Introduction to Machine Learning

Dr Niamh Mimnagh

niamh@prstats.org

https://github.com/niamhmimnagh/IMLR04-Introduction-to-Machine-Learning

Clustering

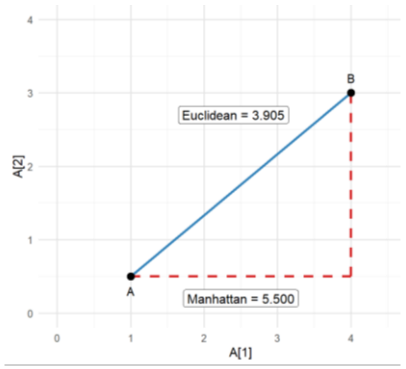
- Clustering is an unsupervised learning technique that aims to group a set of observations into subsets so that observations within each cluster are more similar to one another than to observations in other clusters.
- “Similarity” is typically measured using a distance metric (e.g., Euclidean distance, Manhattan distance, cosine similarity).
- Clusters capture latent structure in the data without requiring pre-defined labels.
- Goals of Clustering:
  - Discover hidden patterns or groupings in data.
  - Reduce data complexity by summarising into representative clusters.
  - Serve as a preprocessing step for other analyses (e.g., anomaly detection, classification).

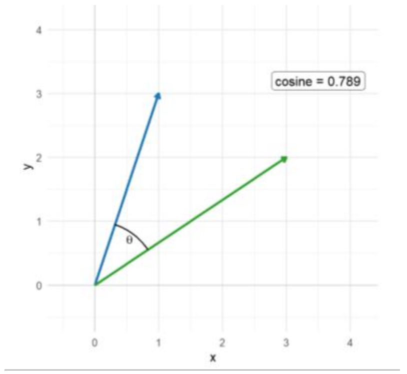
Applications of Clustering

- Customer Segmentation (Marketing):
  - Companies use clustering to group customers with similar purchasing behaviour, demographics, or preferences.
  - Helps in targeted advertising, personalized recommendations, and product development.
- Gene Expression Profiling (Biology/Medicine):
  - Clustering is widely used in bioinformatics to group genes with similar expression levels across conditions or time.
  - Useful for discovering gene functions, identifying disease subtypes, or revealing biomarkers.
- Species Assemblages (Ecology):
  - Ecologists group species or habitats based on traits, abundance, or environmental factors.
  - Supports biodiversity assessment, conservation planning, and identifying ecological niches.

Distance Metrics

- Clustering relies on a measure of distance or similarity to decide which observations should be grouped together.
- Euclidean Distance (Most Common):
  - The straight-line (L2 norm) distance between two points in space.
  - Works well for continuous, numerical data.
  - Example: Distance between species traits (e.g., body length, weight).
- Manhattan Distance (City Block Distance):
  - Sum of absolute differences (L1 norm).
  - More robust to outliers than Euclidean distance.
  - Example: Comparing gene counts across conditions.
- Cosine Similarity:
  - Measures the cosine of the angle between two vectors.
  - Emphasises direction over magnitude, making it ideal for high-dimensional sparse data (e.g., text).





K-Means Clustering

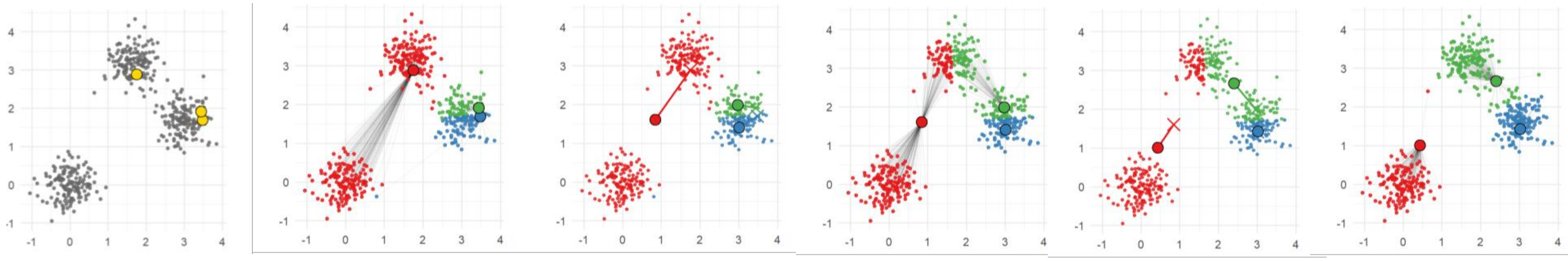
- K-means is one of the most widely used clustering algorithms.
- It partitions the dataset into k clusters, where each observation belongs to the cluster with the nearest centroid.
- Centroid: The mean point (centre) of a cluster.
- Objective Function: Minimise the Within-Cluster Sum of Square:

$$WCSS = \sum_{i=1}^{k} \sum_{x \in C_i} ||x - \mu_i||^2$$

- Where C_i is the set of points in cluster i and μ_i is the centroid.

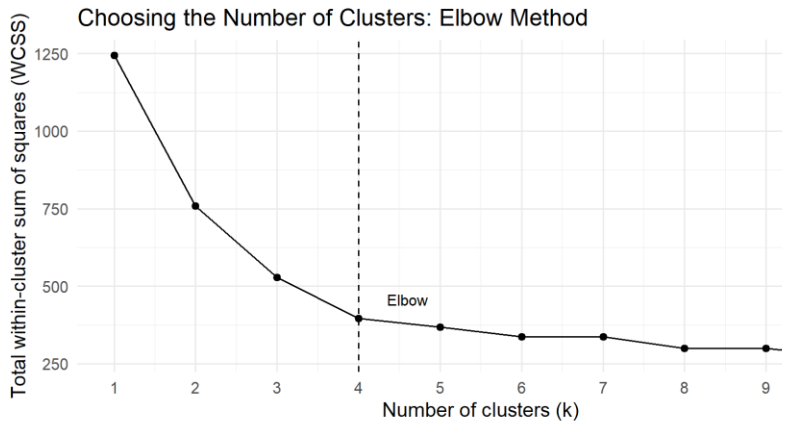
K-Means Algorithm

1. Choose the number of clusters k.
2. Randomly initialize k centroids.
3. Assign each data point to the nearest centroid (cluster assignment).
4. Recompute centroids as the mean of assigned points.
5. Repeat steps $3–4$ until centroids stabilise or maximum iterations reached.



