Hierarchical Indexing

- When working with complicated datasets that need multi-dimensional organization, it is crucial in data wrangling and manipulation activities.
- Hierarchical indexing may help you properly arrange and manage datasets with numerous levels of hierarchy that you may come into during data wrangling.
- It makes it possible to effectively restructure, filter and aggregate data, which makes it easier to extract insightful information.

- Hierarchical indexing is useful in the following typical data wrangling tasks :

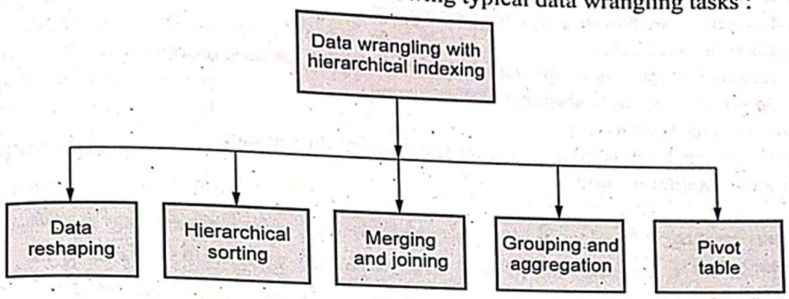


Fig. 4.2.1 Data wrangling tasks

1. **Data reshaping :** You may easily reshape data between wide and long formats using hierarchical indexing. The hierarchical links may be preserved when converting between these forms using pandas functions like "stack()" and "unstack()."
2. **Hierarchical sorting :** Data may be sorted using a hierarchy of levels thanks to hierarchical indexing. When organizing data for display or analysis, you may sort by one level while maintaining the order of the other level(s).
3. **Merging and joining :** Data may be sorted using a hierarchy of levels thanks to hierarchical indexing. When organizing data for display or analysis, you may sort by one level while maintaining the order of the other level(s).
4. **Grouping and aggregating :** The performance of group-based actions on data is facilitated by hierarchical indexing. Data may be grouped based on multiple hierarchical levels and the grouped data can then be processed using aggregate functions.
5. **Pivot tables :** Using hierarchical indexing in software like pandas, you can create pivot tables that tabulate data from many dimensions.

Discuss various functions available in matplotlib:

Matplotlib is a powerful widely used Python library for data visualization & for creating static, animated, and interactive visualizations. It provides various functions to generate and customize plots.

1. Basic Plotting Functions

These functions allow for creating standard plots like line graphs, scatter plots, bar plots, and more.

- `plot()`

  Creates a line plot or a simple 2D graph.

```python
plt.plot(x, y, label="Line Graph")
```

- `scatter()`

  Creates scatter plots to visualize individual data points.

```python
plt.scatter(x, y, color='red')
```

- `bar()` / `barh()`

  Plots vertical or horizontal bar charts.

```python
plt.bar(x, height)
plt.barh(y, width)
```

- `hist()`

  Draws histograms to visualize data distributions.

```python
plt.hist(data, bins=10)
```

```python
plt.title("Line Plot")
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.legend()
plt.show()
```

2. Styling functions

- `pie()`

  Creates a pie chart to show proportions.

```python
plt.pie(values, labels=labels)
```

- `errorbar()`

  Adds error bars to plots.

```python
plt.errorbar(x, y, yerr=errors)
```

- `stem()`

  Creates a stem plot to show discrete data points.

```python
plt.stem(x, y)
```

3. Multi-Plot Management

Functions for handling multiple subplots or figures.

- `figure()`
  Creates a new figure.

```python
plt.figure(figsize=(8, 6))
```

- `subplot()`
  Creates subplots in a grid layout.

```python
plt.subplot(2, 1, 1)  # 2 rows, 1 column, 1st plot
```

- `subplots()`
  Returns a figure and multiple axes for plotting.

```python
fig, axes = plt.subplots(2, 2)
```

- `imshow()`
  Displays images or heatmaps.

```python
plt.imshow(image_data, cmap='hot')
```

- `show()`
  Displays the plot interactively.

```python
plt.show()
```

- `savefig()`
  Saves the plot to a file.

```python
plt.savefig('plot.png')
```

Interactive Features **for image display**

Matplotlib also offers functions for creating interactive visualizations.

Example: Combining Functions

```python
import matplotlib.pyplot as plt
import numpy as np

x = np.linspace(0, 10, 100)
y = np.sin(x)

plt.figure(figsize=(8, 5))
plt.plot(x, y, label='Sine Wave', color='blue')
plt.title("Example Plot")
plt.xlabel("X Values")
plt.ylabel("Y Values")
plt.grid(True)
plt.legend()
plt.savefig("sine_wave.png")
plt.show()
```

Describe the key steps involved in the process of data wrangling.

