Assignment 1 Walkthrough

Warren Mansur

Purpose of Walkthrough

- The goal of this walkthrough is to help you successfully understand and complete Assignment 1.

Purpose of Walkthrough

- The goal of this walkthrough is to help you successfully understand and complete Assignment 1.
- For each subproblem in the assignment, I briefly recap the concept, then show you code examples similar to those required by the assignment.

Purpose of Walkthrough

- The goal of this walkthrough is to help you successfully understand and complete Assignment 1.
- For each subproblem in the assignment, I briefly recap the concept, then show you code examples similar to those required by the assignment.
- These code examples are not shown elsewhere (i.e. not in the instructor's live class, shared example code, or online modules).

Important Reminders

- It's important to attend Prof. Joner's live classrooms to learn the core concepts. The sessions are thorough and deep.

Important Reminders

- It's important to attend Prof. Joner's live classrooms to learn the core concepts. The sessions are thorough and deep.
- This walkthrough recaps some concepts to focus on the practical application for the assignment, but does not go into the same conceptual depth.

Important Reminders

- It's important to attend Prof. Joner's live classrooms to learn the core concepts. The sessions are thorough and deep.
- This walkthrough recaps some concepts to focus on the practical application for the assignment, but does not go into the same conceptual depth.
- Prof. Joner provides one sample R file per week to get you started. They demonstrate use of some important libraries and functions, but still require you to apply the code to the specific assignment scenarios.

Submitting

- It's important to provide two files – one for your answers without the code, and the other with the code.

Submitting

- It's important to provide two files – one for your answers without the code, and the other with the code.
- This keeps your work and answers separate, helping ensure that you receive credit where it is due.

Submitting

- It's important to provide two files – one for your answers without the code, and the other with the code.
- This keeps your work and answers separate, helping ensure that you receive credit where it is due.
- There have been occasions in past runnings where the work over-cluttered the answers, and a few lost some points because of misidentification.

Submitting

- It's important to provide two files – one for your answers without the code, and the other with the code.
- This keeps your work and answers separate, helping ensure that you receive credit where it is due.
- There have been occasions in past runnings where the work over-cluttered the answers, and a few lost some points because of misidentification.
- Use this YAML to hide code globally:

```yaml
# ----- in the document YAML -----
execute:
  echo:    false   # hide code globally
  warning: false   # hide warnings globally
```

Recap: Categorical Feature Type

- Two broad types of features are categorical and numeric.

Recap: Categorical Feature Type

- Two broad types of features are categorical and numeric.
- A categorical feature's values act as labels for group membership.

Recap: Categorical Feature Type

- Two broad types of features are categorical and numeric.
- A categorical feature's values act as labels for group membership.
- A categorical value doesn't express magnitude or distance—arithmetic operations and means are meaningless.

Recap: Categorical Feature Type

- Two broad types of features are categorical and numeric.
- A categorical feature's values act as labels for group membership.
- A categorical value doesn't express magnitude or distance—arithmetic operations and means are meaningless.
- Examples include name, zip code, yes/no, and low/medium/high.

Recap: Categorical Feature Type

- Two broad types of features are categorical and numeric.
- A categorical feature's values act as labels for group membership.
- A categorical value doesn't express magnitude or distance—arithmetic operations and means are meaningless.
- Examples include name, zip code, yes/no, and low/medium/high.
- There are three subtypes—nominal, binary, and ordinal.

Recap: Categorical Subtypes

- **Nominal** values are unordered categories (e.g., name, address, zip code). The values are just labels with no built-in order or numerical distance.

Recap: Categorical Subtypes

- **Nominal** values are unordered categories (e.g., name, address, zip code). The values are just labels with no built-in order or numerical distance.
- **Binary** values have exactly two possible categories, such as yes/no or 1/0.

Recap: Categorical Subtypes

- **Nominal** values are unordered categories (e.g., name, address, zip code). The values are just labels with no built-in order or numerical distance.
- **Binary** values have exactly two possible categories, such as yes/no or 1/0.
- **Ordinal** values are categories with an inherent order (e.g., low $<$ medium $<$ high). They remain labels, but analyses can leverage their ordering (though not a true numeric distance).

Recap: Numeric Feature Type

- Numeric values are valid quantities.

Recap: Numeric Feature Type

- Numeric values are valid quantities.
- Most classical statistics assume numeric values (mean, median, correlation, distance, etc.).

Recap: Numeric Feature Type

- Numeric values are valid quantities.
- Most classical statistics assume numeric values (mean, median, correlation, distance, etc.).
- Examples include age, height, weight, and income.

Recap: Numeric Feature Type

- Numeric values are valid quantities.
- Most classical statistics assume numeric values (mean, median, correlation, distance, etc.).
- Examples include age, height, weight, and income.
- Numeric features have two subtypes—discrete and continuous.

Recap: Numeric Feature Type

- Numeric values are valid quantities.
- Most classical statistics assume numeric values (mean, median, correlation, distance, etc.).
- Examples include age, height, weight, and income.
- Numeric features have two subtypes—discrete and continuous.
- **Discrete** values are integers (e.g., age or number_of_cylinders) without fractional parts.

Recap: Numeric Feature Type

- Numeric values are valid quantities.
- Most classical statistics assume numeric values (mean, median, correlation, distance, etc.).
- Examples include age, height, weight, and income.
- Numeric features have two subtypes—discrete and continuous.
- **Discrete** values are integers (e.g., age or number_of_cylinders) without fractional parts.
- **Continuous** values can take any real number within a range, including fractional values (e.g., height and weight).

Problem 1.1 Recap

(1) Calculate the mean, median, and sample standard deviation (sample)

Problem 1.1 Recap

(1) Calculate the mean, median, and sample standard deviation (sample)

- **Mean** – arithmetic average; add all values and divide by n.

Problem 1.1 Recap

(1) Calculate the mean, median, and sample standard deviation (sample)

- **Mean** – arithmetic average; add all values and divide by n.
- **Median** – middle value (or mean of the middle two) when ordered.

Problem 1.1 Recap

(1) Calculate the mean, median, and sample standard deviation (sample)

- **Mean** – arithmetic average; add all values and divide by n.
- **Median** – middle value (or mean of the middle two) when ordered.
- **Variance** – average squared deviation from the mean; squaring enlarges big gaps.

Problem 1.1 Recap

(1) Calculate the mean, median, and sample standard deviation (sample)

- **Mean** – arithmetic average; add all values and divide by n.
- **Median** – middle value (or mean of the middle two) when ordered.
- **Variance** – average squared deviation from the mean; squaring enlarges big gaps.
- **Sample SD** – square root of the variance; principal dispersion measure.

