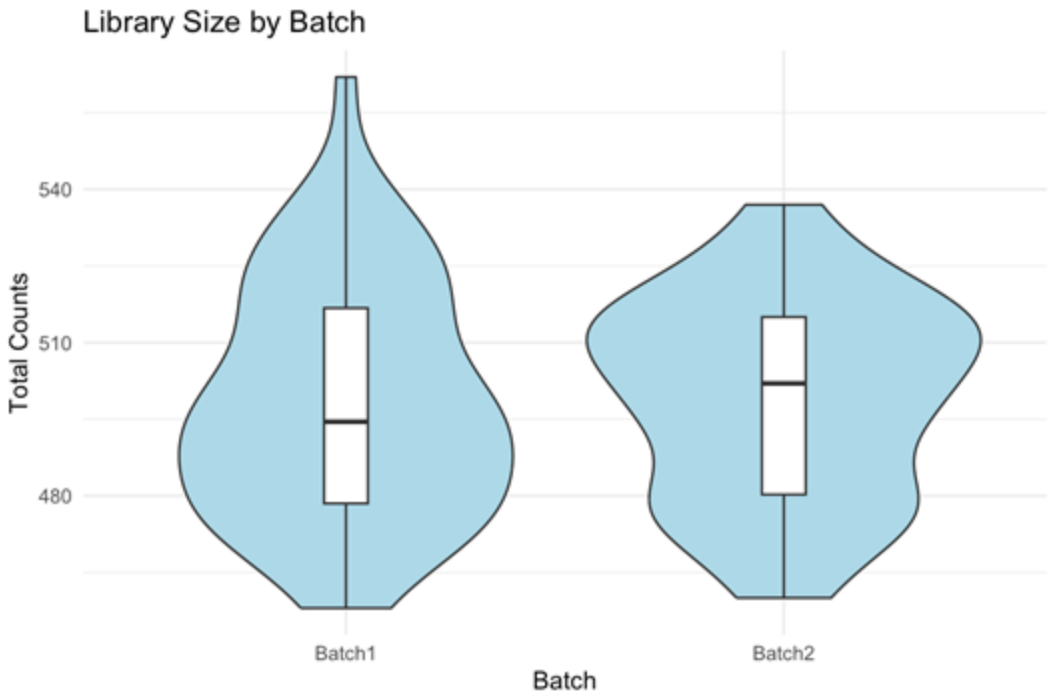
Data Preprocessing// Distance Metrics // Clustering // Dimensionality Reduction // Correlation

Data Normalization and Scaling

Purpose of normalization & scaling: Ensure features contribute comparably to analyses—avoiding domination by variables with larger ranges or units.

Common normalizations:

- **Log or variance-stabilizing transforms:** Mitigate skew and heteroskedasticity in count data (e.g. log(x+1), VST).
- **Centering:** Subtract the mean (or median) from each feature so it has zero average.
- **Scaling to unit variance:** Divide by the standard deviation (or interquartile range) so each feature has comparable spread.
- **Min–max scaling:** Linearly rescale features to a fixed range (e.g. [0,1]), useful when bounds are meaningful.



Data Similarity

Question: How do I decide whether two data points are similar?

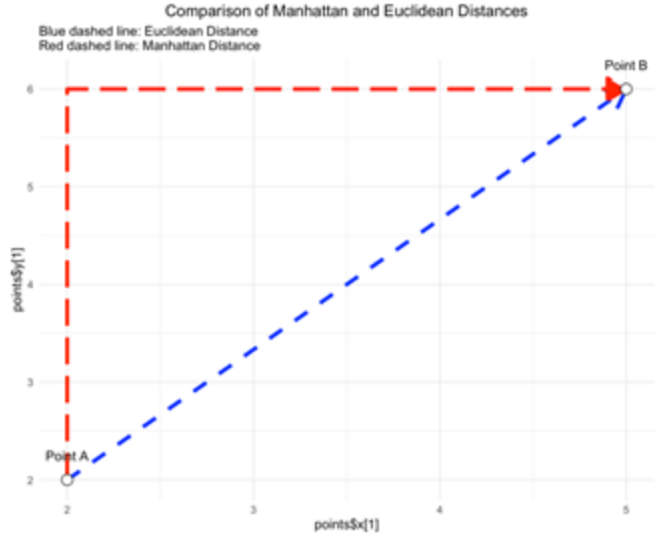
- Cells
- Samples
- Patients

By Measuring Distance:

- **Distance as a measure of dissimilarity:** Quantifies how “far apart” two observations are in feature space, giving data the notion of similarity.
- **Choice shapes results:** From k-means clustering to k-nearest neighbors and manifold learning, the distance metric determines which points group together or influence each other.
- **Scale sensitivity vs. invariance:** Euclidean/Manhattan emphasize absolute differences (sensitive to scale), while Cosine/Correlation focus on patterns or co-variation (scale-invariant).
- **Practically:** Thoughtful metric choice - with appropriate preprocessing - ensures your analyses genuinely capture the similarities and differences that matter for your scientific questions.