Choosing k: The Elbow Method

- K-means requires specifying the number of clusters k in advance.
- Too few clusters oversimplifies the data.
- Too many clusters overfits and loses interpretability.
- Elbow Method:
  - Calculate Total Within-Cluster Sum of Squares (WCSS) for different values of k.
  - WCSS decreases as k increases (clusters get tighter).
  - At some point, the rate of improvement drops - this point is the “elbow”. The elbow indicates a balance between simplicity (fewer clusters) and accuracy (lower WCSS). The chosen k should be at or near the elbow.

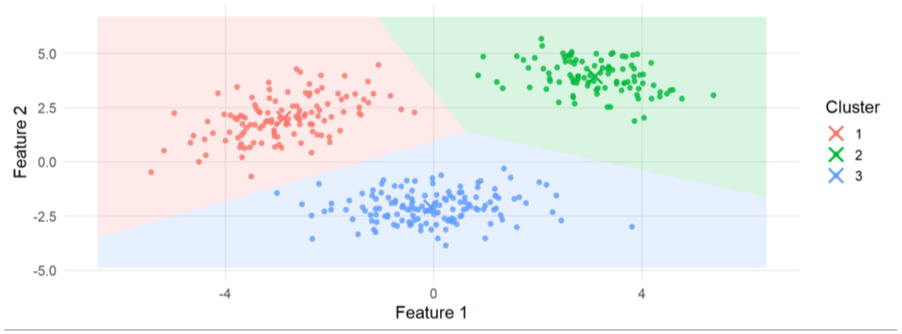


K-Means Clustering in R

```
km_spec <- k_means(num_clusters = 3) %>%     # set k here
  set_engine("stats")

km_wf <- workflow() %>%
  add_recipe(km_rec) %>%
  add_model(km_spec)

km_fit <- fit(km_wf, data = dat)

clustered <- augment(km_fit, dat)    # adds .cluster
columnhead(clustered)
```

Visualising K-Means Clusters

- Understanding structure
  - Cluster plots help reveal the natural groupings in the data
  - We can see whether clusters are compact, overlapping, or poorly separated.
- Assessing model fit
  - Visualisation shows if the chosen k makes sense
  - If clusters strongly overlap, perhaps fewer/more clusters are needed, or k-means isn't appropriate
- Interpreting centroids
  - Plots let us compare cluster centres and understand what the 'average' member of each group looks like



Strengths of K-Means

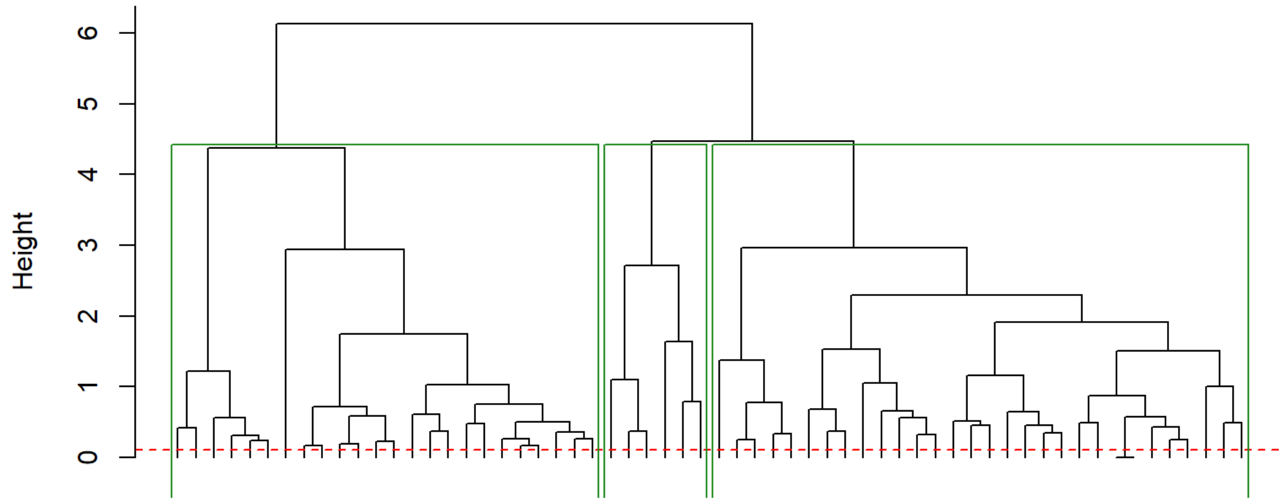
- Fast and computationally efficient
  - Can handle tens of thousands of observations with ease.
- Easy to understand and implement
  - Intuitive: assigns points to nearest centroid.
- Works well in the right conditions
  - Performs best when clusters are spherical, compact, and similar in size.
  - Good baseline method for many clustering tasks.
- Widely used and well-studied
  - Common in marketing, biology, ecology, text mining, and computer vision.
  - Results are easily communicated to non-technical audiences (centroids = "average group member").

Limitations of K-Means

- Sensitive to initialisation
  - Results depend on the starting positions of centroids and chosen k.
- Assumes equal cluster size and spherical shape
  - K-means partitions space using Voronoi regions (circular/spherical boundaries).
  - Struggles with elongated, overlapping, or irregular-shaped clusters.
  - Not ideal for clusters of different densities.
- Requires numeric variables
  - Based on distance metrics (Euclidean/Manhattan), so data must be continuous and numeric.
- Can converge to local minima
  - The algorithm always reduces WCSS, but may get "stuck" in a poor solution. No guarantee of finding the global optimum.

