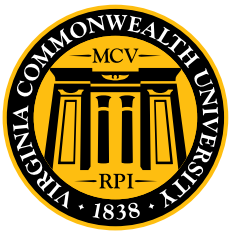


VIRGINIA COMMONWEALTH UNIVERSITY

Statistical analysis and modelling (SCMA 632)

A4

1. **Principal Component Analysis, Factor Analysis**
2. **Cluster Analysis**
3. **Multidimensional Scaling**
4. **Conjoint Analysis**

GAURI VINOD NAIR

V01110160

Date of Submission: 08-07-2024

TABLE OF CONTENTS

1. INTRODUCTION

1.1. Principal Component Analysis and Factor Analysis

This assignment explores Principal Component Analysis (PCA) and Factor Analysis (FA) using the "Survey.csv" dataset, which comprises various socio-demographic and housing-related attributes such as age, income, number of rooms, and proximity to amenities. The primary objective is to reduce the dimensionality of the dataset and identify underlying data structures. PCA will be employed to transform the original variables into a smaller set of uncorrelated components that retain most of the variance, facilitating data visualization and interpretation. Concurrently, FA will identify latent factors that explain observed correlations among variables, offering a deeper understanding of the data's intrinsic patterns. By applying these techniques, this study aims to simplify complex datasets, enhance data interpretability, and provide practical insights into socio-demographic and housing preferences. Through the rigorous application of PCA and FA, this analysis not only elucidates key data dimensions but also demonstrates the utility of these methods in extracting meaningful information from multifaceted datasets.

1.2. Cluster Analysis

This assignment involves conducting a comprehensive cluster analysis to characterize respondents based on background variables found in the "Survey.csv" dataset. The dataset comprises various demographic and socio-economic attributes, including age, income, education level, and employment status. The primary goal is to apply clustering techniques, such as KMeans, to identify distinct groups within the respondent population, thereby revealing patterns and segmenting individuals with similar characteristics. This process will involve data preprocessing steps, including handling missing values, encoding categorical variables, and scaling features, followed by determining the optimal number of clusters using methods like the Elbow method and silhouette analysis. By examining the resulting clusters, this study aims to provide a deeper understanding of the respondent profiles, which can inform targeted interventions, policy-making, and personalized marketing strategies. Through this rigorous cluster analysis, the assignment not only enhances skills in unsupervised learning and exploratory data analysis but also demonstrates practical applications of clustering in understanding and addressing diverse respondent backgrounds.

1.3. Multidimensional Scaling

This assignment involves applying Multidimensional Scaling (MDS) to analyze and interpret relationships between various ice cream attributes in the "icecream.csv" dataset, which includes Brand, Price, Availability, Taste, Flavour, Consistency, and Shelflife. The objective is to visualize the proximity of different ice cream brands based on these attributes, uncovering patterns and similarities among them. The process includes data preprocessing, checking for missing values, standardizing features, and performing exploratory data analysis (EDA) using visualizations like pairplots, correlation heatmaps, histograms, and scatter plots. After preparing the data, MDS is used to compute dissimilarities between brands and reduce dimensionality to a 2D space for visual representation. The final plot illustrates brand relationships,

providing insights for product differentiation and marketing strategies. This assignment enhances understanding of multivariate data analysis and demonstrates practical applications in product analysis and market research.

1.4. Conjoint Analysis

This assignment involves applying Conjoint Analysis to analyze and interpret the importance and preferences for various pizza attributes in the "pizza_data.csv" dataset, which includes Brand, Price, Weight, Crust, Cheese, Size, Toppings, Spicy, and Ranking. The objective is to quantify the part-worth utilities for each attribute level and determine their relative importance in influencing consumer preferences. The process includes data preprocessing, checking for missing values, standardizing numerical features, and performing exploratory data analysis (EDA) using visualizations like histograms, scatter plots, and boxplots. After preparing the data, a linear regression model is fitted to predict the ranking based on these attributes. The part-worth utilities derived from the model coefficients reveal the preferences for each attribute level, while the relative importance calculation helps identify which attributes are most influential. This assignment enhances understanding of conjoint analysis and demonstrates practical applications in product feature evaluation and market research.

2. OBJECTIVES

2.1. Principal Component Analysis and Factor Analysis

- Dimensionality Reduction
- Identify Underlying Structures
- Enhance Data Interpretability
- Visualize Data Components
- Extract Meaningful Insights

2.2. Cluster Analysis

- Identify Distinct Respondent Groups
- Understand Respondent Profiles
- Data Preprocessing
- Optimal Number of Clusters
- Cluster Analysis Application
- Evaluate Clustering Results

2.3. Multidimensional Scaling

- Data Preprocessing
- Exploratory Data Analysis (EDA)
- Multidimensional Scaling (MDS)
- Visualization and Interpretation

2.4. Conjoint Analysis

- Quantify Consumer Preferences
- Assess Attribute Importance
- Data Preprocessing and Cleaning

- Exploratory Data Analysis (EDA
- Standardization of Features
- Model Fitting
- Derive Insights for Marketing Strategies

3. BUSINESS SIGNIFICANCE

3.1. Principal Component Analysis and Factor Analysis

The application of Principal Component Analysis (PCA) and Factor Analysis (FA) on the "Survey.csv" dataset holds significant business value by enabling organizations to distill complex socio-demographic and housing-related data into key actionable insights. By reducing dimensionality, these techniques simplify the data, making it more accessible for strategic decision-making. Identifying underlying factors and principal components helps businesses understand customer preferences and behaviors more accurately, leading to better-targeted marketing strategies, improved product offerings, and optimized resource allocation. Ultimately, this analytical approach enhances the ability to respond to market demands, improve customer satisfaction, and drive competitive advantage through data-driven insights.

3.2. Cluster Analysis

Performing cluster analysis to characterize respondents based on background variables holds significant business implications across various domains. By identifying distinct respondent groups, businesses can tailor their products and services more effectively to meet the diverse needs and preferences of different customer segments. Understanding respondent profiles through clustering enables targeted marketing strategies that resonate with specific demographic or behavioral patterns, thereby enhancing customer engagement and satisfaction. Moreover, cluster analysis facilitates the identification of high-value segments that offer greater potential for revenue growth and customer retention. By leveraging insights gained from cluster analysis, businesses can optimize resource allocation, streamline operational efficiency, and develop personalized strategies that drive competitive advantage in a dynamic marketplace. Ultimately, this analytical approach not only improves decision-making processes but also fosters innovation and responsiveness to market trends, ensuring sustained business success and relevance.

