MVA Summary Notes

Dani

Contents

Chapter 1

Intro

1.1 Introduction

This document provides an introduction to data analysis in R, covering essential topics like data importing, basic data manipulation, and visualization. Each section builds upon the fundamentals of R programming, offering practical code examples, explanations, and visualization techniques to aid in understanding multivariate data analysis.

1.2 Data Importing

- In R, data can be imported using various functions depending on the file format. For CSV files, the function `read.csv()` is commonly used.
- Example: Importing the `cars.csv` dataset:

```
# Importing a .csv file
cars <- read.csv("/path/to/cars.csv", stringsAsFactors = TRUE)

# View the imported data
View(cars)
```

- **Concept**: ‘stringsAsFactors = TRUE‘ converts character columns into factors automatically. This is useful when dealing with categorical data.

1.3 Understanding the Dataset

- Use `str()` to understand the structure of a data frame, and `summary()` to generate basic descriptive statistics:

```
# Check the structure of the data
str(cars)

# Summary of the data
summary(cars)
```

- **Explanation**: `str()` provides an overview of the data types and the first few entries for each column, while `summary()` calculates the mean, median, quartiles, and missing values for numeric columns.

1.4 Variable Transformation

- Convert variables to factors for categorical analysis:

```
# Convert year and number of cylinders to factors
cars$year <- as.factor(cars$year)
cars$cylinders <- as.factor(cars$cylinders)

# Updated summary
summary(cars)
```

- **Concept**: Factors are used to handle categorical data in R, allowing for easier data manipulation and visualization.

1.5 Exploring the Data: Visualizations

1.5.1 Bar Plots for Categorical Variables

- Visualize the distribution of categorical variables using `barplot()`:

```
# Bar plot for year
barplot(summary(cars$year), ylab = "Frequency", main = "Distribution of
    Cars by Year")

# Bar plot for brand
barplot(summary(cars$brand), main = "Distribution of Cars by Brand")
```

- **Concept**: Bar plots display the frequency of each category, making them ideal for understanding distributions of categorical variables.

1.5.2 Contingency Tables and Proportions

- Create contingency tables to understand joint distributions:

```
# Reclassify 'year' into categories
cars$year <- as.numeric(as.character(cars$year))
cars$year.cat <- cut(cars$year, breaks = c(1970, 1975, 1979, 1983))

# Contingency table of year and brand
t <- table(cars$year.cat, cars$brand)
```

- **Proportions**:

```
# Overall proportions
prop.table(t)

# Row-wise proportions
prop.table(t, 1)

# Column-wise proportions
prop.table(t, 2)
```

- **Explanation**: Proportions help in understanding how categories relate to one another, providing insight into patterns within the data.

1.6 Visualizing Quantitative Variables

1.6.1 Histograms for Continuous Variables

- Use `hist()` to visualize distributions of continuous variables:

```
# Visualize distributions
hist(cars$mpg, main = "Distribution of MPG", xlab = "Miles per Gallon")
hist(cars$hp, main = "Distribution of Horsepower", xlab = "Horsepower")
hist(cars$weightlbs, main = "Distribution of Weight", xlab = "Weight (lbs)"
   )
```

- **Best Practice**: Customize histograms with labels and titles to improve clarity.

1.6.2 Boxplots for Comparing Distributions

- Boxplots are ideal for comparing distributions across different categories:

```
# Compare weight distributions across brands
boxplot(cars$weightlbs ~ cars$brand, main = "Weight Distribution by Brand",
        ylab = "Weight (lbs)", xlab = "Brand")
```

- **Interpretation**: Boxplots show the median, quartiles, and potential outliers, providing a concise summary of distributional characteristics.

1.7 Descriptive Statistics

- Calculate descriptive statistics (mean, median, standard deviation) for selected variables:

```
# Calculate mean, median, and standard deviation
mean_values <- apply(cars[, c(1, 4, 5)], 2, mean, na.rm = TRUE)
median_values <- apply(cars[, c(1, 4, 5)], 2, median, na.rm = TRUE)
std_dev <- apply(cars[, c(1, 4, 5)], 2, sd, na.rm = TRUE)

# Create a descriptive statistics table
tab <- rbind(mean_values, median_values, std_dev)
rownames(tab) <- c("Mean", "Median", "Standard Deviation")
tab
```

- **Concept**: Use the `apply()` function to compute summary statistics across columns efficiently.

