importance
attribute_importance = [round(100 * (i / sum(part_worth_range)), 2) for i in
part_worth_range]
print("Attribute importance:", attribute_importance)

# Step 5: Calculate Part-Worths of Each Attribute Level
part_worth_dict = {}
attrib_level = {}

for item, i in zip(conjoint_attributes, range(0, len(conjoint_attributes))):
    print("Attribute:", item)
    print("    Relative importance of attribute", attribute_importance[i])
    print("    Level wise part worths:")
    for j in range(0, len(level_name[i])):
        print("          {}:{}".format(level_name[i][j], part_worth[i][j]))
        part_worth_dict[level_name[i][j]] = part_worth[i][j]
        attrib_level[item] = level_name[i]
```

```
print("Part-worth dictionary:", part_worth_dict)

# Step 6: Plot Relative Importance of Attributes
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(10, 5))
sns.barplot(x=conjoint_attributes, y=attribute_importance)
plt.title('Relative importance of attributes')
plt.xlabel('Attributes')
plt.ylabel('Importance')
plt.show()

# Step 7: Calculate Utility Scores for Each Profile
utility = []
for i in range(df.shape[0]):
    score = (part_worth_dict[df['brand'][i]] + part_worth_dict[df['price'][i]] +
part_worth_dict[df['weight'][i]] +
             part_worth_dict[df['crust'][i]] + part_worth_dict[df['cheese'][i]] +
part_worth_dict[df['size'][i]] +
             part_worth_dict[df['toppings'][i]] + part_worth_dict[df['spicy'][i]])
    utility.append(score)

df['utility'] = utility
print("Utility scores:", utility)

# Step 8: Find the Combination with Maximum Utility
max_utility_index = np.argmax(utility)
print("The profile that has the highest utility score:\n",
df.iloc[max_utility_index])

# Step 9: Determine the Preferred Levels in Each Attribute
for i, j in zip(attrib_level.keys(), range(0, len(conjoint_attributes))):
    print("Preferred level in {} is :: {}".format(i,
level_name[j][important_levels[i]]))
```

R Language

```
# Load required libraries (assuming they are already loaded)
library(stats)
library(dplyr)
library(tidyr)
library(ggplot2)  # Added for plotting

# Step 1: Load dataset (replace 'pizza_data.csv' with your actual dataset path)
df <- read.csv('pizza_data.csv')

# Step 2: Define the linear regression model
model <- formula(ranking ~ brand + price + weight + crust + cheese + size +
toppings + spicy)
model_fit <- lm(model, data = df)

# Step 3: Print model summary
print(summary(model_fit))

# Step 4: Define conjoint attributes
conjoint_attributes <- c('brand', 'price', 'weight', 'crust', 'cheese', 'size', 'toppings',
'spicy')

# Step 5: Calculate part-worths and attribute importance
part_worth <- list()
part_worth_range <- c()
important_levels <- list()
end <- 1

for (item in conjoint_attributes) {
  nlevels <- length(unique(df[[item]]))
  level_name <- unique(df[[item]])

  begin <- end
  end <- begin + nlevels - 1

  new_part_worth <- coef(model_fit)[begin:end]
  new_part_worth <- c(new_part_worth, (-1) * sum(new_part_worth))

  important_levels[[item]] <- which.max(new_part_worth)
  part_worth[[item]] <- new_part_worth
  part_worth_range <- c(part_worth_range, max(new_part_worth) -
min(new_part_worth))
}
```

```
# Print part-worths and attribute importance
print("Part-Worths:")
print(part_worth)
print("Attribute Importance:")
attribute_importance <- round(100 * (part_worth_range /
sum(part_worth_range)), 2)
print(attribute_importance)

# Step 6: Calculate utility scores for each profile
part_worth_dict <- list()
attrib_level <- list()
for (item in seq_along(conjoint_attributes)) {
  part_worth_dict[[conjoint_attributes[item]]] <- setNames(part_worth[[item]],
level_name[item])
  attrib_level[[conjoint_attributes[item]]] <- level_name[item]
}

df$utility <- rowSums(sapply(conjoint_attributes, function(item)
part_worth_dict[[item]][df[[item]]]))