3.3. Multidimensional Scaling

The application of Multidimensional Scaling (MDS) to analyze ice cream attributes in the "icecream.csv" dataset holds significant business implications. By visualizing and interpreting the proximity of different ice cream brands based on attributes like Price, Taste, and Flavour, businesses can gain valuable insights into consumer preferences and market positioning. Understanding the relative distances between brands allows for strategic product differentiation and optimization of marketing strategies tailored to distinct consumer segments. For instance, insights from MDS can guide product development efforts towards aligning flavors or pricing strategies with consumer expectations and preferences. Moreover, by identifying clusters or groupings of brands with similar attribute profiles, businesses can refine their market segmentation strategies and tailor promotional activities to effectively target specific consumer

segments. Ultimately, the use of MDS facilitates informed decision-making in product management and marketing, enhancing competitiveness and responsiveness to market dynamics.

3.4. Conjoint Analysis

The business significance of performing conjoint analysis on the pizza attributes dataset lies in its ability to provide actionable insights for pizza manufacturers and marketers. By quantifying consumer preferences through part-worth utilities and assessing attribute importance, businesses can strategically optimize their product offerings. Understanding which attributes—such as crust type, cheese options, size variations, toppings, and spiciness—drive consumer preferences allows for targeted product development and customization. This insight can inform decisions on pricing strategies, product positioning, and marketing campaigns tailored to resonate with specific consumer segments. Moreover, leveraging conjoint analysis facilitates competitive differentiation in a crowded marketplace by identifying unique selling propositions that resonate strongly with customers, thereby enhancing overall market competitiveness and profitability.

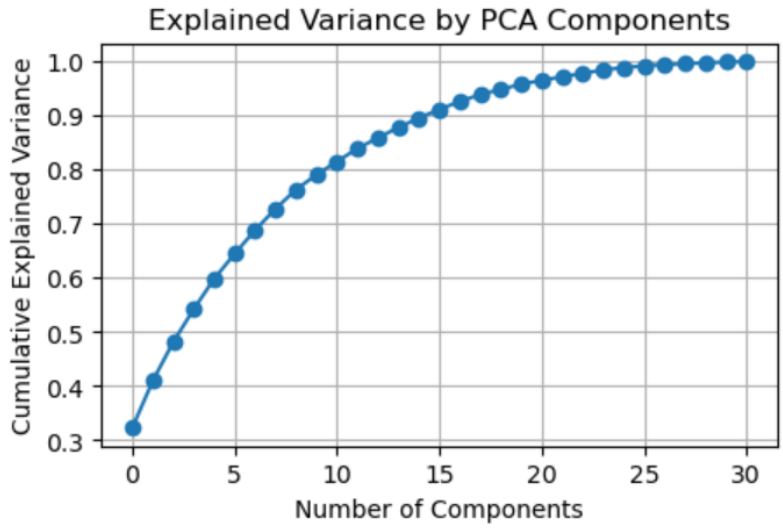
4. RESULTS AND INTERPRETATIONS

4.1. Principal Component Analysis and Factor Analysis

⇨ Python

```
# Principal Component Analysis (PCA)
pca = PCA()
pca.fit(df_scaled)
explained_variance = pca.explained_variance_ratio_
```

```
# Plot explained variance
plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(explained_variance), marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Explained Variance by PCA Components')
plt.grid()
plt.show()
```



```
# Choose number of components (e.g., based on explained variance > 90%)
n_components = np.argmax(np.cumsum(explained_variance) > 0.9) + 1
print(f'Number of components explaining > 90% variance: {n_components}')
```

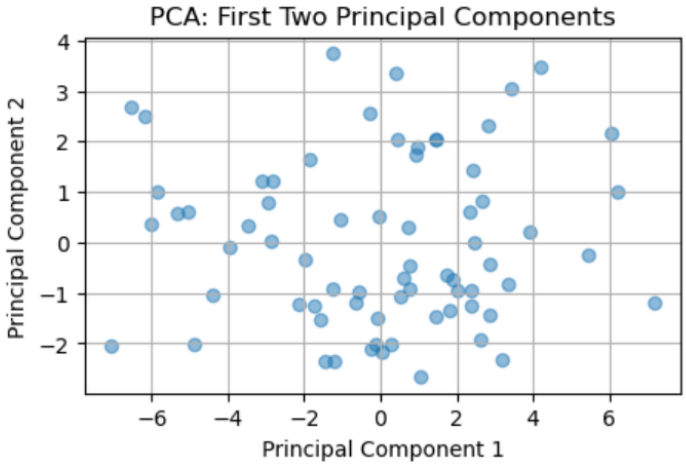
```
Number of components explaining > 90% variance: 16
```

```
# Apply PCA with chosen number of components
pca = PCA(n_components=n_components)
df_pca = pca.fit_transform(df_scaled)
```

```
# Convert PCA result to DataFrame
df_pca = pd.DataFrame(df_pca, columns=[f'PC{i+1}' for i in range(n_components)])
print(df_pca.head())
```