1.8 Advanced Table Operations

- Use `tapply()` to compute group-wise summaries:

```
# Grouped summary of weight by brand
tapply(cars$weightlbs, cars$brand, summary)
```

- **Key Insight**: `tapply()` simplifies the process of calculating statistics for each level of a categorical variable.

1.9 Conclusion

This introductory guide has covered essential techniques for importing, exploring, visualizing, and summarizing data in R. Understanding these foundational tools is critical for more complex analyses, as they allow you to efficiently preprocess and analyze datasets, whether for multivariate analysis, data mining, or machine learning.

Chapter 2

Data Analysis and Visualisation

2.1 Introduction

This guide covers techniques for handling missing data, performing imputation, and visualizing various types of data. Understanding how to manage and analyze incomplete datasets is crucial in applied statistics and data science, especially when working with real-world data that often has missing or messy entries.

2.2 Handling Missing Data

2.2.1 Inspecting Missing Data

- Use the ‘datasets‘ package to access built-in datasets like `airquality`.
- Calculate the proportion of missing data in each variable:

```
library(datasets)
data("airquality")

# Proportion of missing observations
for (i in 1:6){
  print(sum(is.na(airquality[,i]))/length(airquality[,i])*100)
}
```

- **Interpretation**: Identifying variables with high proportions of missing data is the first step in deciding whether to impute, remove, or analyze them separately.

2.2.2 Omitting NA Values

- Omit missing values using `na.omit()`:

```
air_omit <- na.omit(airquality)
summary(air_omit)

# Visualize pairwise relationships
plot(air_omit[,1:4])
```

- **Key Insight**: Use omission only when the proportion of missing data is minimal, as dropping rows may result in information loss and biased analyses.

2.2.3 Visualizing Missing Data Patterns

- Use the `mice` and `VIM` packages to visualize missing data patterns:

```r
install.packages("mice")
library(mice)

# Visualize missing data patterns
md.pattern(airquality)

install.packages("VIM")
library(VIM)
aggr(airquality)  # Aggregated plot of missing values
```

- **Best Practice**: Visualizations such as aggregated bar plots or missing data patterns help determine which variables have systematic missingness.

2.3 Data Imputation

2.3.1 Mean Imputation

- Replace missing values with the variable mean:

```r
install.packages("Hmisc")
library(Hmisc)

airquality_imp <- airquality
airquality_imp$Ozone <- with(airquality_imp, impute(Ozone, mean))

# Verify imputation
summary(airquality_imp$Ozone)
```

- **Limitations**: Mean imputation reduces variance and may lead to biased estimates, especially for non-normally distributed variables.

2.3.2 Multivariate Imputation with `mice`

- Impute using multiple iterations to capture uncertainty:

```r
library(mice)

# Mean imputation with one iteration
imp_mice <- mice(airquality, method = "mean", m = 1, maxit = 1)
complete_data <- complete(imp_mice)

# Compare distributions
plot(density(airquality$Ozone, na.rm = TRUE), ylim = c(0, 0.025))
lines(density(complete_data$Ozone), col = 2)
```

- **Practical Tip**: Use `m = 5` or more for robust imputations. Visualize imputed vs. original distributions to ensure consistency.

2.3.3 Regression-Based Imputation

- Impute using linear regression:

```r
reg_imp <- mice(airquality, method = "norm.predict", m = 1)
air_reg_imp <- complete(reg_imp)

# Visualize imputed values
hist(airquality$Ozone)
hist(reg_imp$imp$Ozone[,1], add = TRUE, col = 2)
```

- **Note**: Regression imputation preserves relationships but may artificially inflate R-squared values.

2.4 Poverty Study Data: Missing Data Codifications

- Handle non-standard missing value codifications (e.g., -99):

```
library(readr)
pov_data <- read_table("PovertyStudy.dat")
pov_data$GNP[pov_data$GNP == -99] <- NA

# Replace with median
pov_data$GNP[is.na(pov_data$GNP)] <- median(pov_data$GNP, na.rm = TRUE)
```

- **Best Practice**: Convert placeholders to NA and then impute to avoid misleading results.