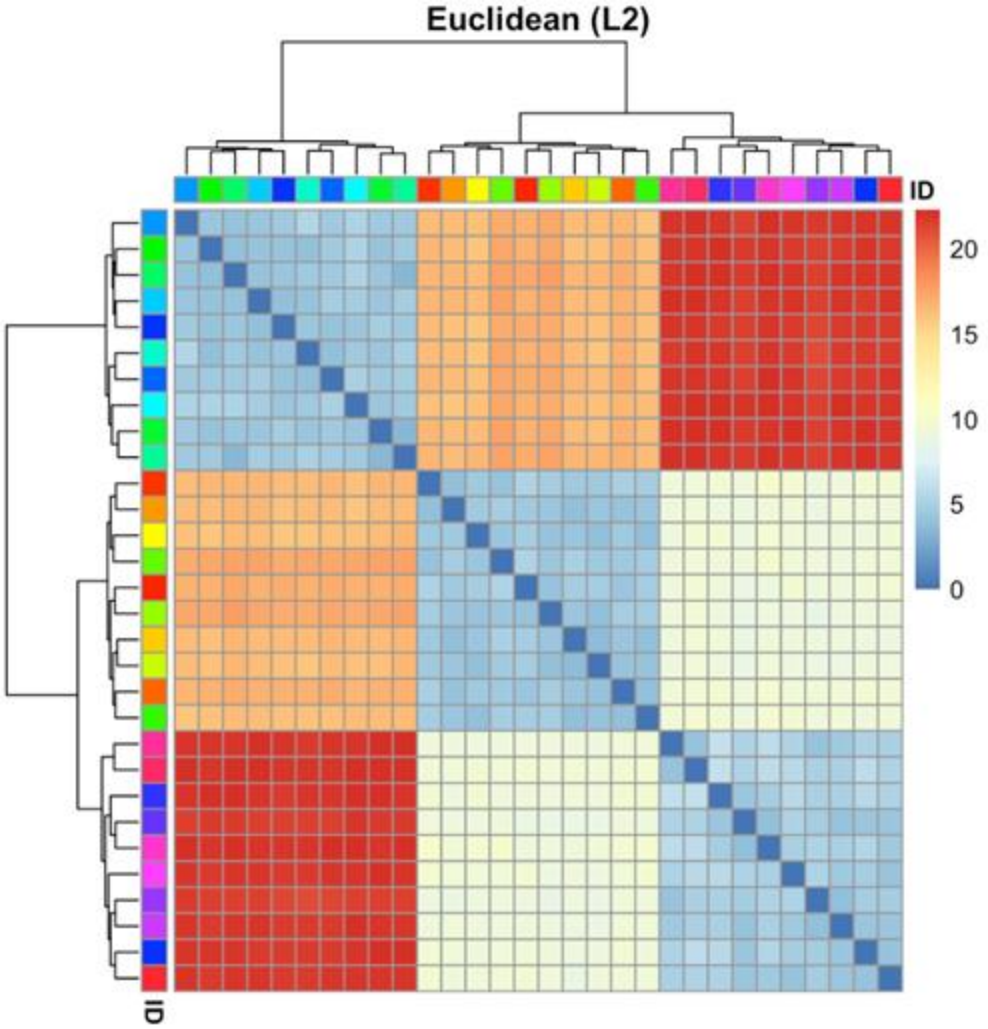
L1 / L2 Distance

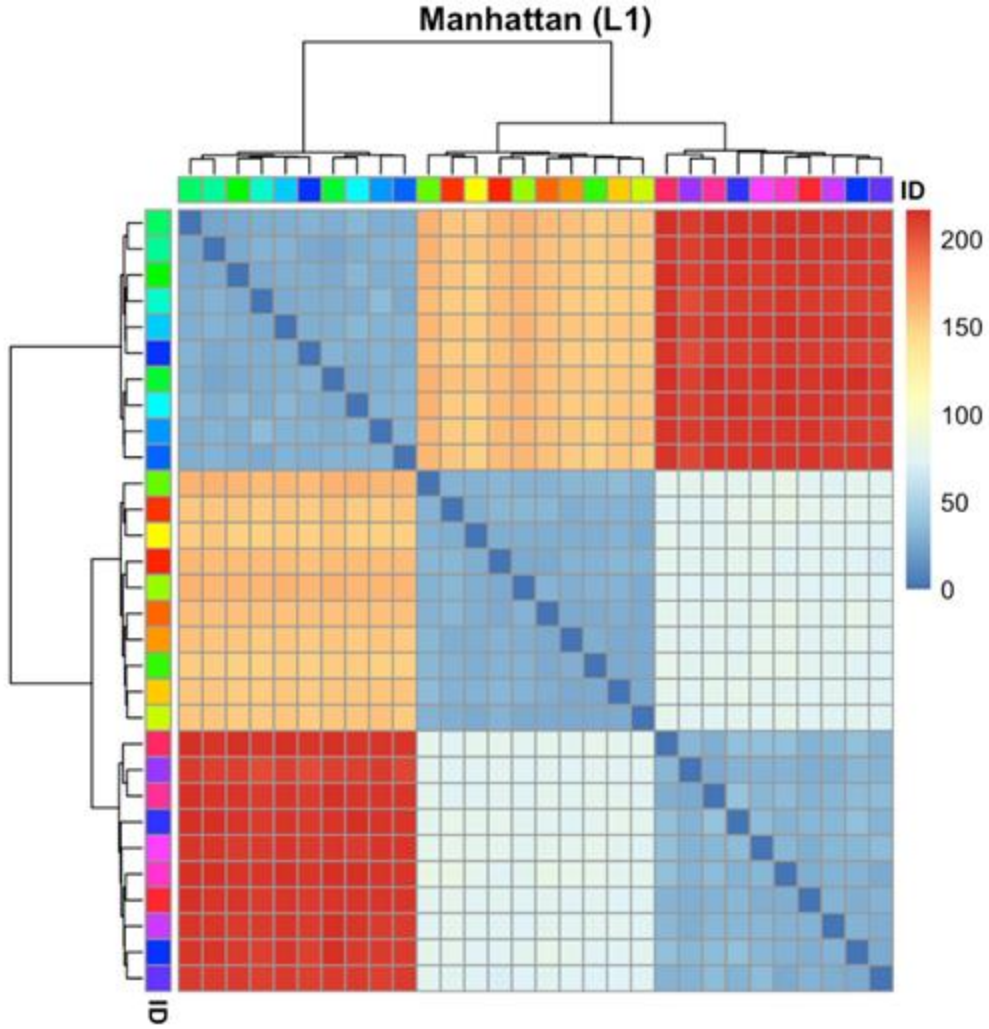
- **L_1 (Manhattan)**: Summarizes the total absolute difference across all dimensions—robust to outliers and especially useful when you want to penalize all deviations equally.
- **L_2 (Euclidean)**: Measures the straight-line ("as-the-crow-flies") distance—emphasizes larger discrepancies and aligns with geometric intuition, but can be dominated by extreme values.




```
expr_mat
#>           Gene1      Gene2      Gene3       Gene4
#> Sample1 -0.5605 -0.23017749  1.5587083  0.07050839
#> Sample2 -0.2302  1.55870831  0.0705084  0.12928774
#> Sample3  1.5587  0.07050772  0.1292877  1.71506499
#> Sample4  0.0705  0.12928774  1.71506499  0.46091621
#> Sample5  0.1293  1.71506499  0.4609162 -1.26506123

# 2) Compute distances
eu_dist <- dist(expr_mat, method = "euclidean")
man_dist <- dist(expr_mat, method = "manhattan")

# 3) Inspect the 'dist' object
eu_dist
#>           Sample1   Sample2   Sample3   Sample4
#> Sample2 0.000000
#> Sample3 2.236068 2.236068
#> Sample4 2.309401 2.309401 1.732051
#> Sample5 2.902622 2.902622 2.598562 2.415229
```