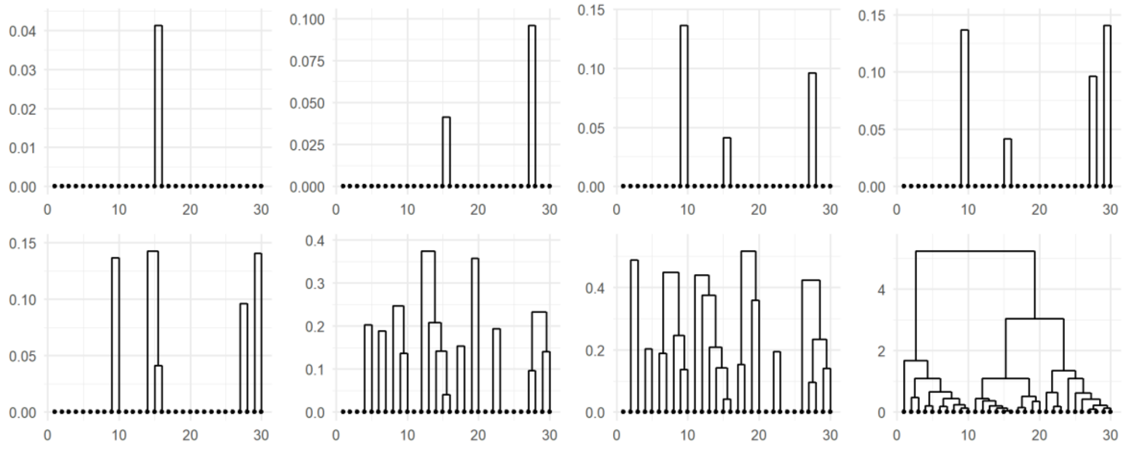
Hierarchical Clustering

- Hierarchical clustering builds a hierarchy of nested clusters without fixing k in advance. The result is a dendrogram (tree) that shows how and when observations or clusters merge and split across dissimilarity levels.



Agglomerative Hierarchical Clustering

1. Start with n singleton clusters
2. Compute a dissimilarity matrix between all items (an $n \times n$ matrix D where D_{ij} quantifies how different observation i is from observation j).
3. Find the closest pair of clusters and merge them.
4. Update inter-cluster distances, and repeat until all points are in one cluster.
5. Read off groups by cutting the dendrogram



Linkage

- Single linkage (distance between closest members):
  - Can find non-spherical shapes, but can cause chaining (long, straggly clusters)
- Complete linkage (distance between farthest members):
  - Yields compact, evenly sized clusters, and less chaining
- Average linkage (mean pairwise distance between clusters):
  - Balanced compromise between single and complete linkage
- Ward’s method (merges that minimise the increase in within-cluster variance (WCSS):
  - Often best for compact, spherical clusters. Typically used with Euclidean distances.

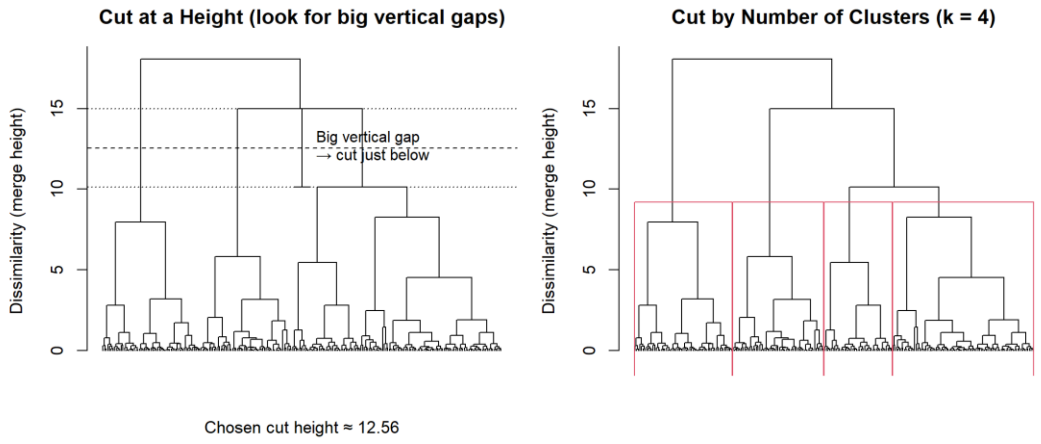
Hierarchical Clustering in R

- To perform hierarchical clustering in R, we need to standardise features, choose a linkage that matches our goal, and a distance metric.

```
hc_spec <- hier_clust(linkage_method = "ward.D2") %>%
set_engine("stats")fit <- hc_spec %>% fit(~ ., data = X)
tree <- fit %>% extract_fit_engine()
cl_k3 <- cutree(tree, k = 3)                    # choose k later
```

Cutting the Tree

- The options are to cut at a certain height, or to ask for a certain number of clusters.
- When visualising the dendrogram, look for big vertical gaps as natural cutpoints, and cut just below that gap. A large gap means you are about to merge two clusters that are quite dissimilar, so we stop right before it.



K-Means vs. Hierarchical Clustering

- When to use
  - K-means: large n, numeric features, roughly spherical/compact clusters
  - Hierarchical: small-medium n, need an interpretable tree and multi-scale view; mixed distances (e.g., Gower/Jaccard) are desired.
- Input requirements
  - K-means: must choose k in advance
  - Hierarchical: no k up front; choose cut height or k after seeing the dendrogram.
- Output
  - K-means: a single partition + centroids (cluster prototypes).
  - Hierarchical: a dendrogram (nested clusters at all resolutions).

Silhouette Scores

- For observation i, the silhouette score compares how well it matches its own cluster vs. the nearest other cluster.
- To compute it we find the cohesion α_i (the average distance from point i to points in its own cluster), and the separation β_i (the minimum of the average distance from i to points in that other cluster)

$$s_i = \frac{\beta_i - \alpha_i}{\max\{\alpha_i, \beta_i\}}$$

- If the silhouette score is approximately 1, then the observation is well-clustered, far from other clusters. If it is approximately 0 then the observation is on a boundary, and if it is less than 0 it is likely misaligned.
- Silhouette scores can be used to choose k: pick the k with the highest average silhouette.

Davies-Bouldin Index

- This measures the average worst-case similarity between each cluster and its most similar other cluster.
- For cluster i, measure the within-cluster scatter S_i(e.g., mean distance to centroid), and the distance between centroids of clusters i and j, M_{ij}.

$$R_{ij} = \frac{S_i + S_j}{M_{ij}}$$

- A big numerator (two loose clusters) and a small denominator (centroids close together) will make R_{ij} large, which means the two clusters are too similar, and poorly separated.
  - For cluster i take its worst neighbour, $R_i = \max(R_{ij})$
  - The DB index is the average of these worst case values. Lower is better – this indicates clusters are tight (small S_i) and far from its neighbours (large M_{ij})
- DB is for relative comparison, and can be used to pick among different k values, algorithms, distances or preprocessing choices.