2.5 Data Transformation: Box-Cox Transformation

- Use Box-Cox transformation to stabilize variance:

```
library(MASS)

# Box-Cox transformation on GNP
boxcox(pov_data$GNP ~ 1, lambda = seq(-1, 1, by = 0.1))

# Log transformation
lGNP <- log(pov_data$GNP)
```

- **Tip**: Choose the transformation that minimizes skewness and maximizes normality.

2.6 Data Visualization Techniques

2.6.1 Visualization of Numerical Variables

- Use **Chernoff Faces** to visualize multidimensional data:

```
library(aplpack)
data(longley)
faces(longley[1:9,], face.type = 0)
```

- **Application**: Useful for detecting patterns in small datasets with multiple variables.

2.6.2 Correlation Plots

- Visualize correlation matrices with `corrplot`:

```
install.packages("corrplot")
library(corrplot)

cr <- cor(pov_data[, 1:6])
corrplot(cr)
```

- **Tip**: Use the `method = "number"` argument to show correlation values directly.

2.7 Visualizing Categorical Data

2.7.1 Bar Plots and Mosaic Plots

- Create bar plots for categorical variables:

```
barplot(prop.table(tab), col = c("red", "blue"),
        legend.text = c("vshaped", "straight"),
        main = "Distribution of Satisfaction Level",
        ylim = c(0, 0.7))
```

- Use mosaic plots for visualizing interactions:

```r
library(RColorBrewer)

# Define pastel colors
pastel_colors <- brewer.pal(3, "Pastel1")
mosaicplot(tab, main = "Transmission Type vs. Engine Shape", color = pastel
    _colors)
```

2.8 Summary Tables and Factor Data

- Cross-tabulations and probability tables:

```r
# Conditional probabilities by row
prop.table(tab, 1)

# Joint and marginal probabilities
prop.table(tab)
```

Chapter 3

Principal Component Analysis

3.1 Overview of PCA and Factor Analysis

Principal Component Analysis (PCA) and **Factor Analysis (FA)** are both multivariate techniques aimed at dimensionality reduction, but with different goals:

- **PCA** focuses on maximizing the variance captured by each component, making it ideal for visualizing high-dimensional data and identifying major sources of variation.
- **FA** seeks to identify latent factors that explain observed variable correlations, making it more suitable for uncovering hidden structures or constructs within the data.

3.1.1 Choosing Between PCA and FA

- **Use PCA when**:
  - You aim to capture as much variance as possible.
  - Data is numeric and relationships are linear.
  - You are more interested in reducing dimensionality for visualization purposes.
- **Use FA when**:
  - The goal is to identify underlying latent variables.
  - You want to explore hidden structures or groupings within the data.
  - Data might be non-normal or categorical, requiring alternative approaches like polychoric FA.

3.2 PCA on USArrests Data using `princomp`

- `princomp()` uses spectral decomposition on the covariance or correlation matrix.
- The choice between using a correlation matrix (`cor = TRUE`) or covariance matrix depends on whether the data needs to be standardized.
- Example:

```r
data <- USArrests
pcUSA <- princomp(data, cor = TRUE)
summary(pcUSA)

# Eigenvalues of the principal components
eigs <- pcUSA$sdev^2
plot(eigs, type = "b", main = "Scree Plot")

# Loading matrix and biplot
pcUSA$loadings
biplot(pcUSA)
```

- **Interpretation**:
  - The **Scree plot** helps visualize the proportion of variance explained by each component.
  - `pcUSA$loadings` provides the contribution of each variable to the components.
  - The first few components should ideally capture the majority of the variance.

3.2.1 PCA using `prcomp`

- `prcomp()` is preferred over `princomp()` when the dataset is large, as it uses Singular Value Decomposition (SVD).
- It provides a numerically stable solution, especially when dealing with highly correlated variables.
- Example:

```
prusa <- prcomp(data, scale. = TRUE)
summary(prusa)

# Eigenvalues
prusa$sdev^2

# Biplot to visualize the scores and loadings
biplot(prusa)
```

- **Key Consideration**:
  - Use ‘prcomp()‘ when numerical stability is a concern, especially for high-dimensional datasets.