L1 / L2 Distance



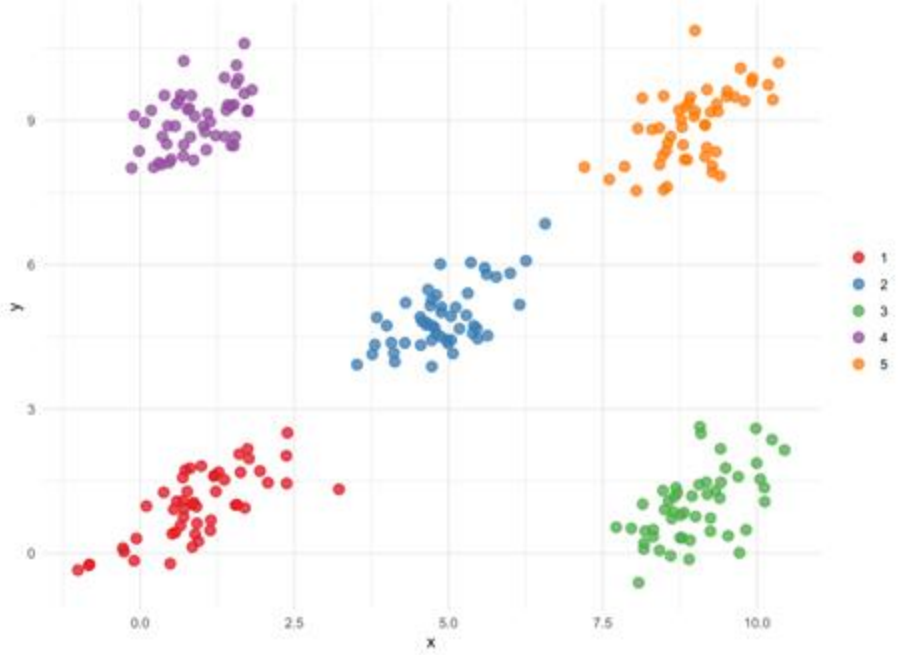


Metric	Formula	Intuition	Strength / weakness
Manhattan (L^1)	$d = \sum_i \lvert x_i - y_i \rvert$	Adds absolute coordinate differences — "city-block" distance	Robust to outliers; ignores correlations between variables
Euclidean (L^2)	$d = \sqrt{\sum_i (x_i - y_i)^2}$	Straight-line distance in Euclidean space	Intuitive; sensitive to scale and extreme values
Cosine	$1 - \cos\theta = 1 - \frac{x \cdot y}{\lVert x \rVert \, \lVert y \rVert}$	Angle between vectors — compares orientation, not length	Scale-free; discards magnitude information
Jaccard	$d = 1 - \frac{\lvert A \cap B \rvert}{\lvert A \cup B \rvert}$	Measures set overlap for binary data	Ideal for presence/absence profiles; ignores abundance; sensitive to rare features
Bray–Curtis	$d = \frac{\sum_i \lvert x_i - y_i \rvert}{\sum_i (x_i + y_i)}$	Normalized L_1 for count/abundance data—compares differences relative to total	Down-weights joint zeros; robust for ecological/OTU tables; sensitive to sampling depth
Hamming	$d = \sum_i \mathbf{1}(x_i \neq y_i)$	Counts mismatches between categorical or binary vectors	Simple and interpretable; all positions equally weighted; not suitable for continuous measurements
Mahalanobis	$d = \sqrt{(x-y)^\top \Sigma^{-1} (x-y)}$	"Whitened" Euclidean distance —accounts for feature covariance	Adjusts for correlations and scale; requires an invertible covariance matrix; sensitive to noise

```r
# Install/load required packages
# install.packages(c("proxy", "vegan"))
library(proxy)    # for cosine, Jaccard, Hamming, Mahalanobis
library(vegan)    # for Bray–Curtis

data_num <- matrix(rnorm(10 * 5), nrow = 10,
                   dimnames = list(paste0("S", 1:10),
                                   paste0("G", 1:5)))
data_bin <- matrix(sample(0:1, 10 * 5, replace = TRUE), nrow = 10,
                   dimnames = list(paste0("S", 1:10),
                                   paste0("F", 1:5)))
# Euclidean (L2)
dist_euc <- dist(data_num, method = "euclidean")

# Manhattan (L1)
dist_man <- dist(data_num, method = "manhattan")

# Cosine distance (1 − cosine similarity)
dist_cos <- proxy::dist(data_num, method = "cosine")

# Jaccard distance (for binary data)
dist_jac <- proxy::dist(data_bin, method = "Jaccard")

# Bray–Curtis dissimilarity (abundance/count data)
dist_bc  <- vegan::vegdist(data_num, method = "bray")

# Hamming distance (for binary or categorical vectors)
dist_ham <- proxy::dist(data_bin, method = "Hamming")

# Mahalanobis distance (accounts for covariance structure)
cov_mat  <- cov(data_num)
dist_mah  <- proxy::dist(data_num, method = "Mahalanobis", cov = cov_mat)
```

Clustering

Goal of clustering is to group observations so that items in the same cluster are more similar to each other than to those in other clusters—revealing the inherent structure in your data without prior labels.



- **Explore & summarize:** Uncover hidden patterns or subpopulations in high-dimensional datasets (e.g. cell types in single-cell RNA-seq).
- **Reduce complexity:** Replace large numbers of points with a few representative clusters for downstream analysis or visualization.
- **Detect anomalies:** Points that don’t fit well into any cluster often flag outliers or novel states.
- **Feature engineering:** Use cluster membership as a new categorical feature for predictive modeling.

Clustering Algorithms

- **Partitioning methods:** (e.g. k-means, k-medoids) assign points to a fixed number of clusters by optimizing centroids or medoids.
- **Hierarchical methods:** (agglomerative or divisive) build a tree of nested clusters—no need to pre-specify cluster count.
- **Density-based methods:** (DBSCAN, OPTICS) find clusters as dense regions in space and naturally handle irregular shapes and noise.
- **Model-based methods:** (Gaussian Mixture Models) assume data are generated by a mixture of probability distributions and fit parameters via likelihood.
- **Graph-based & spectral methods:** (spectral clustering) use eigenvectors of similarity graphs to cut data into cohesive groups.

K-Means

- **What it does:** Partitions *n* observations into *k* clusters by minimizing within-cluster variance.
- **How it works:**
  1. **Initialize** *k* centroids (randomly or via k-means++).
  2. **Assign** each point to the nearest centroid.
  3. **Recompute** each centroid as the mean of its assigned points.
  4. **Repeat** steps 2–3 until assignments no longer change (or a max iteration is reached).
- `kmeans()` is part stats package in base-R.
- **Why use it:** Simple, fast, and scales to large datasets; best when clusters are roughly spherical and equally sized.

```r
# 1) Simulate toy data: 3 blobs in 2D
set.seed(42)
x1 <- matrix(rnorm(50, mean =  0), ncol = 2)
x2 <- matrix(rnorm(50, mean =  5), ncol = 2)
x3 <- matrix(rnorm(50, mean = -5), ncol = 2)
data <- rbind(x1, x2, x3)
rownames(data) <- paste0("Pt", 1:nrow(data))

# 2) Run k-means with k = 3
km <- kmeans(data, centers = 3, nstart = 25)

# 3) Inspect results
print(km$size)            # number of points per cluster
print(km$centers)         # cluster centroids
print(table(km$cluster))  # assignment counts

# 4) (Optional) Plot the clusters
plot(data, col = km$cluster, pch = 16,
     main = "K-means Clustering (k = 3)")
points(km$centers, col = 1:3, pch = 8, cex = 2)
```

Hierarchical Clustering

- Builds a **tree** (dendrogram) of nested clusters—no need to pre-specify the number of clusters.
- **Agglomerative**:
  1. Compute all pairwise distances between observations.
  2. Treat each point as its own cluster.
  3. **Iteratively merge** the two clusters with the smallest inter-cluster distance.
  4. Repeat until all points join into one cluster.
- The choice of **linkage** defines how you measure distance between clusters at each merge.

```r
dist_mat <- dist(data, method = "euclidean")

# 3) Run hclust() with different linkage methods
hc_single   <- hclust(dist_mat, method = "single")
hc_complete <- hclust(dist_mat, method = "complete")
hc_average  <- hclust(dist_mat, method = "average")
hc_ward     <- hclust(dist_mat, method = "ward.D2")  # ward.D2 is Euclidean-based

# 4) Plot all four dendrograms side by side
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))
plot(hc_single,   main = "Single linkage")
plot(hc_complete, main = "Complete linkage")
plot(hc_average,  main = "Average linkage")
plot(hc_ward,     main = "Ward's method")
```