Problem 1.1 Code and Result

```r
# Load the employee wellbeing survey and compute mean, median, and SD of years of experience.
# read.csv(): reads a CSV file; "employee_wellbeing_survey.csv" is the file path.
d <- read.csv("employee_wellbeing_survey.csv")

# d$years_experience: selects the years_experience column for analysis.
# mean(): returns the arithmetic mean; na.rm = TRUE discards NA values.
# median(): computes the sample median; na.rm = TRUE discards NA values.
# sd(): gives the sample standard deviation (uses n − 1); na.rm = TRUE discards NA values.
mean_exp   <- mean(d$years_experience,   na.rm = TRUE)
median_exp <- median(d$years_experience, na.rm = TRUE)
sd_exp     <- sd(d$years_experience,     na.rm = TRUE)

# c(): concatenates results into a named vector for clear printing.
# mean_exp, median_exp, sd_exp: statistics calculated above.
c(mean   = mean_exp,
  median = median_exp,
  sd     = sd_exp)
```

```
   mean  median      sd
8.78875 8.00000 4.76524
```

Problem 1.2 Recap

(2) Determine Q1, Q2, and Q3 of the *age* feature and draw its boxplot.

Problem 1.2 Recap

(2) Determine Q1, Q2, and Q3 of the *age* feature and draw its boxplot.

- **Quartile Q1 (25th percentile)** – value below which 25 percent of observations fall; marks the lower edge of the middle half of the data.

Problem 1.2 Recap

(2) Determine Q1, Q2, and Q3 of the *age* feature and draw its boxplot.

- **Quartile Q1 (25th percentile)** – value below which 25 percent of observations fall; marks the lower edge of the middle half of the data.
- **Quartile Q2 (Median, 50th percentile)** – midpoint of ordered data; splits the dataset into two equal halves.

Problem 1.2 Recap

(2) Determine Q1, Q2, and Q3 of the *age* feature and draw its boxplot.

- **Quartile Q1 (25th percentile)** – value below which 25 percent of observations fall; marks the lower edge of the middle half of the data.
- **Quartile Q2 (Median, 50th percentile)** – midpoint of ordered data; splits the dataset into two equal halves.
- **Quartile Q3 (75th percentile)** – value below which 75 percent of observations fall; marks the upper edge of the middle half of the data.

Problem 1.2 Code and Result

```r
# d$years_experience: selects the years_experience column for analysis.
# quantile(): returns sample quantiles; probs = c(0.25, 0.50, 0.75) requests Q1, Q2 (median), and Q3.
# na.rm = TRUE discards any missing values before calculation.
quart_exp <- quantile(d$years_experience,
                      probs = c(0.25, 0.50, 0.75),
                      na.rm = TRUE)

# Store the individual quartile statistics in clearly named variables for easy reference.
Q1_exp <- quart_exp[1]   # first quartile (25th percentile)
Q2_exp <- quart_exp[2]   # second quartile (median / 50th percentile)
Q3_exp <- quart_exp[3]   # third quartile (75th percentile)

# c(): concatenates results into a named vector for clear printing.
# Q1_exp, Q2_exp, Q3_exp: quartile statistics calculated above.
c(Q1 = Q1_exp,
  Q2 = Q2_exp,
  Q3 = Q3_exp)
```

```
Q1.25% Q2.50% Q3.75%
     5      8     12
```

Problem 1.3 Recap

(3) Plot a boxplot of the *age* feature.

Problem 1.3 Recap

(3) Plot a boxplot of the *age* feature.

- **Boxplot purpose** – visually condenses a variable's distribution into its five-number summary (minimum, Q1, median, Q3, maximum) for quick comparison and outlier detection.

Problem 1.3 Recap

(3) Plot a boxplot of the *age* feature.

- **Boxplot purpose** – visually condenses a variable's distribution into its five-number summary (minimum, Q1, median, Q3, maximum) for quick comparison and outlier detection.
- **Median line** – the horizontal line inside the box marks the data's center; if it leans toward either edge, that signals skewness.

Problem 1.3 Recap

(3) Plot a boxplot of the *age* feature.

- **Boxplot purpose** – visually condenses a variable's distribution into its five-number summary (minimum, Q1, median, Q3, maximum) for quick comparison and outlier detection.
- **Median line** – the horizontal line inside the box marks the data's center; if it leans toward either edge, that signals skewness.
- **Box (Q1–Q3)** – its height equals the interquartile range (IQR), showing the spread of the middle 50 percent; a taller box means greater variability among typical ages.

Problem 1.3 Recap

(3) Plot a boxplot of the *age* feature.

- **Boxplot purpose** – visually condenses a variable's distribution into its five-number summary (minimum, Q1, median, Q3, maximum) for quick comparison and outlier detection.
- **Median line** – the horizontal line inside the box marks the data's center; if it leans toward either edge, that signals skewness.
- **Box (Q1–Q3)** – its height equals the interquartile range (IQR), showing the spread of the middle 50 percent; a taller box means greater variability among typical ages.
- **Whiskers** – extend to the most extreme values still within $1.5 \times$ IQR of the box; they hint at overall range without giving outliers undue visual weight.

Problem 1.3 Recap

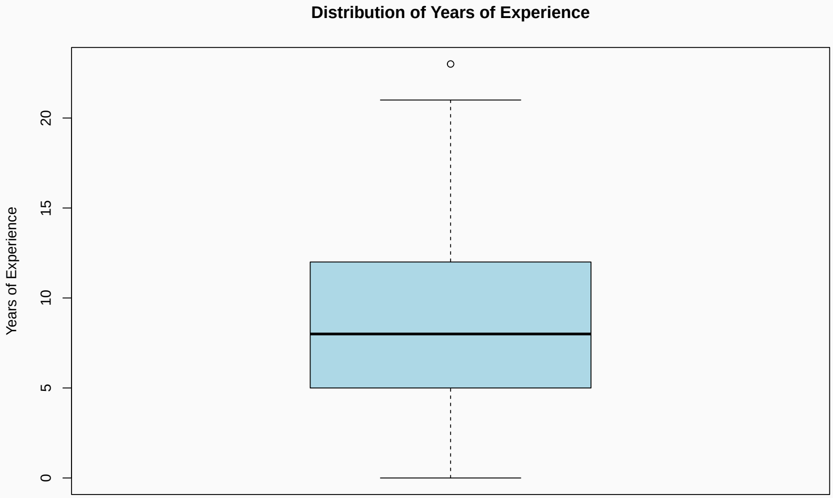
(3) Plot a boxplot of the *age* feature.