Data Wrangling

- To make complicated data sets more accessible and understandable
- data wrangling is the act of cleaning up mistakes and merging different complex data sets.
- It is the process of cleaning, structuring, and transforming raw data into a format that is suitable for analysis.
- It involves handling messy and unstructured data, ensuring it is accurate, consistent, and usable for machine learning, statistical modelling, or reporting.

Data wrangling (also called data cleaning or preprocessing) is the process of transforming raw data into a structured, clean, and usable format for analysis or machine learning.

1. Data Collection

- **Description:** Gathering data from various sources such as databases, APIs, sensors, surveys, or logs.
- **Key considerations:**
  - Use automated methods for large datasets (e.g., SQL queries, API calls).
  - Validate the source for reliability and consistency.

2. Data Cleaning

- **Description:** Removing inaccuracies and inconsistencies in data.
- **Steps:**
  - **Handle missing data:** Use imputation techniques (e.g., mean/mode replacement) or remove affected records if appropriate.
  - **Remove duplicates:** Eliminate repeated rows or entries.
  - **Correct data types:** Ensure columns have the correct data types (e.g., dates, integers).
  - **Standardize data:** Make sure categorical variables are consistent (e.g., "Male/Female" vs. "M/F").

3. Data Transformation

- **Description:** Modifying data into a usable format.
- **Key processes:**
  - **Scaling and normalization:** Adjust numerical values for comparability (e.g., Min-Max Scaling, Standardization).
  - **Encoding:** Convert categorical data into numerical form (e.g., one-hot encoding, label encoding).
  - **Feature engineering:** Create new features that enhance data insights (e.g., aggregating timestamps into "Day of the Week").
  - **Dimensionality reduction:** Reduce the complexity of data (e.g., PCA).

4. Data Integration

- **Description**: Combining multiple datasets into a unified view.
- **Challenges**:
  - Mismatched schemas: Align columns across datasets.
  - Duplicate entries: Ensure data is not repeated after integration.
  - Handling conflicts: Resolve inconsistencies between datasets.

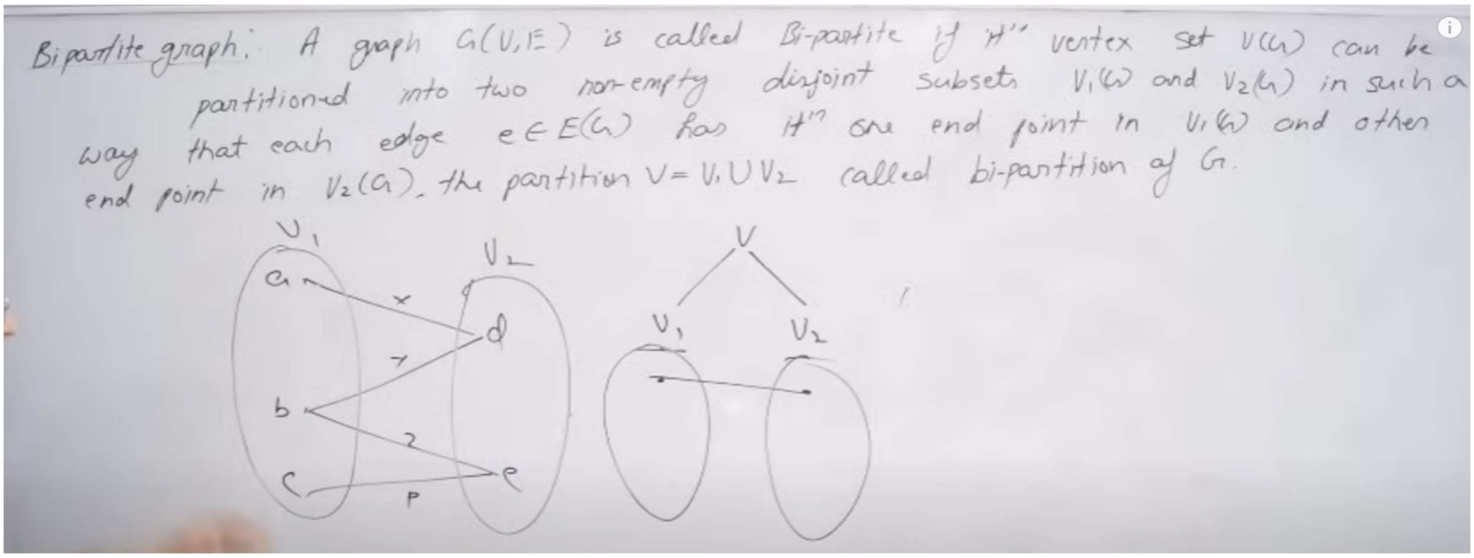
5. Data Validation

- **Description**: Ensuring data quality after cleaning and transformation.
- **Techniques**:
  - **Descriptive statistics**: Use metrics like mean, median, and standard deviation to identify anomalies.
  - **Outlier detection**: Identify extreme values using box plots or z-scores.
  - **Logic checks**: Ensure data aligns with expected relationships (e.g., "Total Sales > 0").