Adjusted Rand Index (ARI)

- This measures the agreement between two partitions based on pair counting, corrected for chance.
- We have two partitions of n items: A (e.g., truth) and B (e.g. clustering) and we want to compare them. We do this using ‘together pairs’ – pairs of items that end up in the same cluster.

- Example:
  - Items: 1,2,3,4,5
  - A: {1,2}, {3,4,5}
  - B: {1,2,3}, {4,5}
  - Together in A: (1,2), (3,4), (3,5), (4,5). Separate in A: (1,3), (1,4), (1,5), (2,3), (2,4), (2,5)
  - Together in B: (1,2), (1,3), (2,3), (4,5). Separate in B: (1,4), (1,5), (2,4), (2,5), (3,4), (3,5)
  - Together in both: (1,2), (4,5). Separate in both: (1,4), (1,5), (2,4), (2,5)

Adjusted Rand Index (ARI)

- We calculate:
  - RI: The Rand Index = $\frac{(together\ in\ both)+(separate\ in\ both)}{\binom{n}{2}} = \frac{2+4}{10} = 0.6$
  - E[RI] = $\frac{(together\ in\ A)}{\binom{n}{2}} \cdot \frac{(together\ in\ B)}{\binom{n}{2}} + \left(1 - \frac{(together\ in\ A)}{\binom{n}{2}}\right) \cdot \left(1 - \frac{(together\ in\ B)}{\binom{n}{2}}\right) =$ $0.4 \cdot 0.4 + 0.6 \cdot 0.6 = 0.52$
  - max(RI) = $\frac{\binom{n}{2}-(together\ in\ A)-(together\ in\ B)+2\sum\binom{n_{ij}}{2}}{\binom{n}{2}} = \frac{10-4-4+8}{10} = 1$

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]} \approx -0.0417$$

Adjusted Rand Index (ARI)

- ARI ≥ 0.80 — Excellent:
  - Partitions nearly identical (up to label names).
- ARI 0.60–0.79 — Good:
  - Clear alignment; only boundary disagreements.
- ARI 0.30–0.59 — Moderate:
  - Some structure matches, but notable mismatches.
- ARI 0.00–0.29 — Near chance:
  - Little reliable agreement; treat with caution.
- ARI < 0.00 — Worse than chance:
  - Cross-cutting - clusters slice through true classes so each cluster mixes labels and each label is split across clusters.

Normalised Mutual Information(NMI)

NMI measures how much information the clustering shares with the labels (or with another clustering) scaled to [0,1] so it is comparable across different k and class sizes.

- Create a contingency table
- Calculate mutual information I(A;B) – how much knowing A reduces uncertainty in B, and vice versa
- Calculate entropies H(A) and H(B) – uncertainty within each partition

$$NMI = \frac{I(A;B)}{\sqrt{H(A)H(B)}}$$

Normalised Mutual Information(NMI)

- NMI ≈ 1.00 — Identical:
  - Partitions exactly match
- NMI 0.95–0.99 — Near-perfect:
  - Tiny discrepancies (usually a handful of boundary points).
- NMI 0.80–0.94 — Excellent:
  - Very strong alignment; minor boundary differences.
- NMI 0.60–0.79 — Good:
  - Clear agreement; most structure shared.
- NMI 0.30–0.59 — Moderate:
  - Partial overlap; notable mismatches remain.
- NMI 0.00–0.29 — Near independence:
  - Little shared structure; cross-cutting - clusters mix labels and labels are split across multiple clusters.

Cluster Validation

- Internal (no labels needed)
  - Silhouette width: cohesion vs separation; average close to 1 is best, near 0 is boundary, <0 is misassigned.
  - Davies–Bouldin (DB) index: average "worst-case" cluster similarity; lower is better (compact & well separated).
- External (labels available)
  - Compare clusters to ground truth using ARI or NMI
- Stability (robustness)
  - Refit on resamples/subsets and compare to a reference solution (e.g., ARI to the full-data fit).
  - Stable clusters reappear across resamples; instability suggests rethinking k, features, scaling, or the algorithm.

Example: Iris

- Setup
  - Drop the label Species: use only the 4 numeric features.
  - Standardise features (mean=0, sd=1) so distances are comparable.
  - K-means(try a grid of k; pick via Elbow and Silhouette).
  - Hierarchical (Euclidean + Ward.D2; decide cut by big vertical gaps or silhouette).

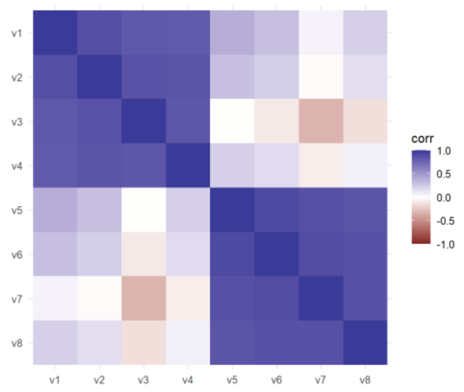
- Can we recover original groupings?

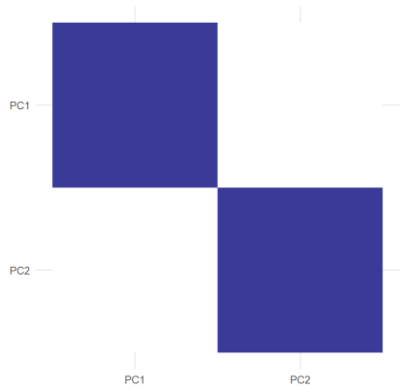
Dimensionality Reduction

- Dimensionality reduction transforms high-dimensional data into a smaller set of features while preserving the essential structure of the dataset. It helps us:
  - Visualise complex data in 2-3 dimensions so we can see patterns that are otherwise hidden.
  - Reduce noise and collinearity, which makes modelling more stable.
  - Speed up learning algorithms by shrinking the feature space.
- It acts as a form of regularisation by removing redundant variance and can help prevent overfitting.

Principal Components Analysis (PCA)

- PCA is a transformation that rotates the data to a new coordinate system.
- The aim is to reduce dimensionality while retaining structure, denoise correlated measurements, and create uncorrelated features that are easier to model and visualise.
- PCs are linear combinations of the original variables; PC1 captures the largest possible variance, PC2 the next largest under orthogonality, and so on.