- **Boxplot purpose** – visually condenses a variable's distribution into its five-number summary (minimum, Q1, median, Q3, maximum) for quick comparison and outlier detection.
- **Median line** – the horizontal line inside the box marks the data's center; if it leans toward either edge, that signals skewness.
- **Box (Q1–Q3)** – its height equals the interquartile range (IQR), showing the spread of the middle 50 percent; a taller box means greater variability among typical ages.
- **Whiskers** – extend to the most extreme values still within $1.5 \times$ IQR of the box; they hint at overall range without giving outliers undue visual weight.
- **Outliers** – points beyond the whiskers are plotted individually; investigate them to decide whether they reflect rare but valid values or data-entry errors.

Problem 1.3 Code

```r
# d$years_experience: selects the numeric feature we want to summarize graphically.
# boxplot(): draws a box-and-whisker plot (five-number summary + outliers).
# main: plot title for context.  ylab: axis label for clarity.
# col: light blue fill improves readability; notch = FALSE gives the classic rectangular box.
# na.rm = TRUE: ignores any missing values so they do not distort the plot.
boxplot(d$years_experience,
        main = "Distribution of Years of Experience",
        ylab = "Years of Experience",
        col = "lightblue",
        notch = FALSE,
        na.rm = TRUE)
```

Problem 1.3 Result



Problem 1.4 Recap

(4) Implement min–max rescaling on the *age* feature, replace the original column, and report the rescaled value for the seventh observation.

Problem 1.4 Recap

(4) Implement min–max rescaling on the *age* feature, replace the original column, and report the rescaled value for the seventh observation.

- **Min–max rescaling (normalization)** – linearly maps every value from its original range ([,min,,max,]) into a chosen interval, typically ([0,1]), without altering the relative ordering.

Problem 1.4 Recap

(4) Implement min–max rescaling on the *age* feature, replace the original column, and report the rescaled value for the seventh observation.

- **Min–max rescaling (normalization)** – linearly maps every value from its original range ([,min,,max,]) into a chosen interval, typically ([0,1]), without altering the relative ordering.
- **Formula** – v' = (v - min) / (max - min); yields 0 for the minimum age, 1 for the maximum age, and a proportionate score for everything in between.

Problem 1.4 Recap

(4) Implement min–max rescaling on the *age* feature, replace the original column, and report the rescaled value for the seventh observation.

- **Min–max rescaling (normalization)** – linearly maps every value from its original range ([,min,,max,]) into a chosen interval, typically ([0,1]), without altering the relative ordering.
- **Formula** – v' = (v - min) / (max - min); yields 0 for the minimum age, 1 for the maximum age, and a proportionate score for everything in between.
- **Purpose** – puts features on the same numeric scale so distance-based or gradient-based algorithms (e.g., k-NN, neural networks, K-means) treat each variable fairly rather than letting larger-magnitude attributes dominate.

Problem 1.4 Recap

(4) Implement min–max rescaling on the *age* feature, replace the original column, and report the rescaled value for the seventh observation.

- **Min–max rescaling (normalization)** – linearly maps every value from its original range ([,min,,max,]) into a chosen interval, typically ([0,1]), without altering the relative ordering.
- **Formula** – v' = (v - min) / (max - min); yields 0 for the minimum age, 1 for the maximum age, and a proportionate score for everything in between.
- **Purpose** – puts features on the same numeric scale so distance-based or gradient-based algorithms (e.g., k-NN, neural networks, K-means) treat each variable fairly rather than letting larger-magnitude attributes dominate.
- **Shape preservation** – because the transformation is linear, the distribution's overall shape and relative spacing remain intact; only the axis scale changes.

Problem 1.4 Code and Result

```r
# d$years_experience: selects the numeric feature to normalize.
# min() and max(): find the observed minimum and maximum; na.rm = TRUE skips missing values.
min_yrs <- min(d$years_experience, na.rm = TRUE)   # minimum years of experience
max_yrs <- max(d$years_experience, na.rm = TRUE)   # maximum years of experience

# Implement min-max rescaling to the [0, 1] interval with the classic formula:
# (x - min) / (max - min).  Replace the original column with the rescaled values.
d$years_experience <- (d$years_experience - min_yrs) / (max_yrs - min_yrs)

# Alternative one-liner mirroring the Module 1 script:
# The scales package provides utilities to map, transform, and format numeric,
# date-time, and categorical data.
# d$years_experience <- scales::rescale(d$years_experience)   # requires library(scales)

# Provide the rescaled years_experience for the seventh observation (row 7).
d$years_experience[7]
```

```
[1] 0.5652174
```

Problem 1.5 Recap

(5) Determine the mode of the *country_of_res* feature.

Problem 1.5 Recap

(5) Determine the mode of the *country_of_res* feature.

- **Mode** – the category (or categories) that appears most often in a dataset; for `country_of_res` it indicates the most common country of residence.

Problem 1.5 Recap

(5) Determine the mode of the *country_of_res* feature.

- **Mode** – the category (or categories) that appears most often in a dataset; for `country_of_res` it indicates the most common country of residence.
- **Computation** – count how many times each unique country occurs, ignoring missing values; the highest count marks the mode.

Problem 1.5 Recap

(5) Determine the mode of the *country_of_res* feature.

- **Mode** – the category (or categories) that appears most often in a dataset; for `country_of_res` it indicates the most common country of residence.
- **Computation** – count how many times each unique country occurs, ignoring missing values; the highest count marks the mode.
- **Interpretive value** – captures the "typical" category for nominal data where mean and median are not meaningful.

Problem 1.5 Code

```r
# Convert any empty strings ("") to real NA values so they are treated as missing.
# The dplyr package supplies tabular data-manipulation including filtering, selecting,
# mutating, summarizing, arranging, and joining.
d$professional_certification <- dplyr::na_if(d$professional_certification, "")

# d$work_location: selects the professional_certification column (categorical) for analysis.
# table(): tabulates frequencies of each unique value; useNA = "no" excludes missing values.
freq_loc <- table(d$professional_certification, useNA = "no")

# The modeest package supplies several functions to estimate the statistical
# mode of a univariate data set
# modeest::mfv(): returns the most frequent value(s) (mode) of a vector.
# na_rm = TRUE discards NA values before calculation.
library(modeest)
mode_loc <- modeest::mfv(d$professional_certification, na_rm = TRUE)

# list(): combines the frequency table and the mode into a single named object for clear printing.
# freq_loc, mode_loc: results created above.
list(frequencies = freq_loc,
     mode        = mode_loc)
```

Problem 1.5 Result

```
$frequencies

     AWS SA  Azure Admin         CCNA          PMP Scrum Master
        130          124          122          110          121

$mode
[1] "AWS SA"
```

Problem 1.6 Recap

(6) Review the **ethnicity** feature. You will notice several missing values in this feature. Determine a reasonable imputation for this feature, explain what you are going to do and why, then replace the original ethnicity feature with the imputed result.