6. Data Storage and Management

- **Description**: Organizing cleaned data for efficient access.
- **Options**:
  - **Structured data**: Use relational databases (e.g., MySQL, PostgreSQL).
  - **Semi-structured data**: Use NoSQL solutions (e.g., MongoDB, Elasticsearch).
  - **Large datasets**: Consider distributed storage (e.g., Hadoop, AWS S3).

Explain Bipartite Graph and its applications



Example of a Bipartite Graph:

- Consider a graph where $U = \{A, B\}$ and $V = \{1, 2, 3\}$:
  - Edges: $(A, 1), (A, 2), (B, 3)$.
  - This graph is bipartite because all edges connect vertices in U to vertices in V.

Applications of Bipartite Graphs

Bipartite graphs are widely used in various domains due to their unique structure. Here are some key applications:

1. Matching Problems

- **Problem:** Pair elements from one set (e.g., jobs) with elements from another set (e.g., workers) based on preferences or constraints.
- **Example:**
  - **Job Assignment:** Vertices in set U represent workers, and set V represents jobs. Edges represent eligibility or qualification.
  - **Algorithm Used:** Maximum Bipartite Matching (using algorithms like the Hopcroft-Karp algorithm).

2. Recommendation Systems

- **Problem:** Recommend items (e.g., movies, books) to users based on past interactions.
- **Example:**
  - A bipartite graph with one set as users and the other set as items. Edges represent user preferences or interactions.
- **Advantage:** Simplifies the collaborative filtering process.

5. Resource Allocation

- **Problem:** Assign resources to tasks efficiently.
- **Example:**
  - Assign classrooms (set U) to courses (set V) where each edge represents a feasible assignment.
  - Applications: Scheduling, operations management.

6. Biological Networks

- **Problem:** Represent interactions between two biological entities.
- **Example:**
  - Protein-chemical interaction networks, where one set represents proteins and the other represents chemicals. Edges signify interactions.

7. Database Query Optimization

- **Problem:** Optimize the execution of database queries involving joins.
- **Example:**
  - Bipartite graphs model relationships between tables and attributes, simplifying query planning.

8. Image Segmentation

- **Problem:** Partition images into regions based on features like color or texture.
- **Example:**
  - Pixels and regions are represented as bipartite graphs. Edges capture similarity measures.

Advantages of Using Bipartite Graphs

1. **Efficient Algorithms:** Many problems involving bipartite graphs, such as matching, have polynomial-time solutions.
2. **Simplified Representation:** Helps model complex problems in an intuitive way.
3. **Scalability:** Bipartite structures are computationally efficient to analyze, even for large datasets

Disadvantages of Bipartite Graphs

1. **Restrictive Structure:** Not all graphs are bipartite, limiting their applicability to specific problems.
2. **Dependency on Algorithms:** Solving problems often requires specialized algorithms (e.g., matching, colouring).
3. **Representation Complexity:** While simple for binary relationships, extending to multi-layered or weighted problems can make them more complex

Explain effective graph layout techniques in data visualization.

Effective graph layouts are crucial in data visualization for understanding complex relationships between nodes and edges.

4.4.2 Graph Layout Techniques FCTH SGL3D

- A network's nodes and edges may be graphically positioned in a 2D or 3D space using graph layout techniques, commonly referred to as graph design or graph layout algorithms. The objective of graph layout is to provide an informative and visually beautiful illustration of the graph that faithfully reflects its interactions and structure.
- When evaluating and illustrating complicated networks, social networks, biological systems and other types of interrelated data, graph layout approaches are vital.

1. Force-directed layout :

- In order to establish an equilibrium state, force-directed layout algorithms mimic forces acting on the graph's nodes and edges. In order to create a balanced and aesthetically pleasing arrangement, nodes resist one another while edges function as springs. These configurations often group together nodes with tight connections to create clusters.