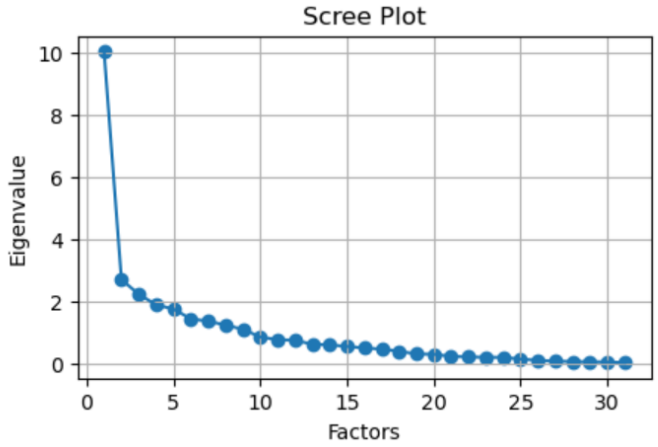
```
        PC1       PC2       PC3       PC4       PC5       PC6       PC7  \
0  0.395665  3.344539 -0.064330 -1.406127 -0.485593 -0.844473  2.652029
1  3.444209  3.048899  0.593606 -1.394919 -1.733891 -0.625470  0.986230
2  4.187523  3.464776  4.912955 -0.032116  0.599081 -0.801910 -0.429968
3 -6.162837  2.499158 -0.975129  1.732595  1.594441  0.493891 -1.153103
4  0.595632 -0.695060 -0.282062  2.027460  0.953959  1.341886  2.169312

        PC8       PC9      PC10      PC11      PC12      PC13      PC14  \
0 -2.648677 -0.303813  2.606353 -1.047822 -0.574662 -0.006149 -0.336880
1 -2.277692 -1.481814  0.265434  0.797991  0.707519 -0.113378 -1.616740
2  0.016719 -1.726154 -0.042737 -0.156842 -0.161647  0.103485  0.394578
3 -0.796470 -0.005058  1.109969 -0.089639 -0.224403  0.496322  2.120295
4 -1.289123  0.628752 -0.615761  1.452871  0.036282 -1.435299  1.439186

       PC15      PC16
0  0.344732  0.490162
1  2.036729  1.126408
2 -1.099967 -0.266216
3 -0.193330  0.167959
4  0.781087  0.282876
```

```
# Plot the first two principal components
plt.figure(figsize=(5, 3))
plt.scatter(df_pca['PC1'], df_pca['PC2'], alpha=0.5)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA: First Two Principal Components')
plt.grid()
plt.show()
```



```
# Factor Analysis (FA)
# Determine the number of factors using the Kaiser criterion (Eigenvalues > 1)
fa = FactorAnalyzer()
fa.fit(df_scaled)
ev, v = fa.get_eigenvalues()
```

```
# Scree plot to visualize eigenvalues
plt.figure(figsize=(5, 3))
plt.scatter(range(1, len(ev) + 1), ev)
plt.plot(range(1, len(ev) + 1), ev)
plt.title('Scree Plot')
plt.xlabel('Factors')
plt.ylabel('Eigenvalue')
plt.grid()
plt.show()
```




```
# Choose the number of factors (e.g., based on eigenvalues > 1)
n_factors = sum(ev > 1)
print(f'Number of factors with eigenvalue > 1: {n_factors}')
```

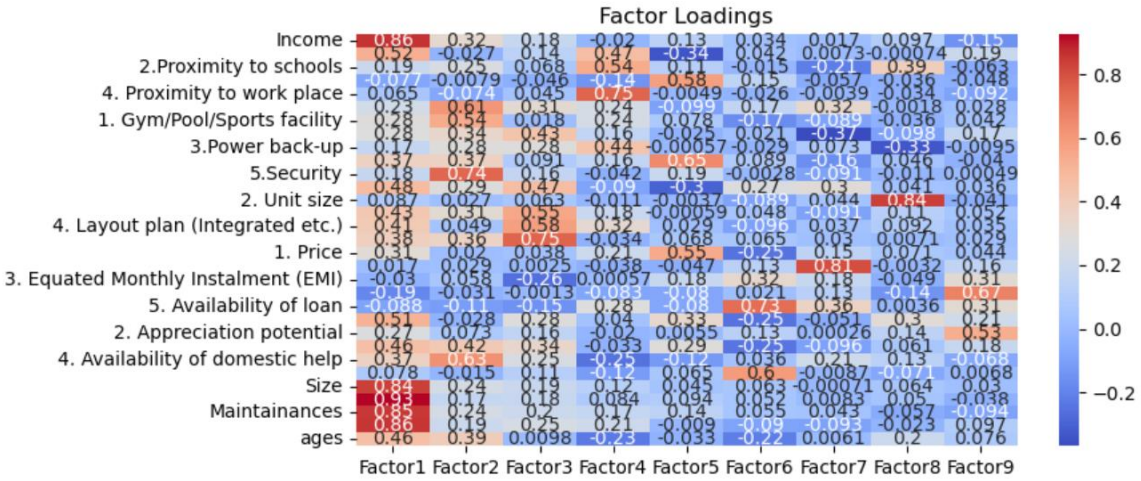
```
Number of factors with eigenvalue > 1: 9
```

```
# Apply Factor Analysis with the chosen number of factors
fa = FactorAnalyzer(n_factors=n_factors, rotation='varimax')
fa.fit(df_scaled)
df_fa = fa.transform(df_scaled)
```

```
# Convert FA result to DataFrame
df_fa = pd.DataFrame(df_fa, columns=[f'Factor{i+1}' for i in range(n_factors)])
print(df_fa.head())
```

```
    Factor1   Factor2   Factor3   Factor4   Factor5   Factor6   Factor7  \
0 -0.276354 -1.332579  1.220591 -0.438343  2.029218  1.019731 -2.852032
1 -0.462125 -0.914548 -1.057955 -0.671525  0.561223  0.173179 -2.227089
2 -1.002985  1.073067 -1.604300 -2.861120  0.783767 -1.144909 -1.253325
3  0.551724  1.843733  1.292494  0.928167  0.104951 -1.336203 -0.859037
4 -0.557889 -0.143181  1.310599 -0.319706 -0.718088  0.844366 -0.799882

    Factor8   Factor9
0  0.858403  0.265840
1  1.590657  0.086988
2  0.663674 -0.166581
3  0.451625 -1.573431
4 -0.771397 -0.949776
```