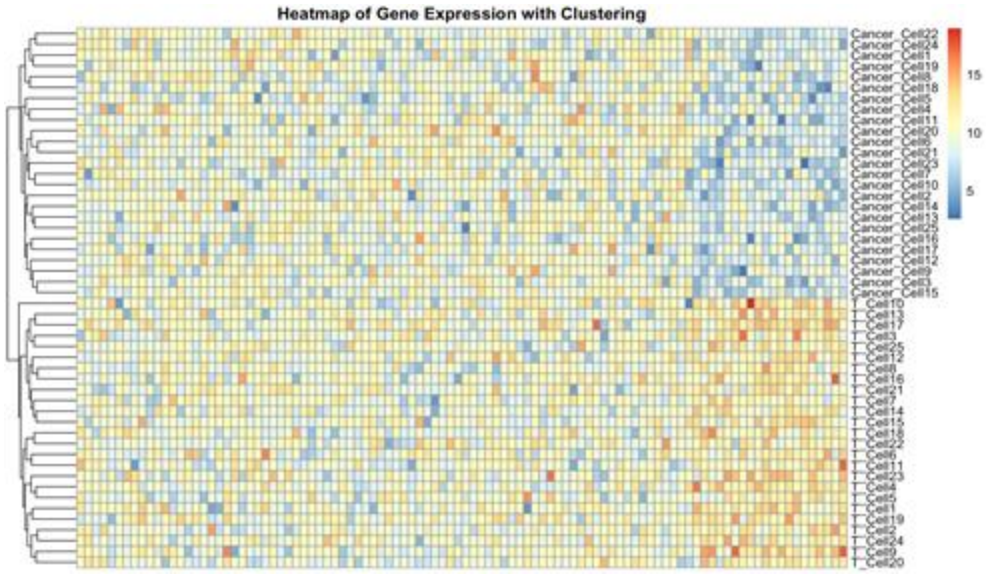
Linkage

- **Single linkage** (“nearest neighbor”)
  - **Rule:** Distance between clusters = the closest pair of points (one in each cluster).
  - **Result:** Tends to form long, “chained” clusters that can snake through feature space.
  - **Use-case:** Good when you want to detect elongated or irregular shapes, but easily derailed by noisy points.
- **Complete linkage** (“farthest neighbor”)
  - **Rule:** Distance between clusters = the furthest pair of points.
  - **Result:** Produces tight, compact, roughly spherical clusters.
  - **Use-case:** Useful when you need clearly separated groups, but can be overly influenced by outliers.
- **Average linkage** (“UPGMA”)
  - **Rule:** Distance between clusters = average of all pairwise distances between points in the two clusters.
  - **Result:** Balances between chaining and compactness, yielding moderate cluster tightness.
  - **Use-case:** A sensible default when you don’t have strong shape assumptions.
- **Ward’s method**
  - **Rule:** Merge the pair of clusters that leads to the smallest increase in total within-cluster variance.
  - **Result:** Favors equally sized, spherical clusters and minimizes the sum of squared deviations.
  - **Use-case:** Often gives the most interpretable, balanced partitioning, especially when features are on comparable scales.

Heatmaps with Dendrograms

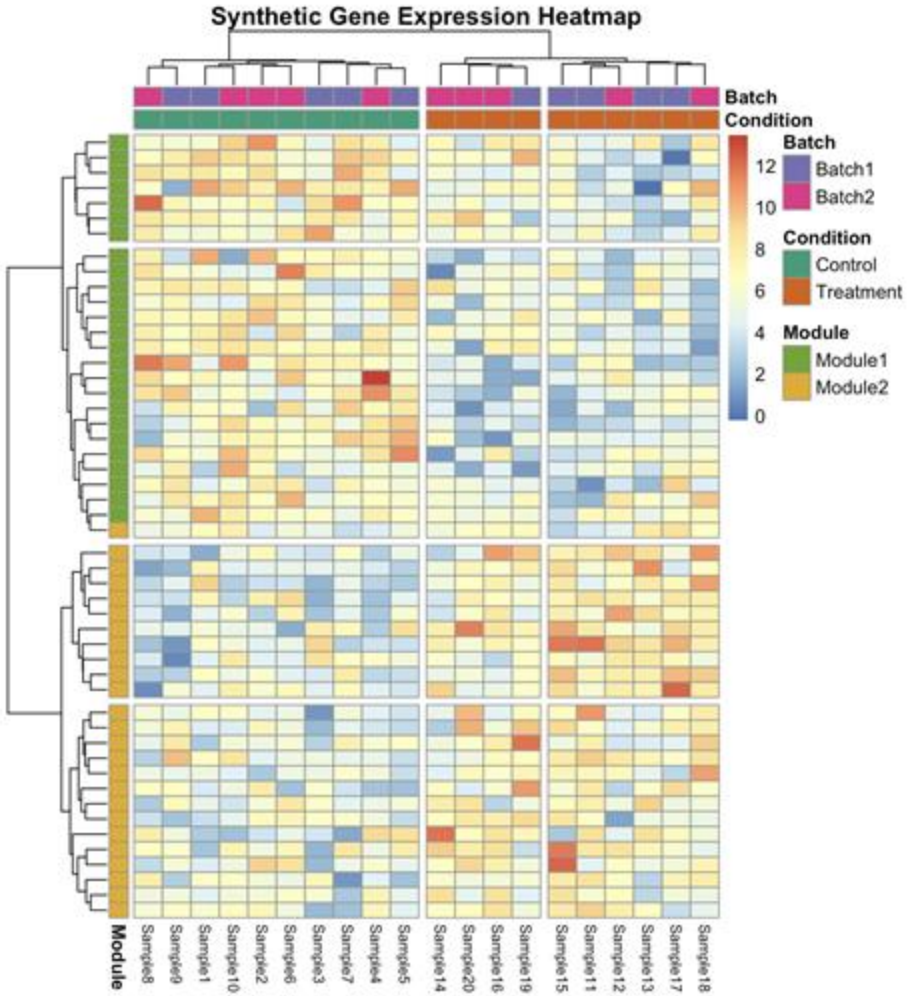
- **Integrated heatmap + clustering:** Automatically computes and draws row/column dendrograms alongside the heatmap.
- **Annotation support:** Easily add color-coded sidebars for rows and/or columns based on metadata.
- **Flexible color palettes:** Customize gradient fills, breaks, and legend labels for clear visual encoding.
- **Clustering control:** Specify distance metrics (clustering_distance_rows, clustering_distance_cols), linkage methods, or supply precomputed hclust objects.
- **Annotation colors:** Supply a named list of colors to map metadata levels to specific hues.
- **Display options:** Toggle row/column names, adjust font sizes, cell dimensions, and border visibility.
- **Output customization:** Export as a grid object for combining with other plots via grid.arrange() or advanced layouts (e.g. patchwork).wa

```
# Creating a heatmap with clustering and annotation
pheatmap(transposed_gene_expression,
         show_rownames = TRUE,
         show_colnames = FALSE,
         clustering_distance_rows = "euclidean",
         cluster_rows  = TRUE,
         cluster_cols  = FALSE,
         main = "Heatmap of Gene Expression with Clustering")
```