# Print the profile with maximum utility
print("Profile with highest utility score:")
print(df[which.max(df$utility), ])

# Determine preferred levels in each attribute
for (item in seq_along(conjoint_attributes)) {
  pref_index <- important_levels[[conjoint_attributes[item]]]
  pref_level <- names(attrib_level[[conjoint_attributes[item]]])[pref_index + 1]
# Adjust index due to R indexing starting from 1
  print(paste("Preferred level in", conjoint_attributes[item], "is:", pref_level))
}

# Plot the relative importance of attributes
importance_df <- data.frame(Attribute = conjoint_attributes, Importance =
attribute_importance)
ggplot(importance_df, aes(x = Attribute, y = Importance)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Relative Importance of Attributes", x = "Attributes", y =
"Importance (%)") +
  theme_minimal()
```

REFERENCES

1. Books

- **Green, P. E., & Srinivasan, V.** (1990). *Conjoint Analysis in Marketing: New Developments with Implications for Research and Practice*. *Journal of Marketing*, 54(4), 3-19. https://doi.org/10.1177/002224299005400401
  - This seminal book provides foundational knowledge and advances in conjoint analysis for marketing.
- **Orme, B. K.** (2014). *Getting Started with Conjoint Analysis: Strategies for Product Design and Pricing Research*. *Research Publishers LLC*.
  - A practical guide for implementing conjoint analysis techniques in product design and pricing strategies.

2. Journal Articles

- **Ryan, M., & Farrar, S.** (2000). *Discrete Choice Experiments in Health Economics: Theories, Models, and Applications*. *Health Economics*, 9(4), 273-282. https://doi.org/10.1002/1099-1050(200006)9:4<273::AID-HEC558>3.0.CO;2-7
  - Explores discrete choice experiments, a methodology related to conjoint analysis, particularly in health economics.
- **Vriens, M., & Verhoef, P. C.** (2001). *Conjoint Analysis: An Overview*. *International Journal of Market Research*, 43(3), 345-357. https://doi.org/10.1177/147078530104300305
  - An overview of conjoint analysis methods and their market research applications.

3. Conference Papers

- **Hensher, D. A., & Greene, W. H.** (2003). *The Mixed Logit Model: Theoretical Development and Applications*. *Proceedings of the 2003 European Transport Conference*.
  - Discusses advanced mixed logit models used in conjoint analysis, presented at a key transport conference.

4. Websites and Online Resources

- **Sawtooth Software.** (2023). *Conjoint Analysis Resources*. https://www.sawtoothsoftware.com/conjoint-analysis
  - Provides resources and tools for conducting and understanding conjoint analysis.
- **Conjoint.ly.** (2024). *Conjoint Analysis Guide*. https://www.conjoint.ly/guide
  - An online guide with practical tips and examples for performing conjoint analysis.

5. Methodology References

- **Hair, J. F., Black, W. C., Babin, B. J., & Anderson, R. E.** (2019). *Multivariate Data Analysis: A Global Perspective*. *Pearson*.
  - Covers multivariate techniques, including those used in conjunction with conjoint analysis.
- **Kuhfeld, W. F.** (2010). *Marketing Research Methods in SAS: Experimental Design, Choice, Conjoint, and Graphical Techniques*. *SAS Institute*. https://support.sas.com/documentation/cdl/en/marketing/65264.htm
  - A detailed resource for experimental design and choice modeling, including conjoint analysis techniques in SAS.

6. Supplementary Readings

- **Grewal, D., & Levy, M.** (2016). *Marketing*. *McGraw-Hill Education*.
  - Provides foundational marketing concepts and methodologies that support understanding of conjoint analysis.
- **Moran, J. A.** (2014). *Introduction to Conjoint Analysis*. *Marketing Research*, 16(2), 10-14.
  - A beginner’s guide to understanding the basics of conjoint analysis for marketing research.