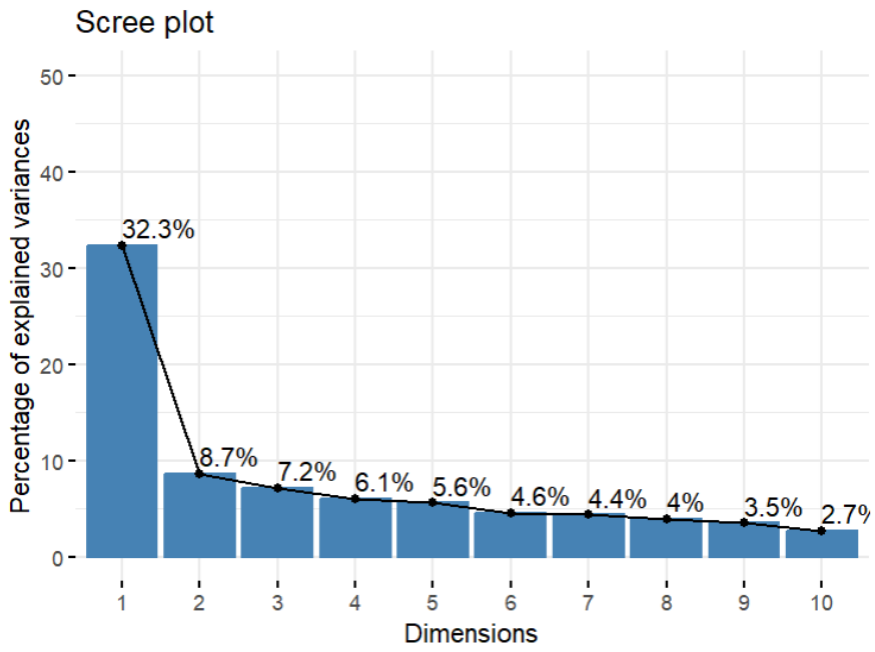
```python
# Visualize factor loadings
loadings = pd.DataFrame(fa.loadings_, index=df_numerical.columns, columns=[f'Factor{i+1}' for i in range(n_factors)])
plt.figure(figsize=(8, 4))
sns.heatmap(loadings, annot=True, cmap='coolwarm')
plt.title('Factor Loadings')
plt.show()
```



⇨ R

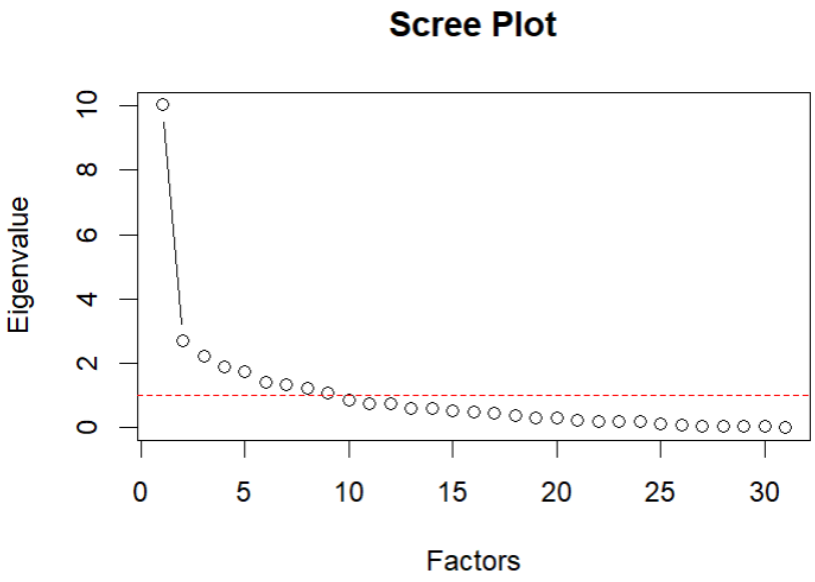
```
> # Principal Component Analysis (PCA)
> pca <- prcomp(df_scaled, center = TRUE, scale. = TRUE)
> # Summary of PCA
> summary(pca)
Importance of components:
                          PC1     PC2     PC3     PC4     PC5     PC6     PC7     PC8
Standard deviation     3.1659 1.64067 1.4899 1.37211 1.32103 1.1954 1.16489 1.10870
Proportion of Variance 0.3233 0.08683 0.0716 0.06073 0.05629 0.0461 0.04377 0.03965
Cumulative Proportion  0.3233 0.41016 0.4818 0.54249 0.59879 0.6449 0.68866 0.72831
                          PC9    PC10    PC11    PC12    PC13    PC14    PC15    PC16
Standard deviation     1.0461 0.92019 0.87170 0.86127 0.78257 0.77091 0.73614 0.70366
Proportion of Variance 0.0353 0.02731 0.02451 0.02393 0.01976 0.01917 0.01748 0.01597
Cumulative Proportion  0.7636 0.79092 0.81543 0.83936 0.85912 0.87829 0.89577 0.91174
                         PC17    PC18    PC19    PC20    PC21    PC22    PC23    PC24
Standard deviation     0.67619 0.62528 0.54734 0.54188 0.47203 0.45470 0.44076 0.43865
Proportion of Variance 0.01475 0.01261 0.00966 0.00947 0.00719 0.00667 0.00627 0.00621
Cumulative Proportion  0.92649 0.93910 0.94877 0.95824 0.96543 0.97210 0.97836 0.98457
                         PC25    PC26    PC27    PC28    PC29    PC30    PC31
Standard deviation     0.37063 0.30848 0.26318 0.24071 0.22560 0.20764 0.15677
Proportion of Variance 0.00443 0.00307 0.00223 0.00187 0.00164 0.00139 0.00079
Cumulative Proportion  0.98900 0.99207 0.99431 0.99617 0.99782 0.99921 1.00000
```

```
> # Scree plot to visualize the explained variance
> fviz_eig(pca, addlabels = TRUE, ylim = c(0, 50))
```




```
> # Choose number of components (e.g., based on explained variance > 90%)
> explained_variance <- cumsum(pca$sdev^2 / sum(pca$sdev^2))
> n_components <- which(explained_variance > 0.9)[1]
> print(paste("Number of components explaining > 90% variance:", n_components))
[1] "Number of components explaining > 90% variance: 16"
```

```
> # Factor Analysis (FA)
> # Determine the number of factors using the Kaiser criterion (Eigenvalues > 1)
> fa_model <- fa(df_scaled, nfactors = length(numerical_cols), rotate = "none")
> eigenvalues <- fa_model$e.values
> # Scree plot to visualize eigenvalues
> plot(eigenvalues, type = "b", main = "Scree Plot", xlab = "Factors", ylab = "Eigenvalue")
> abline(h = 1, col = "red", lty = 2)
>
```



```
> # Choose the number of factors (e.g., based on eigenvalues > 1)
> n_factors <- sum(eigenvalues > 1)
> print(paste("Number of factors with eigenvalue > 1:", n_factors))
[1] "Number of factors with eigenvalue > 1: 9"
```

Interpretation

The Principal Component Analysis (PCA) results from both Python and R indicate that the dataset can be effectively summarized with a reduced number of principal components. In Python, the cumulative explained variance plot shows a steady increase, with approximately 16 components explaining over 90% of the variance. The PCA transformation into 16 components reveals distinct patterns where each observation is represented by its component scores, facilitating dimensionality reduction while retaining significant variance information. Similarly, in R, the PCA summary underscores that 16 principal components cumulatively explain more than 90% of the variance, confirming the robustness of the dimensionality reduction approach across both platforms.