Problem 1.6 Recap

(6) Review the **ethnicity** feature. You will notice several missing values in this feature. Determine a reasonable imputation for this feature, explain what you are going to do and why, then replace the original ethnicity feature with the imputed result.

- Blank/NA entries reduce completeness and can bias any statistics or models drawn from the data. They can arise from data-entry omissions, equipment errors, or decisions not to record certain information.

Problem 1.6 Recap

(6) Review the **ethnicity** feature. You will notice several missing values in this feature. Determine a reasonable imputation for this feature, explain what you are going to do and why, then replace the original ethnicity feature with the imputed result.

- Blank/NA entries reduce completeness and can bias any statistics or models drawn from the data. They can arise from data-entry omissions, equipment errors, or decisions not to record certain information.
- **Imputation goal** – restore completeness without distorting the distribution. For nominal (unordered) categories, one common strategy is mode imputation, filling each NA with the most frequent.

Problem 1.6 Recap

(6) Review the **ethnicity** feature. You will notice several missing values in this feature. Determine a reasonable imputation for this feature, explain what you are going to do and why, then replace the original ethnicity feature with the imputed result.

- Blank/NA entries reduce completeness and can bias any statistics or models drawn from the data. They can arise from data-entry omissions, equipment errors, or decisions not to record certain information.
- **Imputation goal** – restore completeness without distorting the distribution. For nominal (unordered) categories, one common strategy is mode imputation, filling each NA with the most frequent.
- Ethnicity is nominal; means or medians are undefined, and randomly guessing introduces noise. Using the prevailing category minimizes information loss and avoids creating artificial minority groups.

Problem 1.6 Code

```r
# Convert any empty strings ("") to real NA values so they are treated as missing.
# dplyr::na_if(): replaces "" with NA for cleaner missing-value handling.
d$education_level <- dplyr::na_if(d$education_level, "")

# Summarize missingness and current category frequencies for education_level.
before_sum <- sum(is.na(d$education_level))                      # count missing values
before_table <- table(d$education_level, useNA = "ifany")         # frequency table incl. NA

# ---- Impute missing education_level with the mode (most common category) ----
# The modeest package supplies several functions to estimate the statistical mode
# modeest::mfv(): returns the most frequent value(s) (mode) of a vector.
# na_rm = TRUE discards NA values before calculation.
library(modeest)
edu_mode <- modeest::mfv(d$education_level, na_rm = TRUE)  # most frequent level

# The tidyr package reshapes messy data frames into “tidy” form.
# tidyr::replace_na(): replaces NA values with a specified value.
d$education_level <- tidyr::replace_na(d$education_level, edu_mode)

# Verify that imputation succeeded and inspect updated frequencies.
sum(is.na(d$education_level))                          # should now be 0
table(d$education_level)                               # frequency table after fill
```

Problem 1.6 Result

[1] 116

Associate	Bachelor	High School	Master	PhD	<NA>
132	138	120	133	161	116

[1] 0

Associate	Bachelor	High School	Master	PhD
132	138	120	133	277

Problem 1.7 Recap

(7) Create a **bar graph** of your imputed **ethnicity** feature.

Problem 1.7 Recap

(7) Create a **bar graph** of your imputed **ethnicity** feature.

- **Bar graph** – displays the frequency (or percentage) of each categorical level as separate rectangular bars; bar height is proportional to count, with gaps between bars emphasizing that categories are discrete.

Problem 1.7 Recap

(7) Create a **bar graph** of your imputed **ethnicity** feature.

- **Bar graph** – displays the frequency (or percentage) of each categorical level as separate rectangular bars; bar height is proportional to count, with gaps between bars emphasizing that categories are discrete.
- **Nominal data fit** – ethnicity has no inherent order, so bars can be arranged by frequency or alphabetically; means or medians are meaningless, but bar lengths convey group size at a glance.

Problem 1.7 Recap

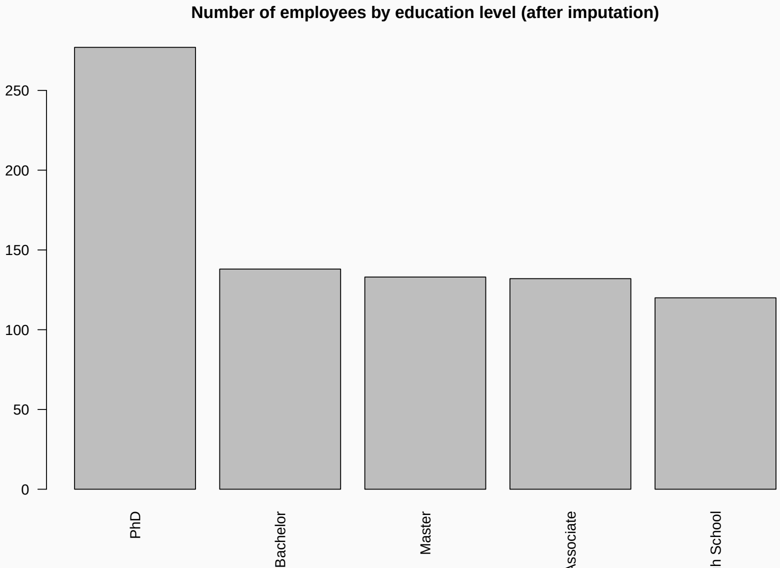
(7) Create a **bar graph** of your imputed **ethnicity** feature.

- **Bar graph** – displays the frequency (or percentage) of each categorical level as separate rectangular bars; bar height is proportional to count, with gaps between bars emphasizing that categories are discrete.
- **Nominal data fit** – ethnicity has no inherent order, so bars can be arranged by frequency or alphabetically; means or medians are meaningless, but bar lengths convey group size at a glance.
- **Good-graph principles** – start the y-axis at zero, title the chart and label axes, keep colors readable, and avoid 3-D or excessive decoration to maximize the "data-to-ink" ratio.