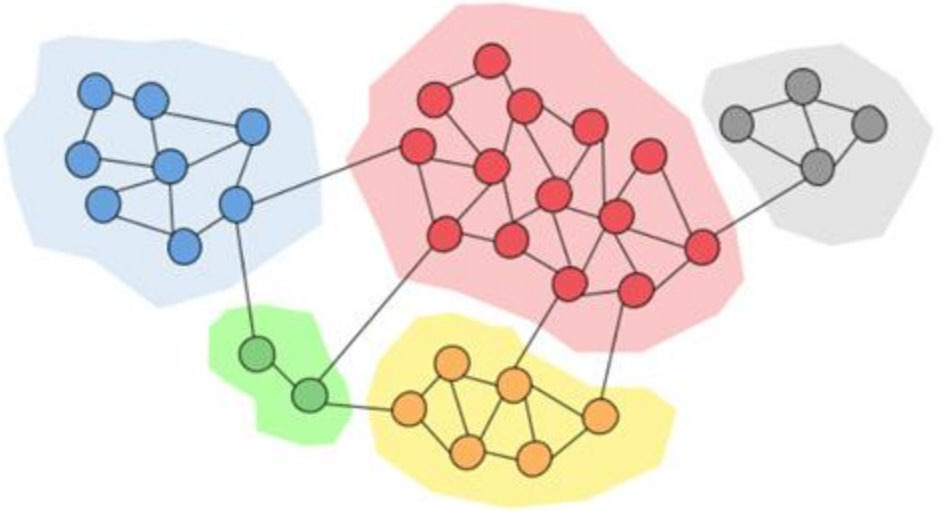
pheatmap

```r
pheatmap(
  expr,                          # data matrix: rows = genes, columns = samples
  color                 = my_palette,        # color palette for heatmap values
  breaks                = breaks_vec,        # breakpoints defining color bins
  annotation_col        = sample_annotation, # DataFrame of metadata for columns
  annotation_row        = gene_annotation,   # DataFrame of metadata for rows
  annotation_colors     = ann_colors,        # List mapping annotation levels to colors
  cluster_rows          = TRUE,              # Whether to cluster rows
  cluster_cols          = TRUE,              # Whether to cluster columns
  clustering_distance_rows = "euclidean",    # Distance metric for row clustering
  clustering_distance_cols = "manhattan",    # Distance metric for column clustering
  clustering_method     = "ward.D2",         # Linkage method for hierarchical clustering
  cutree_rows           = 4,                 # Draw rectangles to cut rows into 4 clusters
  cutree_cols           = 3,                 # Draw rectangles to cut columns into 3 clusters
  show_rownames         = FALSE,             # Hide row names for readability
  show_colnames         = TRUE,              # Show column names
  fontsize_col          = 8,                 # Font size for column labels
  cellwidth             = 15,                # Width of each cell in the heatmap (points)
  cellheight            = 8,                 # Height of each cell in the heatmap (points)
  border_color          = "grey60",          # Color of cell borders
  main                  = "Synthetic Gene Expression Heatmap", # Main title
  legend                = TRUE,              # Display the legend
  treeheight_row        = 50,                # Height of the row dendrogram (points)
  treeheight_col        = 25,                # Height of the column dendrogram (points)
  gaps_row              = c(25, 50),         # Insert gaps after these row indices
  gaps_col              = c(10, 20)          # Insert gaps after these column indices
)
```



Graph Clustering

- **Structure-based community detection:** Treats data as a graph (nodes = samples, edges = similarities) and partitions it by finding groups with dense internal connections and sparse external ones, using algorithms like Louvain, Leiden, spectral clustering, or Infomap.
- **Flexible cluster resolution:** Does not require a predefined number of clusters—instead you tune graph-construction parameters (e.g. k in k-NN) or a resolution parameter to control cluster granularity, enabling discovery of both broad and fine-scale communities.



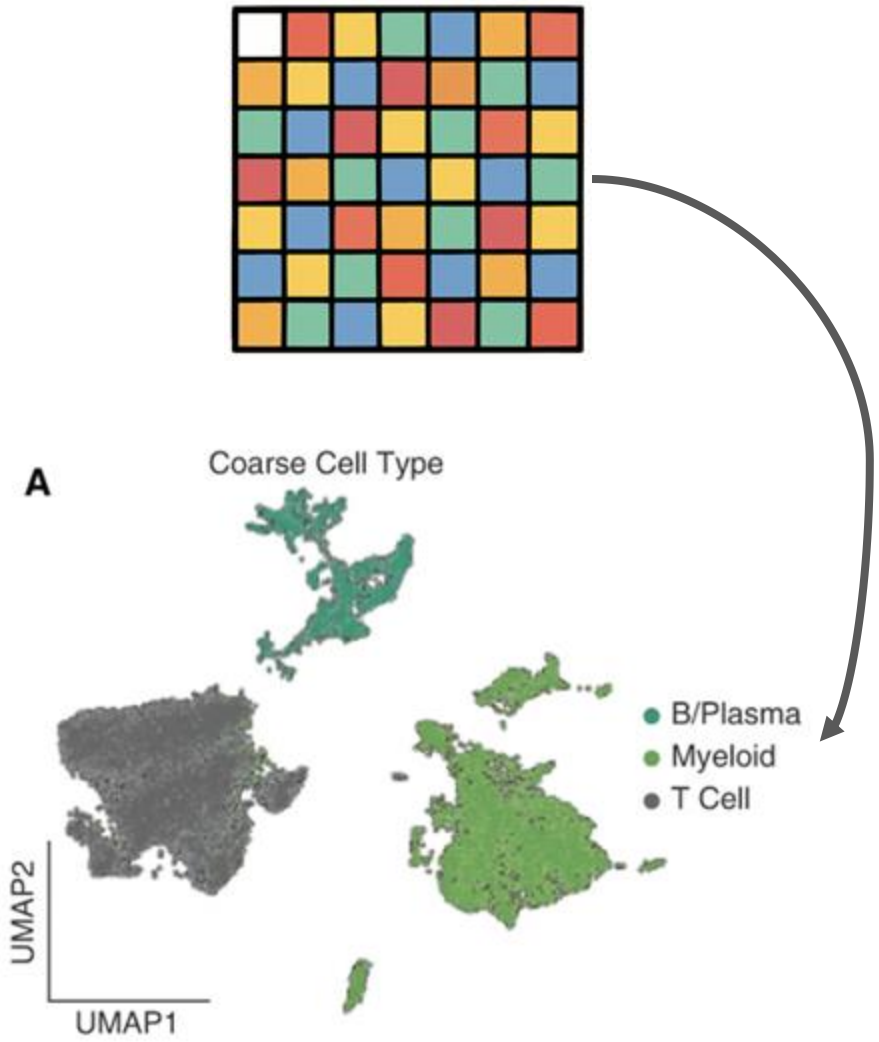
Leiden algorithm

1. Detects communities in network (graph) data by optimizing modularity or other quality functions.
2. Improves on Louvain by guaranteeing well-connected clusters and faster convergence.
3. **No need to pre-specify** the number of clusters; communities emerge from the graph structure.
4. Exposes a **resolution parameter** to tune cluster granularity (higher → more, smaller clusters; lower → fewer, larger clusters).
5. Available in R via the **leiden** or **igraph** package.