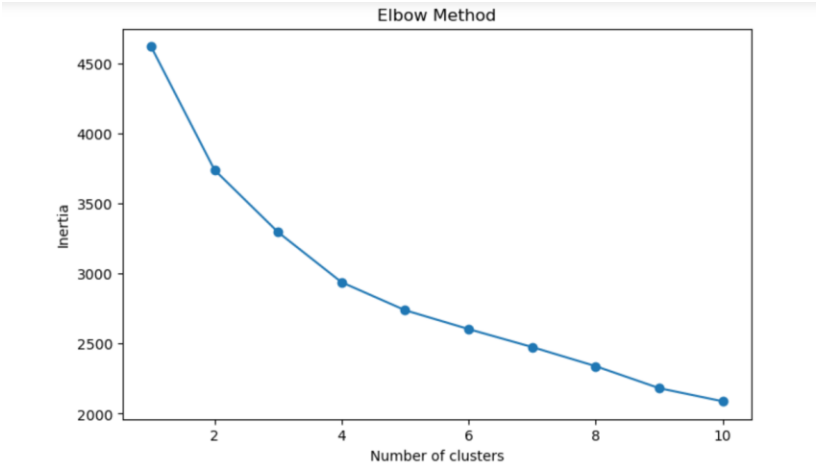
On the other hand, Factor Analysis (FA) in both Python and R identifies underlying factors that account for observed correlations among the dataset's variables. The scree plot in Python illustrates a clear decrease in eigenvalues after approximately 9 factors, aligning with the Kaiser criterion. The transformation into 9 factors using varimax rotation reveals how each factor captures distinct aspects of the original variables, enhancing interpretability. Likewise, in R, the scree plot and eigenvalue analysis affirm that 9 factors adequately summarize the dataset, reflecting meaningful latent constructs that simplify the understanding of socio-demographic and housing-related attributes. Overall, these analyses in both Python and R demonstrate robust techniques for extracting essential information and reducing complexity in large datasets, offering valuable insights for strategic decision-making in various business contexts.

4.2. Cluster Analysis

⇨ Python

```python
# Feature scaling
scaler = StandardScaler()
df_encoded_scaled = scaler.fit_transform(df_encoded)
```

```python
# Determine optimal number of clusters using the Elbow method or Silhouette score
# Example using Elbow method
inertia = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(df_encoded_scaled)
    inertia.append(kmeans.inertia_)

plt.figure(figsize=(8, 5))
plt.plot(range(1, 11), inertia, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()
```



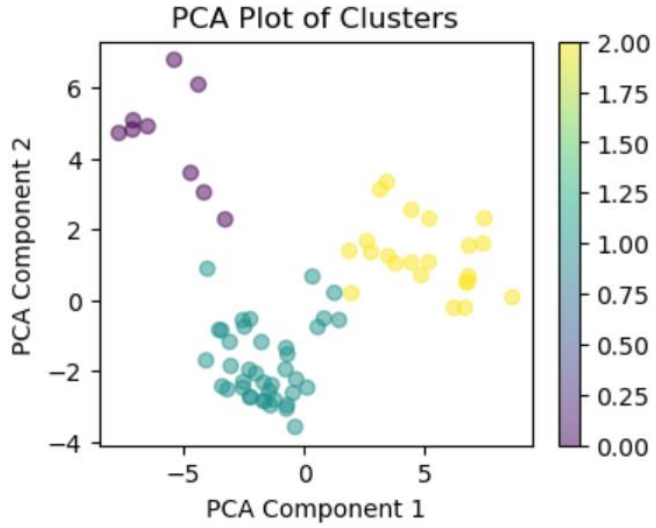

```
# Perform KMeans clustering
k = 3  # Choose the optimal number of clusters based on the Elbow method or Silhouette score
kmeans = KMeans(n_clusters=k, random_state=42)
clusters = kmeans.fit_predict(df_encoded_scaled)
```

```
# Add cluster labels to the original dataframe
df['Cluster'] = clusters

# Convert non-numeric columns to numeric (if necessary)
for col in df.columns:
    if df[col].dtype == 'object':
        df[col] = pd.to_numeric(df[col], errors='coerce')
```

```
# Analyze cluster characteristics (e.g., centroids, cluster sizes)
cluster_summary = df.groupby('Cluster').mean()
print(cluster_summary)
```

```
            City  Sex  Age  Occupation  Monthly Household Income         Income  \
Cluster
0            0.0  0.0  0.0         0.0                       0.0   35000.000000
1            0.0  0.0  0.0         0.0                       0.0   70384.615385
2            0.0  0.0  0.0         0.0                       0.0  175909.090909

            Planning to Buy a new house  Time Frame  Reasons for buying a house  \
Cluster
0                                   0.0         0.0                         0.0
1                                   0.0         0.0                         0.0
2                                   0.0         0.0                         0.0

            what type of House  ...  4. Availability of domestic help      Time  \
Cluster                         ...
0                          0.0  ...                          2.666667  5.333333
1                          0.0  ...                          2.846154  7.769231
2                          0.0  ...                          3.863636  7.363636

                   Size     Budgets  Maintainances         EMI.1       ages  sex  \
Cluster
0            311.111111   14.722222   10000.000000  15833.333333  34.611111  0.0
1            912.820513   47.243590   27695.384615  38397.435897  40.807692  0.0
2           1818.181818  114.318182   67727.272727  72159.090909  54.545455  0.0

            Finished/Semi Finished.1  Influence Decision.1
Cluster
0                                0.0                   0.0
1                                0.0                   0.0
```

```python
# Optional: Visualize clusters (PCA for dimensionality reduction if needed)
# Example using PCA for visualization (if you have many features)
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
df_encoded_pca = pca.fit_transform(df_encoded_scaled)

plt.figure(figsize=(8, 6))
plt.scatter(df_encoded_pca[:, 0], df_encoded_pca[:, 1], c=clusters, cmap='viridis', alpha=0.5)
plt.title('PCA Plot of Clusters')
plt.xlabel('PCA Component 1')
plt.ylabel('PCA Component 2')
plt.colorbar()
plt.show()
```