Problem 1.7 Code

```r
# Create a bar graph of the imputed education_level feature.
# table(): tabulates the frequency of each education level for plotting.
# sort(): used to sort by frequency, largest to smallest
# barplot(): draws a bar graph; las = 2 rotates x-axis labels for readability; main: title.
barplot(sort(table(d$education_level), decreasing = TRUE),
        las   = 2,
        main  = "Number of employees by education level (after imputation)")
```

Problem 1.7 Result



(8) Implement **dummy coding** for the **gender** feature. Replace the original gender feature with the coded result and report the coded gender for the last ten observations.

Problem 1.8 Recap

(8) Implement **dummy coding** for the **gender** feature. Replace the original gender feature with the coded result and report the coded gender for the last ten observations.

- **Dummy coding** – converts a categorical variable with k distinct levels into $k - 1$ binary (0/1) indicator columns; one level is chosen as the **reference** category and is represented by a column of zeros.

Problem 1.8 Recap

(8) Implement **dummy coding** for the **gender** feature. Replace the original gender feature with the coded result and report the coded gender for the last ten observations.

- **Dummy coding** – converts a categorical variable with k distinct levels into $k - 1$ binary (0/1) indicator columns; one level is chosen as the **reference** category and is represented by a column of zeros.
- **Why needed** – most statistical and machine-learning algorithms require numeric inputs; treating gender as text or arbitrary integers would either be rejected or falsely imply an order.

Problem 1.8 Code and Result

```r
# Ensure remote_work is treated as a categorical factor before coding.
d$remote_worker <- as.factor(d$remote_worker)

# fastDummies is a package that rapidly converts categorical variables into one-hot
# (dummy) columns in data frames.
# fastDummies::dummy_cols(): generates (k - 1) dummy variables for the selected column.
# select_columns = "remote_work": choose the feature to transform.
# remove_selected_columns = TRUE: drop the original remote_work column after coding.
# remove_first_dummy = TRUE: omit the first level’s dummy to prevent perfect multicollinearity.
library(fastDummies)
d <- fastDummies::dummy_cols(
        d,
        select_columns       = "remote_worker",
        remove_selected_columns = TRUE,
        remove_first_dummy    = TRUE
      )

# Extract the dummy-coded columns for the last ten observations to illustrate the result.
# grep("^remote_work_", names(d)) finds every new dummy column created above.
coded_remote_work_last10 <- tail(d[ , grep("^remote_worker_", names(d)) ], 10)
coded_remote_work_last10
```

```
[1] 1 1 0 1 1 0 1 0 0 1
```

Problem 1.9 Recap

(9) **Identify which features in your data set are discrete and which are continuous.**

Problem 1.9 Recap

(9) **Identify which features in your data set are discrete and which are continuous.**

- **Feature types** – every column is either **continuous numeric** (can take any real value within an interval, e.g., salary, temperature) or **discrete** (takes only distinct, countable values). Discrete features split further into *numeric-discrete* (counts such as number of children) and *categorical* (nominal, binary, or ordinal labels).

Problem 1.9 Recap

(9) **Identify which features in your data set are discrete and which are continuous.**

- **Feature types** – every column is either **continuous numeric** (can take any real value within an interval, e.g., salary, temperature) or **discrete** (takes only distinct, countable values). Discrete features split further into *numeric-discrete* (counts such as number of children) and *categorical* (nominal, binary, or ordinal labels).
- **Why the distinction matters** – many statistical tests and machine-learning models assume continuous inputs, while others require categorical indicators; misclassifying can lead to incorrect summaries (e.g., computing a mean of ZIP codes) or model errors.

Problem 1.9 Code

```r
# Clean blank strings ("") so that uniqueness counts are not distorted.
# dplyr::mutate(across()): applies a transformation across columns that meet a condition.
library(dplyr)
d_mutate <- mutate(d, across(where(is.character), ~ dplyr::na_if(.x, "")))

# Helper function to classify a single vector.
# Rule derived from lecture: non-numeric → discrete;
# numeric with ≤ 10 unique values = discrete; else continuous.
discrete_or_continuous <- function(vec) {
  if (!is.numeric(vec)) {
    return("discrete")
  }
  uniq <- length(unique(vec[!is.na(vec)]))   # unique() ignores duplicates; !is.na() excludes NAs.
  if (uniq <= 10) "discrete" else "continuous"
}

# sapply(): applies the helper to every column; returns a named character vector of classifications.
feature_type <- sapply(d_mutate, discrete_or_continuous)

# Separate names by class for easy reading.
discrete_feats   <- names(feature_type[feature_type == "discrete"])
continuous_feats <- names(feature_type[feature_type == "continuous"])

# Present the results as a list so they print cleanly in the console.
list(discrete   = discrete_feats,
     continuous = continuous_feats)
```

Problem 1.9 Result

```
$discrete
[1] "department_code"            "job_level"
[3] "work_life_balance_rating"   "education_level"
[5] "professional_certification" "remote_worker_Yes"

$continuous
[1] "age"                     "years_experience"
[3] "commute_distance_miles"  "monthly_wellbeing_score"
```

Problem 1.10 Recap

(10) **Identify which features in your data set are numeric and which are non-numeric.** Compare this classification with the discrete/continuous split you made earlier and discuss the similarities and differences you observe.

Problem 1.10 Recap

(10) **Identify which features in your data set are numeric and which are non-numeric.** Compare this classification with the discrete/continuous split you made earlier and discuss the similarities and differences you observe.

- **Numeric features** represent measurable quantities. They may be **discrete numeric** (counts) or **continuous numeric**.

Problem 1.10 Recap

(10) **Identify which features in your data set are numeric and which are non-numeric.** Compare this classification with the discrete/continuous split you made earlier and discuss the similarities and differences you observe.

- **Numeric features** represent measurable quantities. They may be **discrete numeric** (counts) or **continuous numeric**.
- **Non-numeric features** describe categories rather than quantities; stored as character strings or coded factors (e.g., department, job title, ethnicity). Their levels are labels, not magnitudes.

Problem 1.10 Recap

(10) **Identify which features in your data set are numeric and which are non-numeric.** Compare this classification with the discrete/continuous split you made earlier and discuss the similarities and differences you observe.

- **Numeric features** represent measurable quantities. They may be **discrete numeric** (counts) or **continuous numeric**.
- **Non-numeric features** describe categories rather than quantities; stored as character strings or coded factors (e.g., department, job title, ethnicity). Their levels are labels, not magnitudes.
- Algorithms and summary statistics assume specific input types.