Principal Components Analysis (PCA)

1. Standardise and centre features so different scales don't dominate
2. Compute the covariance matrix to summarise how variables vary together (sign shows direction of association).
3. Eigen-decompose the covariance matrix: eigenvectors give PC directions; eigenvalues give variance explained. Order PCs by descending eigenvalue.
4. Keep the leading PCs; dropping low-variance PCs reduces dimension with minimal information loss.
5. Project the data onto the retained PCs to get scores - the rotated representation used for modelling and visualisation

Principal Components Analysis (PCA)

- Eigenvectors define a PC. An eigenvector is the weight vector for a principal component. Its entries (loadings) tell you which variables drive that PC and in what direction.
- Eigenvalues tell us how much information the PC keeps. Eigenvalues tell us how much variance each PC explains.

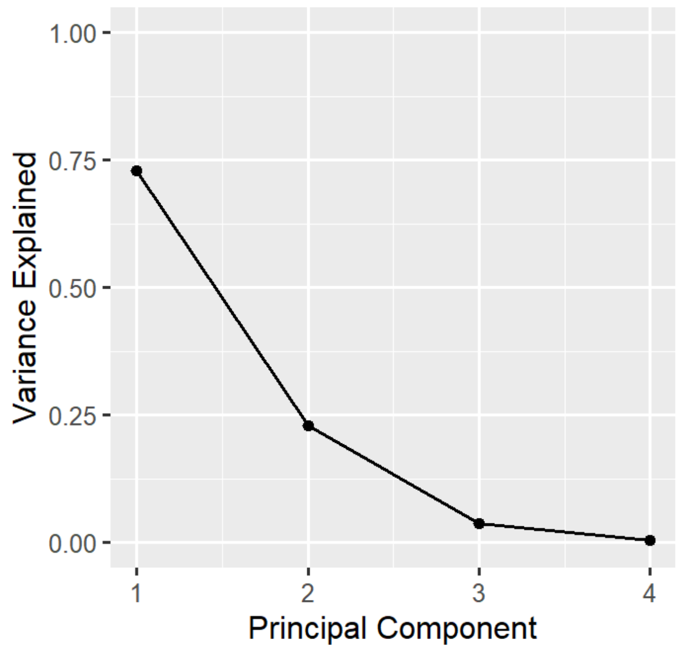
Variable	PC1	PC2
TV	0.57	0.09
Radio	0.55	- 0.3
Online	0.49	0.55
Print	0.33	- 0.82

$$\lambda_1 = 2.6 \Rightarrow PVE_1 = \frac{2.6}{4} = 65\%$$

$$\lambda_2 = 1.1 \Rightarrow PVE_2 = \frac{1.1}{4} = 27.5\%$$

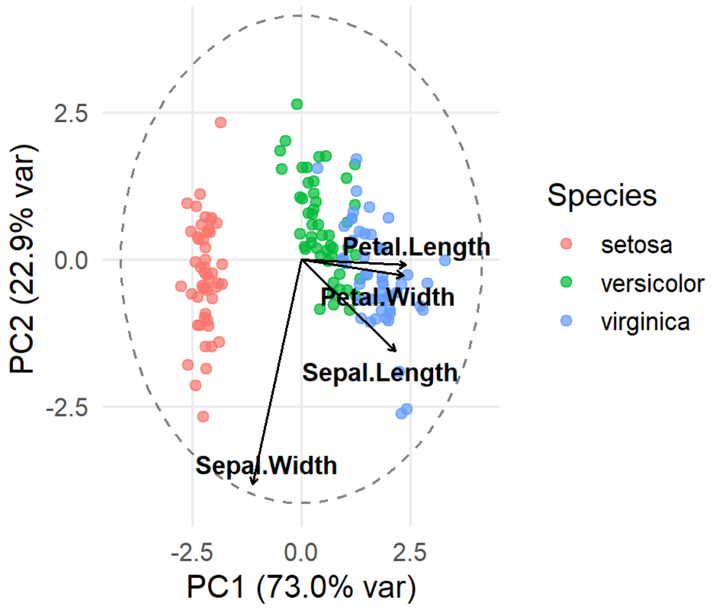
Scree Plots

- A scree plot shows how much variance each principal component explains, ordered from the largest to the smallest component.
- The “elbow” marks where additional components start contributing only small, diminishing amounts of variance.
- We use scree plots to decide how many components to keep, either by picking the elbow or by hitting a target cumulative variance (e.g., 80–95%).



Biplots

- A biplot overlays scores (observations in PC space) and loadings (variable directions) to show how observations and variables relate.
- Points that are close together in a biplot are similar in terms of the retained principal components.
- Arrow direction shows how a variable aligns witl the PCs, and arrow length reflects how strongly that variable contributes to variance in the plotted plane.
- Biplots visualise clusters, detect outliers, and communicate which variables define the main patterns captured by the first components.



Principal Components Analysis in R

```
library(tidymodels)
prcomp_out <-prcomp(iris[,1:4], center = TRUE, scale. = TRUE)
summary(prcomp_out)

Visualise:
library(ggfortify)
autoplot(prcomp_out, data = iris, colour = 'Species')
```

Interpreting PCA Results

- Inspect the loadings on PC1 and PC2. Large-magnitude loadings tell you which original variables drive movement along each axis.
- Translate loadings into plain language, for example “PC1 contrasts petal length/width against sepal width,” and check that this matches domain knowledge.
- Look at the explained variance for PC1 and PC2 to judge whether a 2D view is adequate; if these two components explain little variance, do not overinterpret the scatter.
- Check for outliers: points far from the origin or away from their group may indicate unusual combinations of variables or data issues.
- Consider whether additional PCs (e.g., PC3) improve separation, and be cautious about chasing patterns in very low-variance components that are likely to be noise.

Example: Wine

- The rattle::wine dataset contains 178 wines measured on 13 continuous chemical attributes (Alcohol, Malic acid, Phenols, Flavanoids, Colour intensity etc.)
- It also contains a column indicating wine cultivar (a factor with three classes)
- All predictors are numeric, on different scales, and often correlated.
- Why use PCA?
  - There are 13 variables, which are hard to visualise directly
  - Many features overlap in information (e.g., Phenols and Flavanoids)
  - We want to find underlying ‘latent axes’ that explain variation.
- Our goal is to reduce to 2-3 dimensions that capture most of the variance, identify key combinations of chemical properties, and interpret which chemistries drive the main principal components.