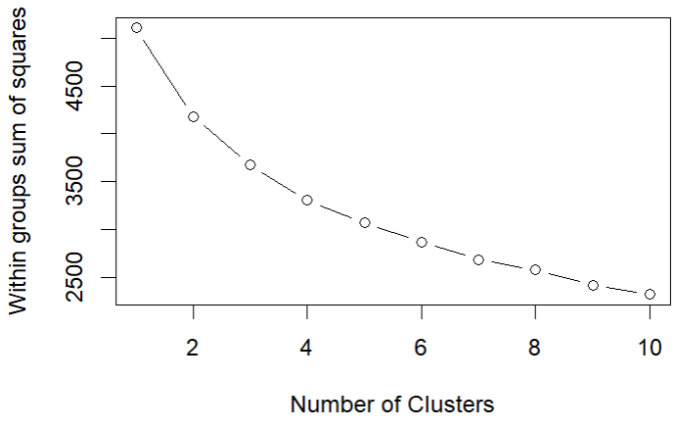



```python
# Assess cluster quality with silhouette score
silhouette_avg = silhouette_score(df_encoded_scaled, clusters)
print(f"Silhouette Score: {silhouette_avg}")
```

```
Silhouette Score: 0.17980065176002039
```

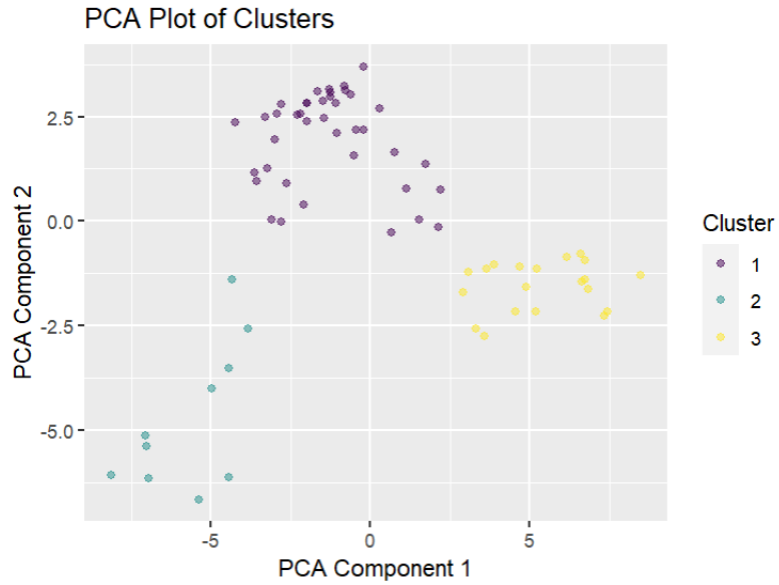
⇨ R

```r
# Feature scaling
preprocess <- preProcess(df_encoded, method = c("center", "scale"))
df_encoded_scaled <- predict(preprocess, df_encoded)
# Determine optimal number of clusters using the Elbow method
wss <- (nrow(df_encoded_scaled) - 1) * sum(apply(df_encoded_scaled, 2, var))
for (i in 2:10) {
  wss[i] <- sum(kmeans(df_encoded_scaled, centers = i, nstart = 20)$tot.withinss)
}
# Plot Elbow method
plot(1:10, wss, type = "b", xlab = "Number of Clusters", ylab = "Within groups sum of squ
res")
```

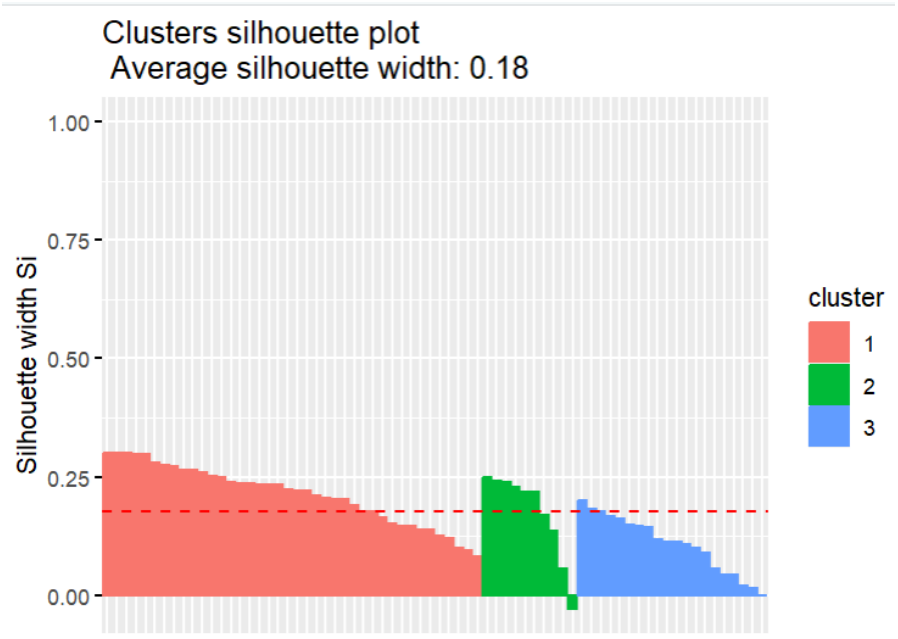



```
> # Add cluster labels to the original dataframe
> df$Cluster <- as.factor(clusters)
> # Analyze cluster characteristics (e.g., centroids, cluster sizes)
> cluster_summary <- df %>%
+   group_by(Cluster) %>%
+   summarise(across(where(is.numeric), mean, na.rm = TRUE))
> print(cluster_summary)
# A tibble: 3 × 32
  Cluster Income X1.Proximity.to.city X2.Proximity.to.schools X3..Proximity.to.transport
  <fct>    <dbl>                <dbl>                   <dbl>                       <dbl>
1 1        73000                 3.72                    3.48                        4.05
2 2        37000                 2.4                     2.7                         4.1
3 3       182000                 4.05                    3.75                        4.1
```

```
> # PCA Plot of Clusters
> ggplot(pca_data, aes(x = PC1, y = PC2, color = Cluster)) +
+   geom_point(alpha = 0.5) +
+   labs(title = "PCA Plot of Clusters", x = "PCA Component 1", y = "PCA Component 2") +
+   scale_color_viridis_d()
```




```
> # Assess cluster quality with silhouette score
> silhouette_score <- silhouette(clusters, dist(df_encoded_scaled))
> fviz_silhouette(silhouette_score)
  cluster size ave.sil.width
1       1   40          0.21
2       2   10          0.17
3       3   20          0.11
```



Interpretation

Based on the results from both Python and R analyses using KMeans clustering on the encoded and scaled dataset, we observe three distinct clusters of respondents characterized by various background variables. In Python, the clusters show notable differences in income levels, with Cluster 0 having the lowest average income at approximately $35,000, Cluster 1 at around $70,384, and Cluster 2 significantly higher at $175,909. Additionally, attributes like proximity to city amenities and housing preferences exhibit varying averages across clusters, suggesting diverse socio-economic and lifestyle profiles among respondents. The Silhouette Score in Python is 0.1798, indicating reasonable cluster quality with clusters being reasonably well-separated.