Problem 1.10 Recap

(10) **Identify which features in your data set are numeric and which are non-numeric.** Compare this classification with the discrete/continuous split you made earlier and discuss the similarities and differences you observe.

- **Numeric features** represent measurable quantities. They may be **discrete numeric** (counts) or **continuous numeric**.
- **Non-numeric features** describe categories rather than quantities; stored as character strings or coded factors (e.g., department, job title, ethnicity). Their levels are labels, not magnitudes.
- Algorithms and summary statistics assume specific input types.
- All continuous features are numeric, but some numeric features are discrete. Conversely, categorical variables are always non-numeric even though they are also "discrete."

Problem 1.10 Code

```r
# Convert blank strings ("") in character columns to real NA so counts & classes are accurate.
# dplyr::mutate(across()): applies a function across every character column that meets the condition.
library(dplyr)
d_mutate <- mutate(d, across(where(is.character), ~ dplyr::na_if(.x, "")))

# Determine the base R class of each column.
# sapply(): iterates over the data frame’s columns; class() returns each column’s type label.
col_classes <- sapply(d_mutate, class)

# A column is numeric if its class is "numeric" or "integer"; otherwise treat it as non-numeric.
numeric_flags <- col_classes %in% c("numeric", "integer")

# Separate the feature names by class.
numeric_feats     <- names(col_classes[numeric_flags])
nonnumeric_feats  <- names(col_classes[!numeric_flags])

# Present the results as a list so they print cleanly in the console.
list(numeric     = numeric_feats,
     non_numeric = nonnumeric_feats)
```

Problem 1.10 Result

```
$numeric
[1] "age"                      "department_code"
[3] "years_experience"         "commute_distance_miles"
[5] "monthly_wellbeing_score"  "work_life_balance_rating"
[7] "remote_worker_Yes"

$non_numeric
[1] "job_level"                  "education_level"
[3] "professional_certification"
```

Problem 1.11 Recap

(11) After completing all requested tasks above, **print the first four observations** of the data.

Problem 1.11 Recap

(11) After completing all requested tasks above, **print the first four observations** of the data.

- Showing the top rows lets you visually confirm that earlier steps took effect as intended.

Problem 1.11 Recap

(11) After completing all requested tasks above, **print the first four observations** of the data.

- Showing the top rows lets you visually confirm that earlier steps took effect as intended.
- It is a **quality check** that ensures that no unintended effects or data loss occurred during processing.

Problem 1.11 Recap

(11) After completing all requested tasks above, **print the first four observations** of the data.

- Showing the top rows lets you visually confirm that earlier steps took effect as intended.
- It is a **quality check** that ensures that no unintended effects or data loss occurred during processing.
- It also acts as a **snapshot** to make it easy for others (graders) to visually confirm the same.

Problem 1.11 Recap

(11) After completing all requested tasks above, **print the first four observations** of the data.

- Showing the top rows lets you visually confirm that earlier steps took effect as intended.
- It is a **quality check** that ensures that no unintended effects or data loss occurred during processing.
- It also acts as a **snapshot** to make it easy for others (graders) to visually confirm the same.
- It can also verify the dataset ready for additional visualization or modeling.

Problem 1.11 Code

```r
# head(): shows the first rows of a data frame; n = 4 restricts output to four observations.
head(d, n = 4)
```

Problem 1.11 Result

```
  age department_code job_level years_experience commute_distance_miles
1  56             301       Mid        0.3043478                   2.90
2  46             301       Mid        0.3913043                   7.57
3  32             102       Mid        0.3913043                   1.79
4  60             201      Lead        0.7391304                   9.09
  monthly_wellbeing_score work_life_balance_rating education_level
1                    5.72                        3             PhD
2                    4.54                        1             PhD
3                    8.87                        4        Bachelor
4                    7.02                        1       Associate
  professional_certification remote_worker_Yes
1                       <NA>                 1
2                        PMP                 0
3                        PMP                 1
```

Problem 2.1 Recap

Problem: Create a scatterplot of feature **A1** vs. **A5**

- **Scatterplot** – plots each observation as an (x = A1, y = A5) point; primary tool for bivariate exploration.

Problem 2.1 Recap

Problem: Create a scatterplot of feature **A1** vs. **A5**

- **Scatterplot** – plots each observation as an ($x = A1$, $y = A5$) point; primary tool for bivariate exploration.
- **Direction** – look for a positive slope, negative slope, or no visible trend.

Problem 2.1 Recap

Problem: Create a scatterplot of feature **A1** vs. **A5**

- **Scatterplot** – plots each observation as an ($x = A1$, $y = A5$) point; primary tool for bivariate exploration.
- **Direction** – look for a positive slope, negative slope, or no visible trend.
- **Form** – inspect whether the relationship appears linear, curved, clustered, or segmented.

Problem 2.1 Recap

Problem: Create a scatterplot of feature **A1** vs. **A5**

- **Scatterplot** – plots each observation as an ($x = \text{A1}$, $y = \text{A5}$) point; primary tool for bivariate exploration.
- **Direction** – look for a positive slope, negative slope, or no visible trend.
- **Form** – inspect whether the relationship appears linear, curved, clustered, or segmented.
- **Outliers** – single points that break the overall pattern; may signal data entry errors or rare cases.