Association Rule Learning

- Association rule learning identifies interesting relationships (rules) among variables in large datasets. This is especially useful in market basket analysis and recommendation systems.
- It is an unsupervised technique: there is no target variable; instead, we mine the dataset for frequent itemsets and turn them into human-readable rules. Helps identify co-occurrence patterns - Results are descriptive, not causal.
- Finds rules like: {milk, bread} → {butter}
- Typical use cases include market basket analysis, cross-sell/upsell suggestions, recommendation, store layout and promotions, and co-usage patterns in logs or clickstreams.

Key Terms

- An Itemset is a set of one or more items that can appear together in a transaction (e.g., {milk, bread} in a shopping basket).
- Antecedent items are the items on the left-hand side of a rule, representing the condition (e.g., {milk, bread} in {milk, bread} → {butter}).
- Consequent items are the items on the right-hand side of a rule representing the outcome we expect to co-occur (e.g., {butter} in {milk, bread} → {butter}).

Key Terms

- Support measures how frequently an itemset appears in the dataset; it is the proportion (or count) of transactions containing that itemset,

$$support(X) = \frac{\#\, transactions\ with\ X}{total\ transactions}$$

- Confidence measures the reliability of a rule or strength of the implication; it is the conditional probability of the consequent given the antecedent,

$$confidence(X \rightarrow Y) = \frac{support(X \cup Y)}{support(X)}$$

- Lift measures how much more often the antecedent and consequent occur together than if they were independent;

$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(Y)}$$

- Lift > 1 indicates positive association, values of approximately 1 mean no association and values less than 1 means they co-occur less than expected.

Example

- If a shopper has milk in their basket, how likely is it they also have bread? And is that likelihood materially higher than its baseline prevalence? i.e. is P(bread|milk) meaningfully higher than just P(bread) by itself?
- If 20% of baskets contain milk, 30% contain bread, and 12% contain both:
  - support({milk, bread}) = 0.12. The support just tells us 12% of baskets contain the pair. This gauges how common the pattern is (is it worth caring about?)
  - confidence({milk} → {bread}) = $\frac{0.12}{0.2} = 0.6$. Among milk-buyers, 60% also buy bread. This answers the question: 'if a customer buys milk, how likely are they to also buy bread?'
  - Direction matters with confidence. confidence({bread} → {milk}) = $\frac{0.12}{0.3} = 0.4$
  - lift = $\frac{0.6}{0.3} = 2.00$. Milk and bread co-occur twice as often as expected by chance.
- With high support (common), high confidence (reliable) and lift>1 (beyond chance), you'd justify cross-sell prompts ('Buying milk? Consider bread'), adjacent shelf placements, bundle/discounts, or targeted coupons to milk-buyers

Interpreting Rules

- Support tells you how common the pattern is in the whole dataset, so higher support means the rule is based on enough transactions to be operationally useful rather than a curiosity.
- Confidence tells you how reliably the consequent appears when the antecedent is present, so higher confidence answers “if X occurs, how often do we also see Y?” with greater certainty.
- Lift tells you whether the co-occurrence is stronger than random chance, so lift above 1 indicates a positive association, around 1 suggests independence, and below 1 indicates the items occur together less than expected.
- Interpret the three together rather than in isolation, because very high lift with tiny support is often unstable, and very high confidence can simply reflect a very common consequent.

Example: Groceries

- We are going to use arules::Groceries (a transactions dataset of supermarket baskets) to find co-occurrence patterns between items.
- We will first explore item frequencies to understand which products are common enough to support reliable rules.
- Then we'll fit association rules, and assess rules using support, confidence, and lift to judge how common, how reliable, and how above-baseline the associations are.

Use Cases of Association Rules

- Market basket analysis: Discover products that co-occur in baskets to inform cross-sell prompts, shelf adjacency, promotions, and bundle design.
- Recommender systems: Turn high-lift rules into simple “because you bought X, consider Y” suggestions.
- Intrusion detection / security logs: Flag unusual co-occurrences of events or commands whose joint appearance is rare under normal behaviour.
- Bioinformatics: Identify genes, variants, or pathways that co-occur across samples or conditions to generate hypotheses about biological relationships.

Limitations of Association Rules

- Association rule mining can explode into thousands of trivial or redundant rules, so most value comes from smart pruning (support/confidence/lift thresholds, max rule length, redundancy filters).
- The patterns are associative, not causal, meaning co-occurrence may reflect confounders (promotions, seasonality, customer mix) rather than one item driving another.
- Confidence is biased by popular consequents and lift is unstable at very low support, so results can be misleading without minimum-support and uncertainty checks.
- Rules ignore time and order, so they tell you what appears together in final baskets, not what customers are likely to add next without sequence data.

Evaluating Association Rules

- Association rules don't have a train/test split, so you should prefer time-based holdouts to check whether rules reappear on new data.
- Thresholds like support ≥ X, confidence ≥ Y, lift ≥ Z are common but subjective; pair them with additional measures such as leverage (absolute above-chance co-occurrence) and conviction (directional strength).
- Evaluate stability by re-mining on time slices or bootstrap samples and retaining rules that recur with similar support, confidence, and lift.
- Prune redundancy by removing rules that are subsumed by shorter antecedents with similar or better metrics, or by using closed/maximal itemsets to cut duplicates.