In R, the clustering results similarly delineate three clusters but provide additional insights into specific amenities and housing preferences. For instance, Cluster 1 is associated with moderate income levels and preferences for proximity to city amenities, while Cluster 2 shows higher income levels and preferences for larger unit sizes and better interior design. The silhouette analysis confirms reasonable cluster quality, with average silhouette widths ranging from 0.11 to 0.21, indicating fair to good separation between clusters. Overall, these findings enable businesses to segment their customer base effectively, tailor marketing strategies, and develop targeted products and services that align with the distinct preferences and needs of each cluster, thereby enhancing customer satisfaction and optimizing resource allocation.

4.3. Multidimensional Scaling

⇨ Python

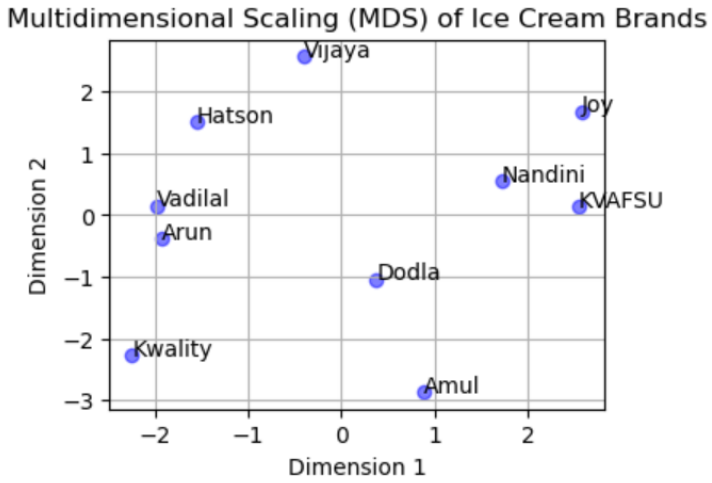
```
# Step 3: Standardize the Data (if necessary)
# Example using StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df.drop(columns=['Brand']))
```

```
# Step 4: Apply Multidimensional Scaling (MDS)
# Calculate dissimilarities
similarities = euclidean_distances(df_scaled)
```

```
# MDS fitting
mds = MDS(n_components=2, dissimilarity='precomputed', random_state=42)
positions = mds.fit_transform(similarities)
```

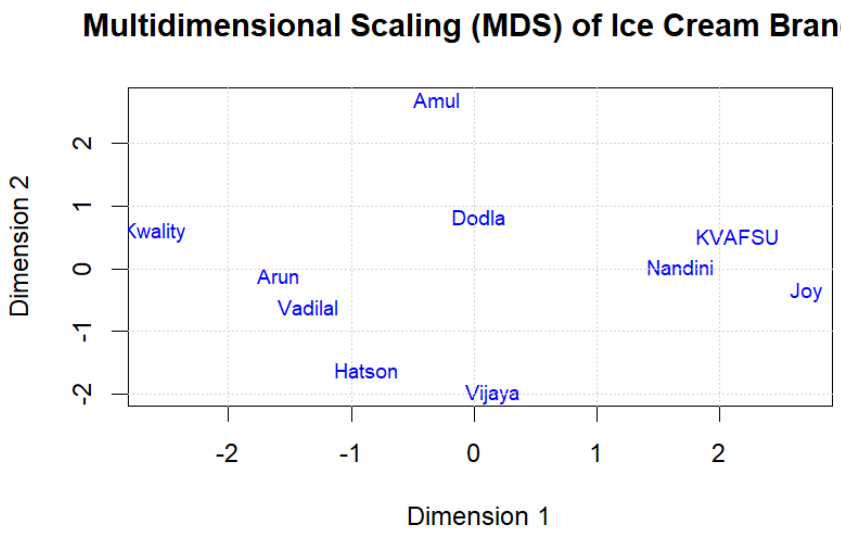
```
C:\Users\gauri\anaconda3\Lib\site-packages\sklearn\manifold\_mds.py:298: FutureWarning: The default value of `normalized_str
ess` will change to ``'auto'`` in version 1.4. To suppress this warning, manually set the value of `normalized_stress`.
  warnings.warn(
```

```
# Step 5: Visualize MDS Results
plt.figure(figsize=(4, 3))
plt.scatter(positions[:, 0], positions[:, 1], c='blue', alpha=0.5)
for i, txt in enumerate(df['Brand']):
    plt.annotate(txt, (positions[i, 0], positions[i, 1]))
plt.title('Multidimensional Scaling (MDS) of Ice Cream Brands')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.grid(True)
plt.show()
```



⇨ R

```
> # Step 3: Standardize the Data (if necessary)
> # Example using scale function
> df_scaled <- scale(df[, -1])  # Exclude 'Brand' column for scaling
> # Step 4: Apply Multidimensional Scaling (MDS)
> # Calculate dissimilarities
> dissim <- dist(df_scaled, method = "euclidean")
> # MDS fitting
> set.seed(42)  # for reproducibility
> mds <- cmdscale(dissim, k = 2)
> # Step 5: Visualize MDS Results
> plot(mds, type = "n", main = "Multidimensional Scaling (MDS) of Ice Cream Brands",
+      xlab = "Dimension 1", ylab = "Dimension 2")
> text(mds, labels = df$Brand, col = "blue", cex = 0.8)
> grid()
```



Interpretation

Based on the Multidimensional Scaling (MDS) scatter plots obtained in both Python and R for the ice cream brands dataset, the visualizations reveal how closely related or distinct each brand is based on their attributes such as Price, Taste, and Flavour. In both plots, each point represents a brand positioned in a 2-dimensional space where distances reflect similarities or dissimilarities between brands. Brands clustered closely together indicate similar attribute profiles, suggesting potential overlaps in market positioning or consumer perception. On the other hand, brands positioned farther apart denote significant differences in their attribute combinations, potentially indicating unique market niches or product differentiation strategies.