Problem 2.1 Code

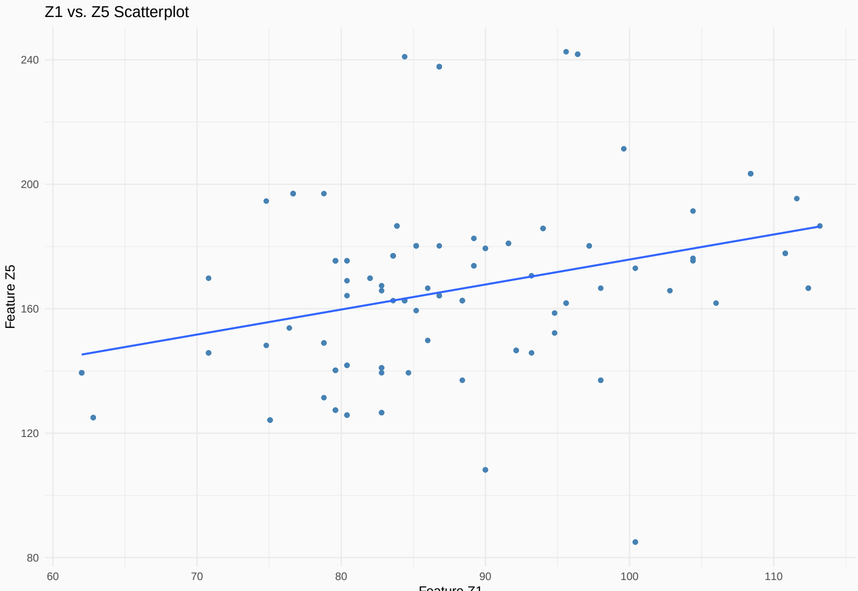
```r
d <- read.csv("alternate_correlation.csv")

# Remove rows that contain missing values in either feature before plotting.
# tidyr::drop_na(): drops rows with NA in the selected columns.
d_clean <- tidyr::drop_na(d, Z1, Z5)

# Ensure both features are numeric so the scatterplot has meaningful axes.
# suppressWarnings(): hides coercion warnings; as.numeric(): converts to numeric.
d_clean$Z1 <- suppressWarnings(as.numeric(d_clean$Z1))
d_clean$Z5 <- suppressWarnings(as.numeric(d_clean$Z5))

# Create a scatterplot with a least-squares trend line.
# ggplot2 is an R package that lets you build layered, customizable graphics by
# mapping data columns to geometric objects.
# ggplot(): initializes the plot; aes(): maps x and y; geom_point(): draws points;
# geom_smooth(): adds a linear model fit; se = FALSE removes the ribbon.
library(ggplot2)
ggplot(d_clean, aes(x = Z1, y = Z5)) +
  geom_point(color = "steelblue") +
  geom_smooth(method = "lm", se = FALSE, linewidth = 0.8) +
  labs(title = "Z1 vs. Z5 Scatterplot",
       x = "Feature Z1",
       y = "Feature Z5") +
  theme_minimal()
```

Problem 2.1 Result



Problem 2.2 Recap

Compute the correlation matrix for all five features in the data set

- **Correlation matrix** – a 5×5 symmetric table whose (i,j) entry is the correlation coefficient between feature *i* and feature *j*.

Problem 2.2 Recap

Compute the correlation matrix for all five features in the data set

- **Correlation matrix** – a 5×5 symmetric table whose (i,j) entry is the correlation coefficient between feature *i* and feature *j*.
- **Pearson's *r*** – default measure in the lecture; standardizes each feature, then takes the mean product of Z-scores; values range from –1 (perfect negative) to $+1$ (perfect positive).

Problem 2.2 Recap

Compute the correlation matrix for all five features in the data set

- **Correlation matrix** – a 5×5 symmetric table whose (i,j) entry is the correlation coefficient between feature *i* and feature *j*.
- **Pearson's *r*** – default measure in the lecture; standardizes each feature, then takes the mean product of Z-scores; values range from –1 (perfect negative) to $+1$ (perfect positive).
- **Diagonal of 1.0** – each feature is perfectly correlated with itself, so the main diagonal is 1's by definition.

Problem 2.2 Code

```r
d <- read.csv("alternate_correlation.csv")

# Create a vector of column names to clean.
cols <- paste0("Z", 1:5)

# Ensure every feature is numeric for a valid Pearson correlation.
# suppressWarnings(): hides coercion warnings; as.numeric(): converts to numeric.
for (v in cols) d[[v]] <- suppressWarnings(as.numeric(d[[v]]))

# Compute the Pearson correlation matrix, handling missing values pairwise.
# The Pearson correlation is the standard yard-stick for how tightly two numeric
# columns move together.
# cor(): computes correlations; use = "pairwise.complete.obs" keeps all available
# observations for each pair; method = "pearson" is the default linear correlation.
cor_mat <- cor(d[cols], use = "pairwise.complete.obs", method = "pearson")

# Print the correlation matrix so we  can see the result.
cor_mat
```

Problem 2.2 Result

```
           Z1         Z2         Z3         Z4        Z5
Z1 1.00000000 0.47470877 0.19928925 0.09388067 0.2952696
Z2 0.47470877 1.00000000 0.04539518 0.09425457 0.3563165
Z3 0.19928925 0.04539518 1.00000000 0.50556877 0.1300788
Z4 0.09388067 0.09425457 0.50556877 1.00000000 0.1577464
Z5 0.29526959 0.35631655 0.13007877 0.15774641 1.0000000
```

Problem 2.3 Recap

Identify the strongest correlation in the data set. Which factors are involved? Is it a positive correlation or a negative correlation?

- **Strongest pair** – locate the off-diagonal cell in the correlation matrix with the largest absolute value $|r|$; that pair of features exhibits the most pronounced linear relationship.

Problem 2.3 Recap

Identify the strongest correlation in the data set. Which factors are involved? Is it a positive correlation or a negative correlation?

- **Strongest pair** – locate the off-diagonal cell in the correlation matrix with the largest absolute value $|r|$; that pair of features exhibits the most pronounced linear relationship.
- **Sign vs. magnitude** – the *magnitude* ($|r|$) signals strength, while the *sign* ($+$ or $-$) reveals direction.

Problem 2.3 Recap

Identify the strongest correlation in the data set. Which factors are involved? Is it a positive correlation or a negative correlation?

- **Strongest pair** – locate the off-diagonal cell in the correlation matrix with the largest absolute value $|r|$; that pair of features exhibits the most pronounced linear relationship.
- **Sign vs. magnitude** – the *magnitude* ($|r|$) signals strength, while the *sign* ($+$ or $-$) reveals direction.
- **Decision rule** – the lecture treats $|r|$ 0.7 as "strong." Among cells meeting that threshold, pick the one with the highest $|r|$; if none reach 0.7, choose the highest available value and note it is only moderate or weak.