Comparing and Selecting Models

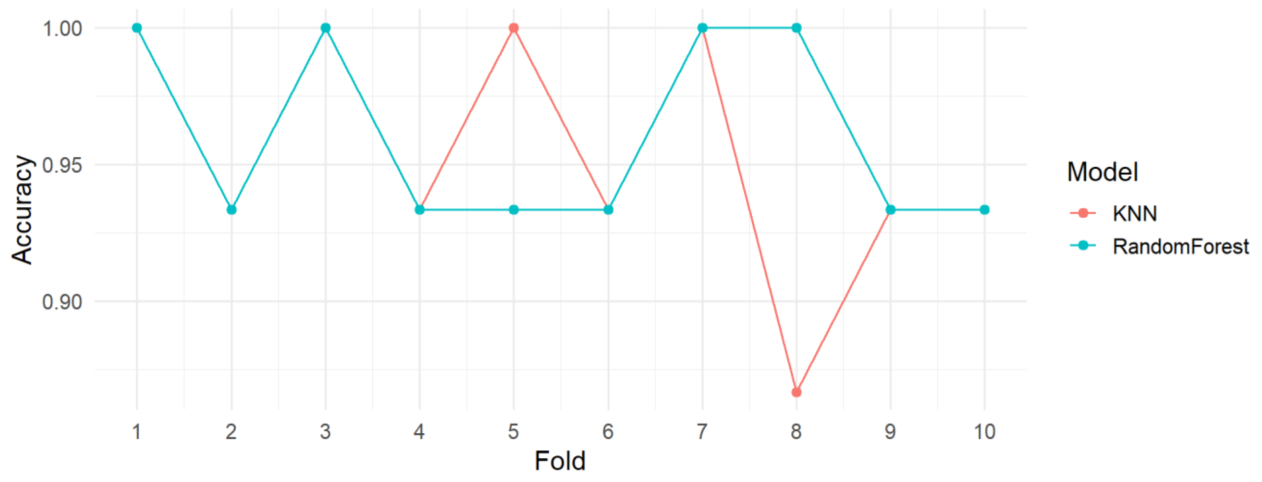
- Many models can explain the same data, so we compare them systematically rather than picking by convenience or familiarity.
- We begin with a clear objective and metric (e.g., RMSE, accuracy, ARI/NMI for clustering) that reflects the business or scientific goal.
- We check interpretability and actionability (feature effects, loadings, rules) because a slightly less accurate but explainable model can be more useful.
- We consider constraints beyond accuracy—training/inference time, memory, latency, compliance, fairness, and maintainability—since these determine how deployable a model is.
- For unsupervised models, we use internal indices (silhouette, DB) and external checks (ARI/NMI vs labels, post-hoc visuals) and prefer solutions that are both valid and interpretable.

Bias-Variance Trade-Off

- Bias is the error introduced by an overly simple model that cannot capture the true structure of the data, leading to systematic underfitting across different samples.
- Variance is the error introduced by a model that is too sensitive to the quirks of the training data, leading to predictions that fluctuate widely across different samples.
- Underfitting (high bias) appears as high training and test error, weak patterns in residuals, and little improvement with more data or training time.
- Overfitting (high variance) appears as low training error but high test error, large fold-to-fold variability in cross-validation, and unstable feature effects.

Visualising Model Performance

- Looking at performance across folds shows how stable a model is: small variation across folds suggests robust generalisation; large variation means the model is sensitive to which cases it sees.
- Per-fold plots help diagnose problems: a single outlier fold can reveal leakage, poor stratification, group contamination, or a distribution shift.



Model Simplicity vs. Accuracy

- When two models achieve similar performance, we prefer the simpler one because it is easier to interpret, debug, monitor, and explain.
- Simpler models are usually faster to train and deploy, cheaper to run, and less brittle under data drift, which lowers operational and governance risk.
- We look for diminishing returns: if the accuracy gain from extra complexity is small relative to added cost or instability, the simpler model wins.
- In parametric settings, AIC and BIC formalise parsimony by penalising model size: AIC targets out-of-sample prediction; BIC is stricter and favours smaller models as sample size grows.
- AIC/BIC apply to likelihood-based models (e.g., GLMs, Gaussian mixtures); for non-likelihood or black-box models we rely on cross-validation, holdout tests, and stability checks instead.

Final Model Evaluation

- Decide on preprocessing and hyperparameters using CV results, refit on the full training set, and touch the test set only once for a final, unbiased estimate.
- Use task-appropriate metrics: report a primary metric plus supporting ones (e.g., accuracy + per-class recall/precision; RMSE + MAE).
- Fix thresholds outside the test set: choose classification thresholds via CV or a validation split, then apply unchanged to the test set.
- Check robustness & shift: verify stability across seeds/resamples, compare train vs test feature distributions, and run simple stress tests (missingness, outliers, small noise).

Missing Data

- Real-world datasets almost always contain missing values, which arise from nonresponse, data entry errors, or system limits.
- Missingness matters because it can distort model estimates, and it can waste data if rows/columns are dropped indiscriminately.
- Before modelling, we should quantify how much is missing overall and per variable and look for patterns.
- The mechanism behind missingness determines how serious the bias can be, and which remedies are appropriate. Common options are to omit affected data, impute plausible values, or model missingness explicitly, each with trade-offs in bias and variance.

Missingness Mechanisms

MCAR — Missing Completely at Random

- Missingness is unrelated to any data (observed or unobserved).
- Complete-case analysis is unbiased; you only lose power.

MAR — Missing At Random

- Missingness depends only on observed variables (after conditioning).
- Complete-case can be biased; prefer multiple imputation or ML methods.

MNAR — Missing Not At Random

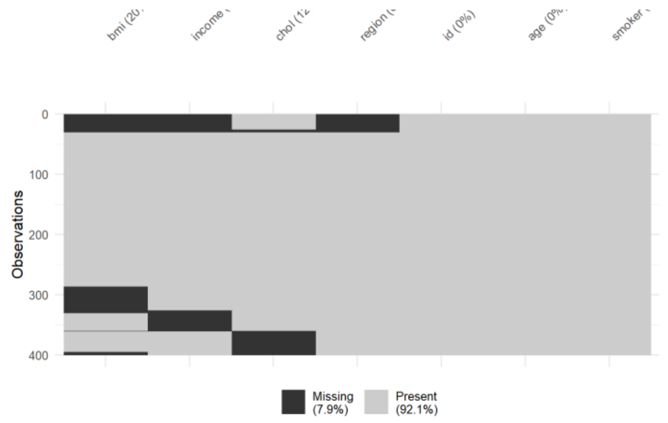
- Missingness depends on the unobserved value itself.
- Standard GLMs and routine imputation are generally biased; need explicit models of missingness and sensitivity analyses.

Practical Plan for Analysis

- Diagnose patterns: who/when is missing; visualise by covariates/time.
- Argue mechanism: MCAR/MAR/MNAR based on context and checks.
- Choose strategy:
  - MCAR → Complete cases acceptable.
  - MAR → Imputation (mean, median, last value carried forward)
  - MNAR → Jointly model the outcome and probability of missingness.
- Include predictors of missingness in imputation/models to make MAR plausible.
- Report clearly what was missing, assumptions, and sensitivity results.