Dimensionality Reduction

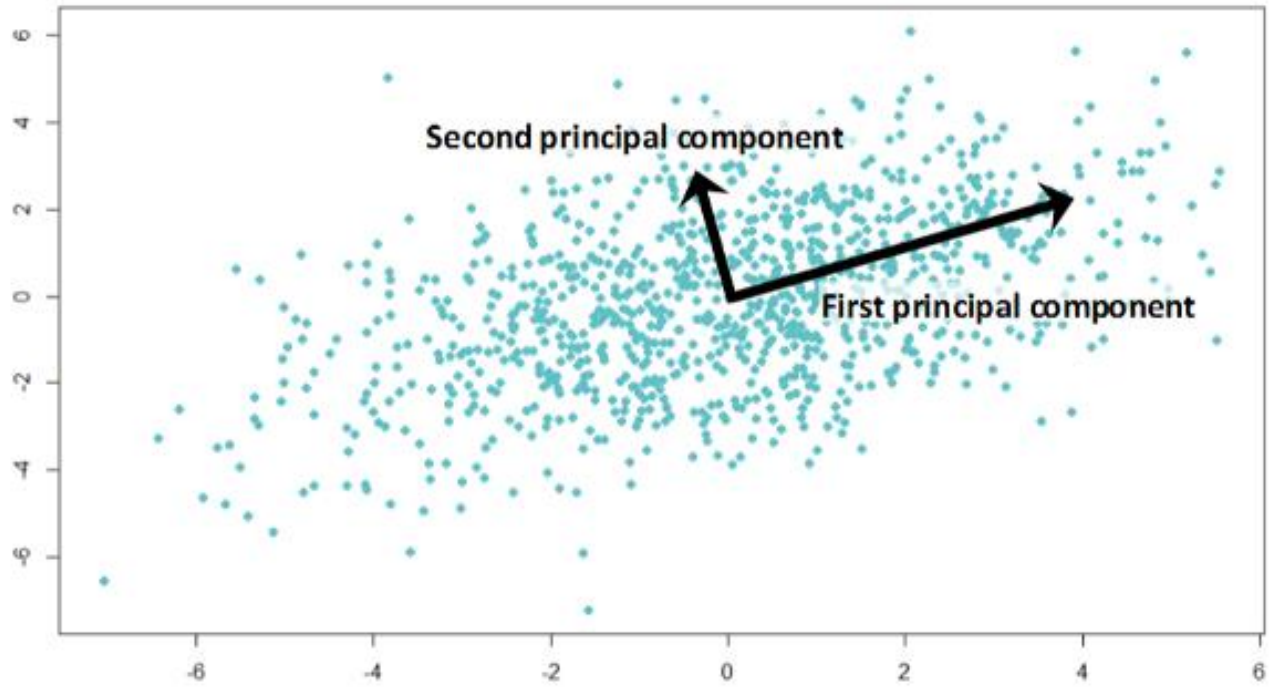
Dimensionality reduction is the process of summarizing or transforming a large set of variables into a smaller collection of representative features (components, embeddings, etc.).

- **Visualization:** Project high-dimensional data into 2–3 dimensions for intuitive plotting.
- **Denoising:** Filter out random variation and highlight true signal.
- **Efficiency:** Reduce computational and storage costs for downstream analyses.
- **Overfitting prevention:** Lower model complexity by eliminating redundant or noisy features.
- **Common methods:**
  - **Principal Component Analysis (PCA):** Linear projection maximizing variance capture.
  - **Diffusion Component Analysis:** Nonlinear axes based on random-walks over data graphs, preserving connectivity.
  - **Autoencoders:** Neural-network-based compression and reconstruction via a bottleneck layer.
  - **t-SNE / UMAP:** Nonlinear embeddings that preserve local (t-SNE) or both local and global (UMAP) neighborhood structure for visualization.



Principal Component Analysis (PCA)

- **Principal Component Analysis (PCA):** A standard technique to reduce high-dimensional data to a smaller set of orthogonal “principal components.”
- **Principal components:** Each is a linear combination of the original variables that captures a distinct axis of variation in the data.
- **Variance explained:** Components are ranked so that the kth component explains the maximum remaining variance after the first k-1.
- **First component:** Accounts for the largest share of total variance—often interpreted as the dominant “trend” or signal in the dataset.
- **Subsequent components:** Describe progressively subtler patterns, with each orthogonal to (uncorrelated with) all earlier components.



PCA in R

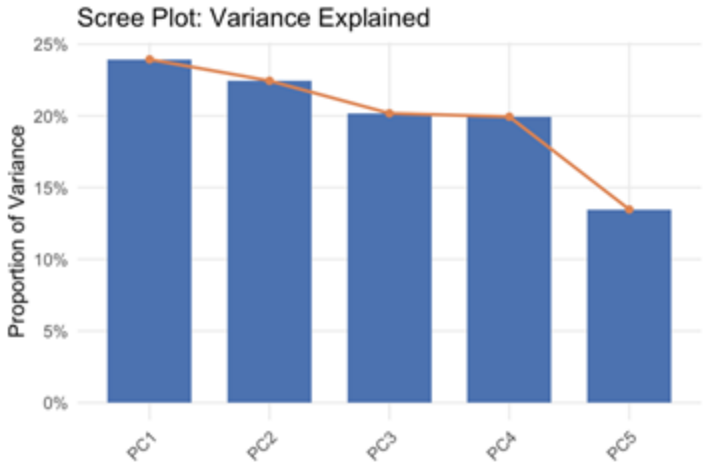
```
prcomp(data, center = TRUE, scale. = TRUE)
```

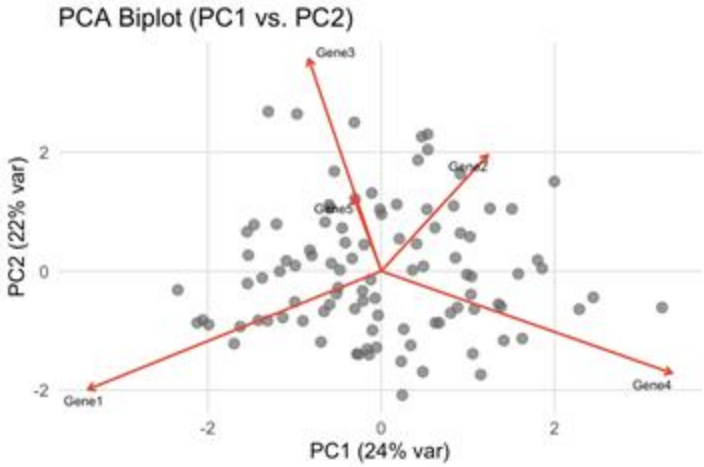
- data: numeric matrix or dataframe
- *center*
  - subtracts the mean per variable
- *scale*
  - standardize each column to have zero mean and unit variance
- *rank*
  - number of PCs to compute
- We can use summary to view components

```r
# Load necessary packages
library(ggplot2)

# Load data
data(iris)
iris_data <- iris[, 1:4]

# PCA
pca_results <- prcomp(iris_data, center = TRUE, scale. = TRUE)
print(summary(pca_results))

# Scatter plot of the first two PCs
pc_df <- data.frame(
  PC1     = pca_results$x[, 1],
  PC2     = pca_results$x[, 2],
  Species = iris$Species
)

ggplot(pc_df, aes(x = PC1, y = PC2, color = Species)) +
  geom_point() +
  labs(
    title = "PCA of Iris Dataset",
    x     = "Principal Component 1",
    y     = "Principal Component 2"
  ) +
  theme_minimal()
```