Problem 2.3 Code Part 1

```r
d <- read.csv("alternate_correlation.csv")

# Create a vector of column names to clean.
cols <- paste0("Z", 1:5)

# Ensure every feature is numeric for a valid Pearson correlation.
# suppressWarnings(): hides coercion warnings; as.numeric(): converts to numeric.
for (v in cols) d[[v]] <- suppressWarnings(as.numeric(d[[v]]))

# Compute the Pearson correlation matrix, handling missing values pairwise.
# The Pearson correlation is the standard yard-stick for how tightly two numeric
# columns move together.
# cor(): computes correlations; use = "pairwise.complete.obs" keeps all available
# observations for each pair; method = "pearson" is the default linear correlation.
cor_mat <- cor(d[cols], use = "pairwise.complete.obs", method = "pearson")
```

Problem 2.3 Code Part 2

```r
lapply(c("tidyr", "tibble"), library, character.only = TRUE)
cor_df <- cor_mat %>%                 # start with the matrix
  as.data.frame() %>%                 # 1. make it a data frame so tidy verbs work
  rownames_to_column("Feature1") %>% # 2. preserve row names as a real column
  pivot_longer(                       # 3. pivot from wide to long:
      -Feature1,                      #    everything *except* Feature1 …
      names_to  = "Feature2",         #    … becomes Feature2
      values_to = "r") %>%            #    correlations go into column r
  filter(Feature1 < Feature2) %>%     # 4. keep only upper-triangle rows:
                                      #    same pair once, drop the 1.0 diagonal
  mutate(abs_r = abs(r)) %>%          # 5. add |r| so we rank by strength
  arrange(desc(abs_r))                # 6. strongest correlation first

# Extract the top row: the feature pair with the largest |r|.
strongest <- cor_df[1, ]

# Report the pair, the correlation coefficient, and its sign.
list(
  feature_pair   = paste(strongest$Feature1, strongest$Feature2, sep = " - "),
  correlation_r  = strongest$r,
  correlation_is = ifelse(strongest$r > 0, "positive", "negative")
)
```

Problem 2.3 Result

```
$feature_pair
[1] "Z3 - Z4"

$correlation_r
[1] 0.5055688

$correlation_is
[1] "positive"
```

Problem 2.4 Recap

Implement z-score normalization on all features in the data set

- You slide every column so its average is 0 and stretch or shrink it so one “step” equals one standard deviation.

Problem 2.4 Recap

Implement z-score normalization on all features in the data set

- You slide every column so its average is 0 and stretch or shrink it so one “step” equals one standard deviation.
- Find the mean, find the spread (SD), subtract the mean from each value, then divide by SD.

Problem 2.4 Recap

Implement z-score normalization on all features in the data set

- You slide every column so its average is 0 and stretch or shrink it so one "step" equals one standard deviation.
- Find the mean, find the spread (SD), subtract the mean from each value, then divide by SD.
- This formula is a shorthand: $z = (x - \text{mean}) / \text{SD}$

Problem 2.4 Recap

Implement z-score normalization on all features in the data set

- You slide every column so its average is 0 and stretch or shrink it so one “step” equals one standard deviation.
- Find the mean, find the spread (SD), subtract the mean from each value, then divide by SD.
- This formula is a shorthand: $z = (x - \text{mean}) / \text{SD}$
- It puts every feature on the same yard stick.

Problem 2.4 Recap

Implement z-score normalization on all features in the data set

- You slide every column so its average is 0 and stretch or shrink it so one "step" equals one standard deviation.
- Find the mean, find the spread (SD), subtract the mean from each value, then divide by SD.
- This formula is a shorthand: $z = (x - \text{mean}) / \text{SD}$
- It puts every feature on the same yard stick.
- Makes units disappear (centimeters, dollars, etc.), so columns measured in different units no longer dominate the model.

Problem 2.4 Code and Result

```r
d <- read.csv("alternate_correlation.csv")

# Apply z-score normalization column-wise.
# scale(): centers (subtracts mean) and scales (divides by SD) when both
# center = TRUE and scale = TRUE (the defaults). Results are returned as a
# matrix; wrap with as.data.frame() for consistency with d.
d[cols] <- as.data.frame(scale(d[cols], center = TRUE, scale = TRUE))

# Optional: inspect the first few rows to confirm transformation.
head(d[cols])
```

```
          Z1          Z2         Z3         Z4          Z5
1  1.1182920 -0.03051514  0.1562237  0.5762430  1.49553849
2 -0.2251155 -1.09854486 -0.5819467 -1.4749695 -0.08827831
3  1.8960542  2.01835822 -0.9174787 -0.1621935  1.23589639
4  2.1081712  2.60686440 -0.2464147 -1.4585598  0.40504168
5  2.1081712  2.60686440 -0.2464147 -1.4585598  0.40504168
6 -0.9088392 -0.59722479  0.9615005  0.8511055  1.02818271
```

Problem 2.5 Recap

Compute the correlation matrix for all five normalized features in the data set. Compare this correlation matrix with the matrix you obtained earlier and discuss the similarities and/or differences you see.

- Apply the same Pearson procedure to the z-score–transformed data, yielding a new 5×5 table.

Problem 2.5 Recap

Compute the correlation matrix for all five normalized features in the data set. Compare this correlation matrix with the matrix you obtained earlier and discuss the similarities and/or differences you see.

- Apply the same Pearson procedure to the z-score–transformed data, yielding a new 5×5 table.
- Compare the results.

Problem 2.5 Code

```r
d <- read.csv("alternate_correlation.csv")

# Create a vector of column names to clean.
cols <- paste0("Z", 1:5)

# Compute the Pearson correlation matrix, handling missing values pairwise.
# The Pearson correlation is the standard yard-stick for how tightly two numeric
# columns move together.
# cor(): computes correlations; use = "pairwise.complete.obs" keeps all available
# observations for each pair; method = "pearson" is the default linear correlation.
cor_mat_norm <- cor(d[cols], use = "pairwise.complete.obs", method = "pearson")

library(cli)
cli::cat_rule("Correlation matrix (raw features)")
print(cor_mat)

cli::cat_rule("Correlation matrix (normalized features)")
print(cor_mat_norm)
```

Problem 2.5 Result

```
-- Correlation matrix (raw features) ------------------------------------------

           Z1         Z2         Z3         Z4        Z5

Z1 1.00000000 0.47470877 0.19928925 0.09388067 0.2952696

Z2 0.47470877 1.00000000 0.04539518 0.09425457 0.3563165

Z3 0.19928925 0.04539518 1.00000000 0.50556877 0.1300788

Z4 0.09388067 0.09425457 0.50556877 1.00000000 0.1577464

Z5 0.29526959 0.35631655 0.13007877 0.15774641 1.0000000

-- Correlation matrix (normalized features) -----------------------------------

           Z1         Z2         Z3         Z4        Z5

Z1 1.00000000 0.47470877 0.19928925 0.09388067 0.2952696

Z2 0.47470877 1.00000000 0.04539518 0.09425457 0.3563165

Z3 0.19928925 0.04539518 1.00000000 0.50556877 0.1300788

Z4 0.09388067 0.09425457 0.50556877 1.00000000 0.1577464
```