3.3 Factor Analysis using `FactoMineR`

- `FactoMineR` is a versatile package for advanced factor analysis and PCA.
- It provides additional outputs like communalities, cosines, and variable contributions.
- Example on `USArrests`:

```
library(FactoMineR)
usafa <- PCA(USArrests)
summary(usafa)

# Accessing eigenvalues and variable loadings
usafa$eig
usafa$var$coord
```

- **Practical Tip**:
  - Use `FactoMineR` when you need to assess multiple dimensions, visualize variable contributions, or perform hierarchical clustering on principal components.

3.4 Bartlett’s Test and KMO Index using `psych`

- **Bartlett’s Test of Sphericity** tests if the correlation matrix is an identity matrix, indicating if the variables are sufficiently correlated to perform factor analysis.
- Example using the Food Price dataset:

```
library(psych)
R <- cor(food)
cortest.bartlett(R, n = nrow(food))
```

- **Kaiser-Meyer-Olkin (KMO) Index** assesses sampling adequacy.

  - Values range from 0 to 1, with values above 0.6 indicating that the data is factorable.

```
kmo(food)
```

- **Interpretation of KMO**:
  - KMO > 0.9: Excellent
  - $0.7 >$ KMO > 0.9: Good to average
  - KMO < 0.5: Unacceptable

3.5 Advanced PCA on Food Price Data

- **Test of Factorability** is essential before PCA to ensure that the variables are correlated.
- PCA is performed using the correlation matrix when variables are on different scales.

```
pca_food <- princomp(food, cor = TRUE)
summary(pca_food)

# Loadings and visualization
plot(pca_food, type = "lines", main = "Scree Plot for Food Prices")
biplot(pca_food)
```

- **Interpretation**:
  - If the first two components explain a significant proportion of variance (e.g., ¿70%), use them for further analysis.
  - Biplots provide insight into how variables contribute to each component.

3.6 Component and Factor Loadings

- **Factor Loadings** indicate how much a variable contributes to a factor. Higher loadings suggest a stronger relationship with the factor.
- Example:

```
pcUSA$loadings
```

- **Communalities** measure the proportion of variance a variable shares with all components:

```
com1 <- (pcUSA$loadings[1,]*sqrt(eigs))^2
sum(com1[1:2])  # Sum of communalities for the first two components
```

3.7 Factor Analysis of Heptathlon Data

- Factor Analysis is performed on the `heptathlon` dataset, taking into account different directions for specific variables.
- Example:

```
data("heptathlon", package = "HSAUR")
pca_hep <- PCA(heptathlon, quanti.sup = 8)
plot(pca_hep$eig[,1], type = "o", main = "Scree Plot for Heptathlon Data")
```

- **Practical Tip**:
  - Supplementary variables and individuals (`ind.sup`, `quanti.sup`) can be included to analyze additional points without affecting the main analysis.

Chapter 4

Multidimensional Scaling

4.1 Introduction to Multidimensional Scaling (MDS)

Multidimensional Scaling (MDS) is a set of methods for visualizing the structure of distance data in a lower-dimensional space. The purpose is to project high-dimensional dissimilarities into a 2D or 3D plane while retaining as much information about their relative similarities as possible.

4.1.1 Key Types of MDS

There are two primary types of MDS based on the treatment of dissimilarities:

- **Metric MDS**: Preserves the actual dissimilarity values, making it suitable for data where the numerical distance has a direct interpretation.
- **Non-Metric MDS**: Focuses on preserving the rank order of dissimilarities rather than exact values. This is useful for data where only the order of similarity matters (e.g., preference rankings).

4.1.2 Choosing Between Metric and Non-Metric MDS

The decision between Metric and Non-Metric MDS often depends on whether the distance matrix contains meaningful magnitudes. For example:

- Use **Metric MDS** when:
  - You have continuous data with interpretable Euclidean distances.
  - Exact distances matter more than their order.
- Use **Non-Metric MDS** when:
  - Dealing with ordinal or categorical data.
  - Only the relative ranking of distances is meaningful.
  - You suspect that a non-linear relationship might better capture the patterns in the data.