Principal Component Analysis (PCA)

```r
pca_res <- prcomp(expr, center = TRUE, scale. = TRUE)

# 3) Inspect the amount of variance explained
summary(pca_res)
#   - "Proportion of Variance" tells you how much each principal component (PC)
#     explains of the total variance.
#   - The first PC (PC1) will have the highest proportion.

plot(pca_res, type = "l",
     main = "Scree Plot: Variance Explained by PCs",
     xlab = "Principal Component",
     ylab = "Standard Deviation")

biplot(pca_res,
       choices = c(1, 2),          # PC1 vs. PC2
       scale   = 0,                # show variables in original units
       cex     = c(0.6, 0.8),      # point/text sizes
       main    = "PCA Biplot (PC1 vs PC2)")
```

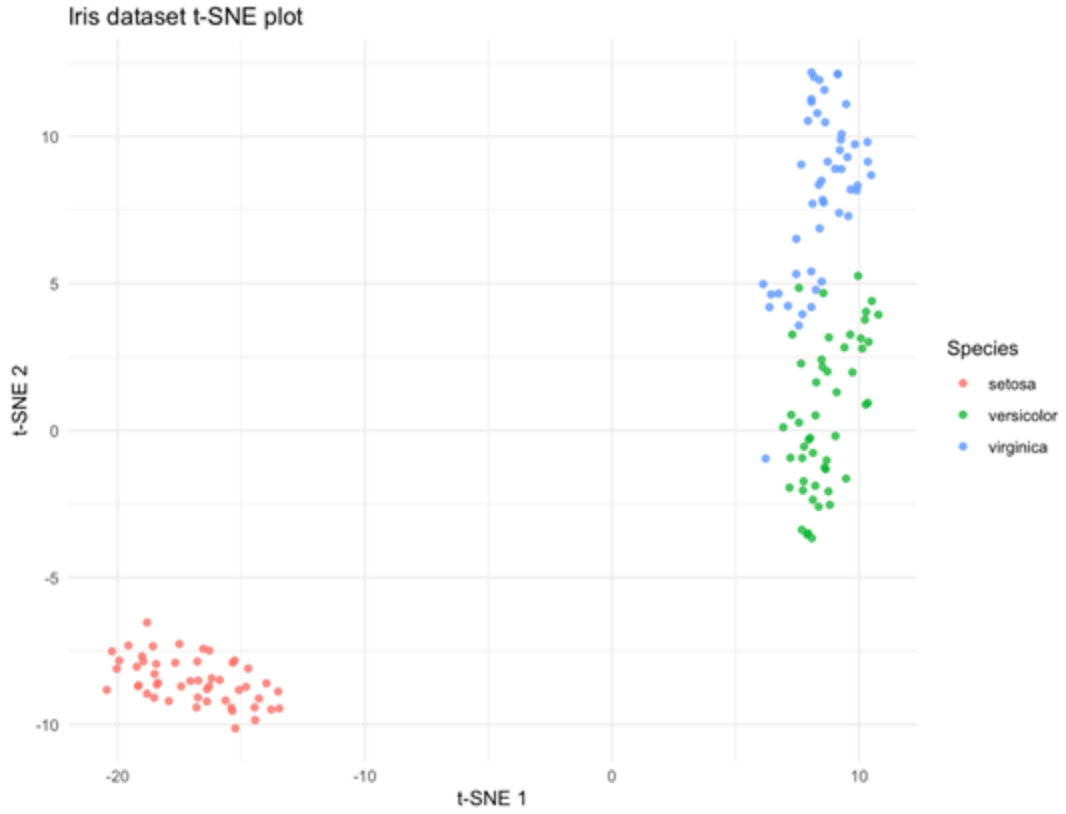




t-SNE

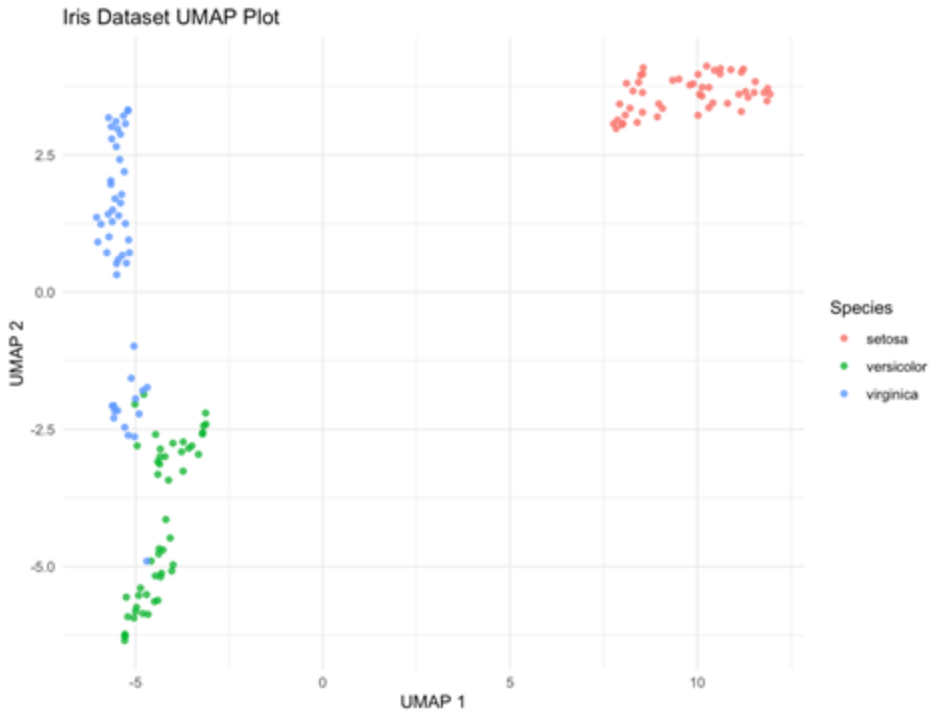
t-Distributed Stochastic Neighbor Embedding

- Attempts to preserve local structure between points in the high-dimensional state in the lower dimensional representation.
- Converts euclidean distances to probabilities in the high and low dimensions and minimizes the difference between distributions.
- Great for very complex data.
- At the expense of global structure.
- No linear mapping from low dimension to original data - unlike PCA.
- Can use Rtsne
  - `tsne_results <- Rtsne(data, dims = 2, perplexity = 30, verbose = TRUE)`
  - perplexity: hyperparameter that balances attention between local and global structure
  - dims: number of components