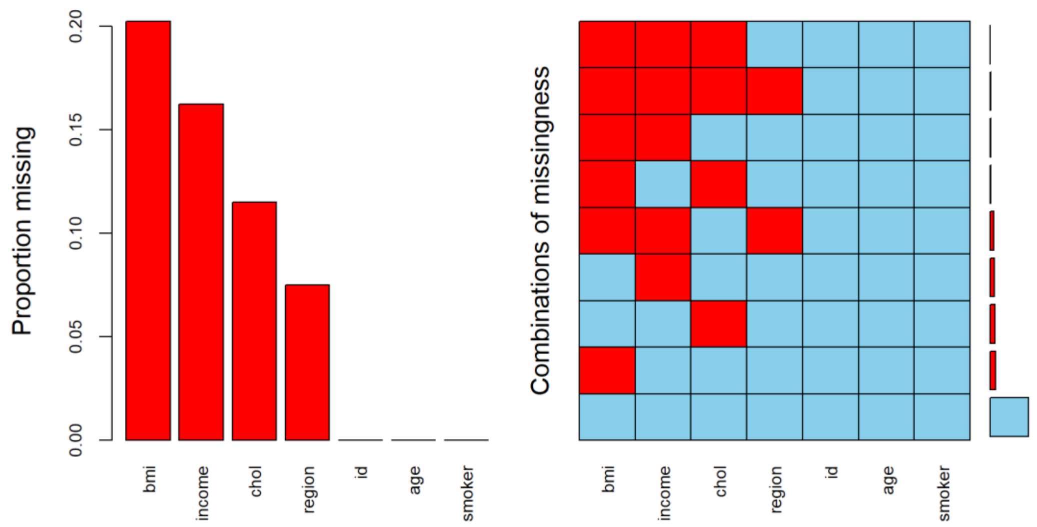
Visualising Missing Data Patterns

- The goal is to see how much is missing, where it occurs, and whether it follows patterns (by variable, by case, over time/groups) before choosing a strategy.
- naniar::vis_miss(data) shows a heatmap of missingness. Vertical stripes indicate variables mostly missing; horizontal bands indicate cases with lots missing; block patterns hint at MAR/MNAR mechanisms.



Visualising Missing Data Patterns

- VIM::aggr(data) summarises per-variable percentage missing and shows common co-missing combinations (which variables tend to be missing together), helping you spot redundancy and plan imputations.



Removing Rows with Missing Data

- Listwise deletion drops any row containing at least one NA, keeping only fully observed records.
- Use it only when missingness is small (roughly $< 5\text{–}10\%$), plausibly MCAR, and the remaining sample size and class balance are still adequate.
- Be cautious because dropping rows can bias estimates, shrink the dataset, distort class proportions, and reduce power if missingness is not MCAR.

```
rec <- recipe(y ~ ., data = df) %>%
 step_naomit(all_predictors(), all_outcomes())
```

Simple Imputation Methods

- Simple imputation fills missing values with a single constant per variable: mean/median for numeric and mode for categorical, so you keep all rows and can fit models that don't accept NAs.
- Use when missingness is low and plausibly MCAR/MAR.
- This is fast, reproducible, and easy to explain. However, it can bias estimates, shrink variance, and distort correlations (creates spikes at the fill value).

```
rec <- recipe(y ~ ., data = df) %>%
step_indicate_na(all_predictors()) %>% # flags first
  # simple imputations
   step_impute_median(all_numeric_predictors()) %>%
   step_impute_mode(all_nominal_predictors())
```

Advanced Imputation Methods

- k-Nearest Neighbours (kNN) imputation fills a missing value with information from the k most similar rows (average for numeric, mode for categorical), so it can capture non-linear relations among features. It requires standardising predictors, a sensible distance metric, and a k chosen by validation; it can be slow on large datasets and fragile when many variables are missing.
- Multiple Imputation by Chained Equations (MICE) creates (more than 1) complete datasets by iteratively modelling each variable with missingness as a function of the others, adding randomness to reflect imputation uncertainty

When Not to Impute

- When the fact that it’s missing is informative. If absence carries meaning (e.g., “no test ordered” implies low risk; “no click” implies disinterest), imputing a guess can erase signal.
  - Treat “missing” as its own level for categorical variables. Add an explicit “Missing” category so models can learn that pattern rather than forcing a possibly wrong value.
  - Flag missingness for numeric variables instead of hiding it. Add a binary indicator (e.g., age_missing = 1/0) and, if needed, fill the numeric with a neutral value (median or 0) so algorithms can run while preserving the “was missing” signal.
- Model missingness explicitly when it’s likely MNAR. Use two-part models (first predict missing vs observed, then model the outcome given observed).

Dealing with Outliers

- Outliers are observations that are unusually far from the bulk of the data; what counts as “far” is context-dependent and can be univariate (one feature) or multivariate (a combination).
- They arise from data errors (typos, unit mix-ups, sensor glitches), sampling quirks (rare subgroups), or genuine extremes (fraud, spikes, rare events).
- Outliers can distort estimates - pulling means, inflating variances, bending regression lines, dominating distance-based methods (k-means, PCA), and triggering model assumption violations.
- Start with visual checks (histograms, boxplots, scatter/QQ plots.
- Investigate provenance before acting: confirm data entry, check timestamps and duplicates, and ask domain experts whether the value is plausible.

Removing or Capping Outliers

- Remove observations only when you have clear evidence of error (impossible values, unit mix-ups, duplicates, sensor glitches), and document the rule you used.
- If extremes are plausible but too influential, cap (Winsorise) the tails at chosen percentiles (e.g., 1st /99th or 5th /95th) to limit leverage while keeping rows. Use group-wise caps if distributions differ by segment.
- After changes, recheck distributions and model fit (QQ plots, residuals), and keep an uncapped copy for auditing. Prefer doing these steps inside resampling to avoid leakage.

Outlier Treatment in Recipes

```
rec_outliers <- recipe(y ~ ., data = df) %>%
# 1) Tame heavy tails
  step_log(income, offset = 1)

# 2) Winsorise (cap) a variable using training-set quantiles
  %>% step_mutate(income=pmin(pmax(income, !!caps_income[1]),
!!caps_income[2]))

# 3) Standardise, then create an outlier flag (|z| > 3)
  %>% step_normalize(all_numeric_predictors())
  %>% step_mutate(income_outlier = as.integer(abs(income) > 3))
```