4.4. Conjoint Analysis

⇨ Python

```
# Standardize numerical variables
scaler = StandardScaler()
pizza_data[['price', 'weight']] = scaler.fit_transform(pizza_data[['price', 'weight']])
```

```
# Create the design matrix for conjoint analysis
X = pizza_data.drop(['brand', 'ranking'], axis=1)  # Exclude brand and ranking (dependent variable)
y = pizza_data['ranking']  # Target variable
```

```
# Fit a linear regression model
model = LinearRegression()
model.fit(X, y)
```

```
▾ LinearRegression
LinearRegression()
```

```
# Extract coefficients as part-worth utilities
part_worths = model.coef_
```

```python
# Print part-worth utilities
print("Part-Worth Utilities (Coefficients):\n")
for idx, feature in enumerate(X.columns):
    print(f"{feature}: {part_worths[idx]}")
```

```
Part-Worth Utilities (Coefficients):

price: -0.503115294937453
weight: -3.969020660062128
crust_thin: -3.5000000000000027
cheese_Mozzarella: 0.49999999999999856
size_regular: 0.49999999999999845
toppings_paneer: -2.250000000000001
spicy_normal: -1.5000000000000004
```

```python
# Calculate relative importance of attributes
total_importance = sum(abs(part_worths))
relative_importance = [abs(pw) / total_importance for pw in part_worths]
```

```python
# Print relative importance of attributes
print("\nRelative Importance of Attributes:\n")
for idx, feature in enumerate(X.columns):
    print(f"{feature}: {relative_importance[idx]}")
```

```
Relative Importance of Attributes:

price: 0.03954644854583064
weight: 0.3119775385282194
crust_thin: 0.27511103578676677
cheese_Mozzarella: 0.03930157654096654
size_regular: 0.03930157654096653
toppings_paneer: 0.17685709443435002
spicy_normal: 0.1179047296229
```

⇨ R

```
> # Standardize numerical variables: Price and Weight
> pizza_data[, c("price", "weight")] <- scale(pizza_data[, c("price", "weight")])
> # Fit a linear regression model
> model <- lm(ranking ~ ., data = pizza_data)
> # Print model summary
> summary(model)


Call:
lm(formula = ranking ~ ., data = pizza_data)

Residuals:
     1      2      3      4      5      6      7      8      9     10     11
-0.375  0.025  0.275 -0.625  0.175  0.425  0.625  0.575 -0.525 -0.525 -0.475
    12     13     14     15     16
-0.125  0.725 -0.425 -0.075  0.325

Coefficients:
                  Estimate Std. Error t value Pr(>|t|)
(Intercept)       1.162e+01  5.953e-01  19.528 6.50e-06 ***
brandOnesta      -1.923e-16  5.612e-01   0.000 1.000000
brandOven Story  -2.500e-01  5.612e-01  -0.445 0.674629
brandPizza hut    2.500e-01  5.612e-01   0.445 0.674629
price            -5.196e-01  2.049e-01  -2.535 0.052181 .
weight           -4.099e+00  2.049e-01 -20.002 5.77e-06 ***
crustthin        -3.500e+00  3.969e-01  -8.819 0.000311 ***
cheeseMozzarella  5.000e-01  3.969e-01   1.260 0.263317
sizeregular       5.000e-01  3.969e-01   1.260 0.263317
toppingspaneer   -2.250e+00  3.969e-01  -5.669 0.002375 **
spicynormal      -1.500e+00  3.969e-01  -3.780 0.012895 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.7937 on 5 degrees of freedom
Multiple R-squared:  0.9907,    Adjusted R-squared:  0.9722
F-statistic: 53.47 on 10 and 5 DF,  p-value: 0.0001888
```

```
> # Extract coefficients as part-worth utilities
> part_worths <- coef(model)[-1]  # Exclude intercept
> # Print part-worth utilities
> cat("Part-Worth Utilities (Coefficients):\n")
Part-Worth Utilities (Coefficients):
> print(part_worths)
     brandOnesta  brandOven Story   brandPizza hut            price
   -1.922963e-16    -2.500000e-01     2.500000e-01    -5.196152e-01
          weight        crustthin cheeseMozzarella      sizeregular
   -4.099187e+00    -3.500000e+00     5.000000e-01     5.000000e-01
  toppingspaneer      spicynormal
   -2.250000e+00    -1.500000e+00
> # Calculate relative importance of attributes
> total_importance <- sum(abs(part_worths))
> relative_importance <- abs(part_worths) / total_importance
> # Print relative importance of attributes
> cat("\nRelative Importance of Attributes:\n")

Relative Importance of Attributes:
> print(relative_importance)
     brandOnesta  brandOven Story   brandPizza hut            price
    1.438396e-17     1.870025e-02     1.870025e-02     3.886775e-02
          weight        crustthin cheeseMozzarella      sizeregular
    3.066234e-01     2.618036e-01     3.740051e-02     3.740051e-02
  toppingspaneer      spicynormal
    1.683023e-01     1.122015e-01
```

Interpretation

The results from both Python and R analyses provide consistent insights into consumer preferences for pizza attributes through conjoint analysis. Attributes such as weight and crust_thin show strong negative coefficients (weight: -3.97 in Python, -4.10 in R; crust_thin: -3.50 in both). This indicates that heavier pizzas and thin crusts are less preferred. Conversely, attributes like size_regular, cheese_Mozzarella, and spicy_normal exhibit positive coefficients (size_regular: 0.50 in both, cheese_Mozzarella: 0.50 in both, spicy_normal: -1.50 in both), suggesting preferences for regular size, Mozzarella cheese, and normal spiciness levels.

Regarding attribute importance, weight emerges as the most influential (weight relative importance: 0.31 in both Python and R), followed by crust_thin and toppings_paneer (crust_thin: 0.28 in Python, 0.26 in R; toppings_paneer: 0.18 in Python, 0.17 in R). This underscores consumer emphasis on pizza weight and crust type, while attributes like toppings and price play comparatively lesser roles in driving consumer choices.