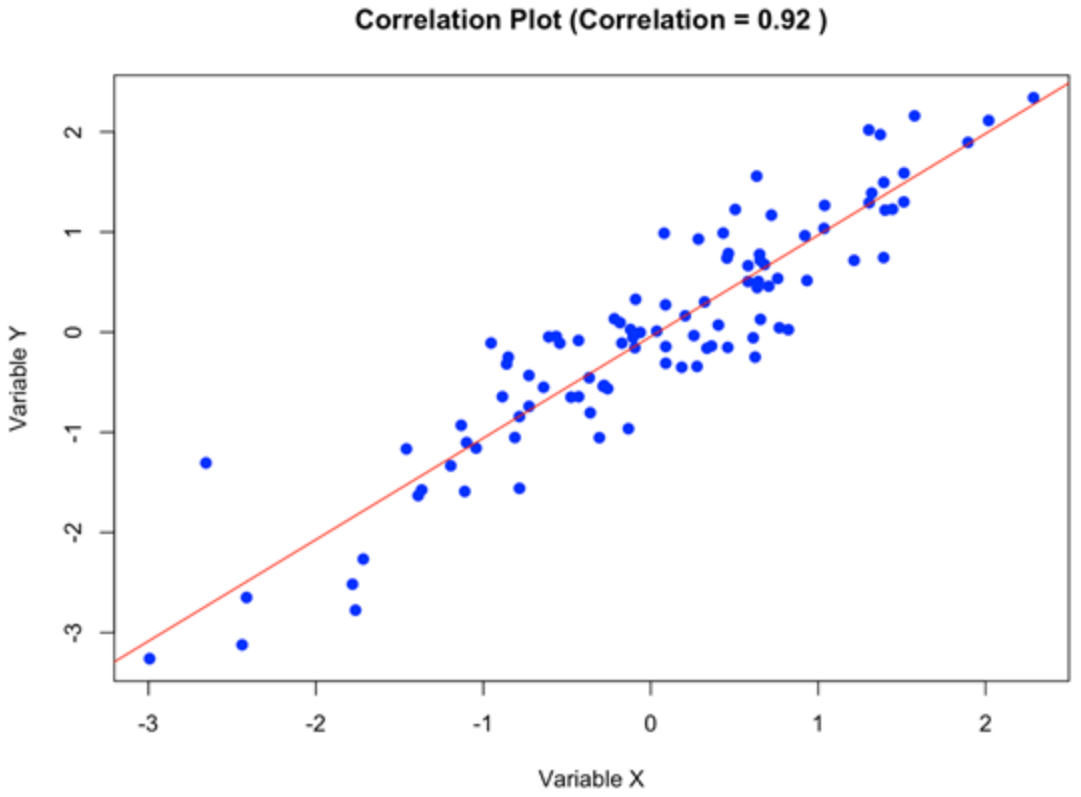
UMAP

- Uniform Manifold Approximation and Projection
- Similar to t-SNE, UMAP focuses on preserving the local structure of the data but also tries to retain more of the global structure.
- Does this by assuming a uniform distribution of data points.
- Faster than t-SNE
- Non-linear
- Can use `umap` library
  - `umap_results <- umap(data)`
  - n_neighbors: increasing preserves more global structure, computationally expensive
  - min_dist: controls the absolute min dist between points in embedding.
  - method: distance metric.. “euclidean”, etc.



Correlation

- Statistical measure that describes the strength and direction of the relationship between two variables.
- Quantifies how much changes in one variable correspond to changes in another.
- Correlation analysis is often used to explore relationships between gene expression levels across different samples or experimental conditions.



Correlation

Spearman vs. Pearson

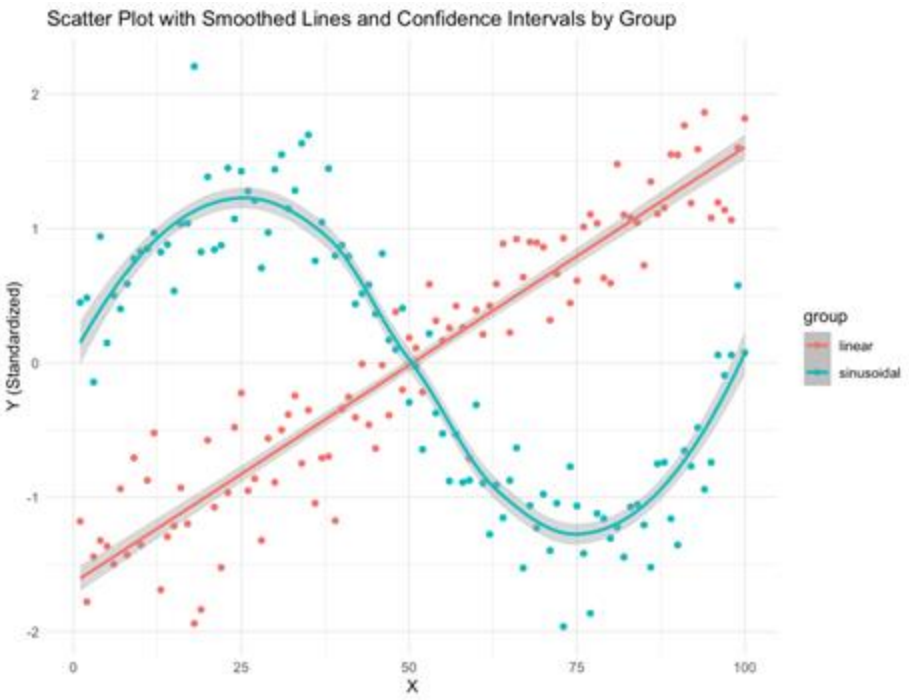
- Pearson Correlation: Measures the linear relationship between two variables. It assumes that the variables are normally distributed and have a linear relationship.
- Spearman Correlation: Measures the monotonic relationship between two variables. It does not assume linearity and is more robust to outliers and non-normal distributions.

We can use the `cor` function to compute both pearson and spearman.

`geom_smooth()` is a ggplot2 function that fits a model to a set of data points and plots a smooth lined.

- method argument specifies the model (“lm”,”loess”)
  - lm: linear regression
  - loess: locally weighted smoothing
  - ...
- Add confidence intervals with *se*.

```r
# Generate example data
set.seed(42)
x <- rnorm(100)  # Generate 100 random numbers from a standard normal distribution
y <- x + rnorm(100, mean = 0, sd = 0.5)  # Create y as a noisy version of x

# Calculate Spearman correlation coefficient
spearman_correlation <- cor(x, y, method = "spearman")
print(paste("Spearman correlation coefficient:", round(spearman_correlation, 2)))
```



Hands-on: Reading, writing, and interpreting data structures

1. Consider starting and saving a new RScript
2. Work through the blocks of code under the **Course: Clustering concepts and correlation** page
   - Review the details on how the code works in the Lecture slides for assistance
   - Put a post-it on your laptop if you get stuck, indicating for a TA to come up to you
   - Work through the blocks of code on this page, practicing in both your Rscript and the console
3. Take the next step
   - There are a list of **Additional exercises** at the bottom of the page for you to try on your own