4.2 Distance Calculations using the `cluster` Package

4.2.1 Understanding the `daisy()` Function

- The `daisy()` function is highly versatile for computing distance matrices in R, particularly for mixed-type data.
- **Available Distance Metrics**:
  - **Euclidean Distance**: Measures straight-line distance and assumes continuous, numeric data.
  - **Gower Distance**: Handles mixed data types (numeric, ordinal, and categorical).
- **Example**:

```
library(cluster)
data("iris")

# Calculate Euclidean distance
euclidean_dist <- daisy(iris[, -5], metric = "euclidean")

# Calculate Gower distance
gower_dist <- daisy(iris[, -5], metric = "gower")
```

- **Practical Consideration**:
  - `daisy()` is often preferable over `dist()` when working with data frames containing non-numeric variables.
  - Consider scaling your data when using Euclidean distances to avoid variables with larger scales dominating the result.

4.3 Principal Coordinates Analysis using `labdsv`

- **Principal Coordinates Analysis (PCoA)** is a linear method that represents dissimilarities in Euclidean space.
- Ideal for visualizing data with a known distance matrix.
- Example using the `eurodist` dataset:

```
library("labdsv")

# Perform Principal Coordinates Analysis (PCoA)
pco_eu <- pco(eurodist, k = 2)

# Visualization
plot(pco_eu$points[,1], pco_eu$points[,2], type = "n", main = "PCoA of
    European Cities")
text(pco_eu$points[,1], pco_eu$points[,2], labels(eurodist), cex = 0.8)
```

4.4 Classical MDS using `cmdscale()` from the `stats` Package

- Classical MDS (‘cmdscale‘) is ideal when the goal is to retain the true Euclidean distances.
- Computes a configuration that best fits the input dissimilarity matrix.
- Example:

```
# Perform Classical MDS on eurodist dataset
mds_eu <- cmdscale(eurodist, eig = TRUE)

# Plotting the MDS solution
plot(mds_eu$points[,1], mds_eu$points[,2], main = "Classical MDS", type = "
    n")
text(mds_eu$points[,1], mds_eu$points[,2], labels(eurodist), cex = 0.8)
```

- **Interpretation**:
  - The first two components typically capture the majority of the variance.
  - The eigenvalues indicate the variance explained by each dimension.
  - Negative eigenvalues suggest poor fit and indicate that the distance matrix might not be truly Euclidean.

4.5 Non-Metric MDS using `isoMDS()` from the `MASS` Package

- Non-Metric MDS (‘isoMDS‘) preserves the rank order of dissimilarities.
- Useful for understanding non-linear relationships in the data.

```r
library(MASS)

# Perform Non-Metric MDS
usa_nmds <- isoMDS(dist(USArrests))

# Visualization
plot(usa_nmds$points, main = "Non-Metric MDS on USArrests", type = "n")
text(usa_nmds$points, labels = row.names(USArrests), cex = 0.8)
```

- **Interpreting Stress Values**:
  - The ‘stress‘ value indicates how well the 2D configuration matches the original dissimilarities.
  - Lower stress values (< 0.1) indicate a good fit, while higher values suggest distortions.

4.6 Non-Linear Mapping with `sammon()`

- The `sammon()` function in R performs a form of non-metric MDS, but it emphasizes preserving local structures.

```r
nmds_sam <- sammon(dist(USArrests))
plot(nmds_sam$points, main = "Sammon Mapping on USArrests", type = "n")
text(nmds_sam$points, labels = row.names(USArrests))
```

- **When to Use Sammon Mapping**:
  - When maintaining local neighborhood distances is more important than preserving global structure.

4.7 3D Visualizations using the `rgl` Package

- For visualizing MDS results in 3D, use the ‘rgl‘ package:

```r
library(rgl)

# 3D Visualization
plot3d(mds_eu$points[,1:3], main = "3D MDS Visualization of European Cities
   ")
```

- **Practical Tip**: Use ‘rgl‘ to interactively rotate and explore the 3D configuration.

4.8 Advanced Considerations: Goodness-of-Fit Analysis

- Use the sum of squared eigenvalues and stress values to evaluate how well the MDS solution represents the original data.
- Example:

```r
sum(mds_eu$eig[1:2]) / sum(mds_eu$eig)  # Proportion of variance explained
   by first 2 dimensions
```

- **Key Insight**: A higher proportion suggests that the first two dimensions capture most of the variance, making the 2D plot a good representation.