- **Use Cases:**
  - Social network analysis
  - Visualizing relationships in graphs like knowledge graphs
- **Advantages:**
  - Intuitive and aesthetically pleasing
  - Highlights clusters and community structures
- **Disadvantages:**
  - Computationally expensive for large graphs
  - May result in overlapping nodes in dense areas

2. Circular Layout

- **Description:** Nodes are arranged in a circular pattern, with edges connecting them inside or outside the circle.
- **Use Cases:**
  - Comparing relationships across nodes (e.g., gene interaction networks)
  - Visualization of cycles or ring-like structures
- **Advantages:**
  - Simple and symmetric representation
  - Easy to identify patterns in cyclical data
- **Disadvantages:**
  - Can become cluttered with many edges
  - Edge crossings increase with dense graphs

3. Tree Layout

- **Description**: Nodes are organized hierarchically in a branching tree structure, typically used for acyclic graphs.
- **Use Cases**:
  - Representing organizational hierarchies
  - Phylogenetic trees in biology
  - File system structures
- **Advantages**:
  - Clearly shows parent-child relationships
  - Effective for hierarchical data
- **Disadvantages**:
  - Inefficient for dense or cyclic graphs
  - Layout becomes cluttered for deep trees

4. Hierarchical Layout

- **Description**: Similar to the tree layout but allows for directed acyclic graphs (DAGs). Nodes are placed in levels based on their hierarchy.
- **Use Cases**:
  - Workflow diagrams
  - Process flows and dependency charts
- **Advantages**:
  - Emphasizes directional flows and levels
  - Highlights dependencies and precedence
- **Disadvantages**:
  - Inefficient for non-hierarchical data
  - Requires manual adjustment for dense layouts

a hierarchical pattern, often from top the bottom or left to right. It is often used to visualize data that has numerous hierarchical levels, such as that seen in organizational charts or file systems.

5. Spectral layout :

- Eigenvectors of the adjacency matrix of the network are used in spectral layout methods to place the nodes. These diagrams, which are based on spectral graph theory, are useful for representing big, sparse graphs. By using the graph's eigenvalues, spectral layouts aim to maintain the graph's structural characteristics.

- **Use Cases:**
  - Graph clustering
  - Analyzing structural properties of graphs
- **Advantages:**
  - Captures global graph structure
  - Highlights community or cluster formations
- **Disadvantages:**
  - Computationally expensive for large graphs
  - Results may be less interpretable than force-directed layouts

6. Grid Layout

- **Description:** Nodes are placed on a grid, often used for regular or lattice structures.
- **Use Cases:**
  - Visualization of regular or grid-like data (e.g., circuit diagrams)
  - Adjacency matrices
- **Advantages:**
  - Simplifies visualization of structured data
  - Easy to understand and navigate
- **Disadvantages:**
  - Ineffective for irregular graphs
  - Wastes space if the graph isn't grid-like

7. Layered layout :

- In a layered arrangement, nodes are arranged according to how far they are from a source node in Directed Acyclic Graphs (DAGs). This design is helpful for visualizing project timetables, dependencies and processes.

- **Use Cases:**
  - Flowcharts
  - Network routing diagrams
- **Advantages:**
  - Reduces clutter by clearly separating layers
  - Emphasizes progression or flow
- **Disadvantages:**
  - Requires preprocessing to assign layers
  - Can result in wasted space

8. 3D Layout

- **Description**: Nodes and edges are visualized in a 3D space, adding depth to the layout.
- **Use Cases**:
  - Large-scale graphs with overlapping structures
  - Interactive visualizations for immersive analysis
- **Advantages**:
  - Can reveal patterns hidden in 2D layouts
  - Interactive views allow for deeper exploration
- **Disadvantages**:
  - More difficult to interpret without interaction
  - Computationally expensive and requires advanced rendering

Aspect	Force-directed Techniques	Multidimensional Scaling (MDS)
Definition	A graph layout algorithm that uses physical simulations (like forces) to position nodes in a graph.	A dimensionality reduction technique that preserves pairwise distances as closely as possible.
Purpose	Visualize the structure of networks or graphs.	Project high-dimensional data into lower dimensions while preserving distance relationships.
Input	A graph with nodes and edges.	A distance matrix or raw data points.
Output	A visual graph layout with nodes and edges arranged in space.	Low-dimensional representation of data points.
Approach	Uses forces like attraction and repulsion to achieve equilibrium.	Minimizes stress (difference between original and reduced distances).
Underlying Principle	Based on physical simulations (e.g., Hooke's law, Coulomb's law).	Based on distance preservation or similarity optimization.
Data Type	Relational data (nodes and edges).	Pairwise relationships (e.g., distances, similarities).
Scalability	Can struggle with large, dense graphs due to computational costs.	Can handle moderate-sized datasets but may become computationally expensive for very large datasets.
Dimensionality	Generally 2D or 3D layouts for graph visualization.	Reduces to 2D or 3D but can handle higher dimensions.
Applications	Social network analysis, graph visualization, biological networks.	Data exploration, clustering visualization, similarity-based analysis.
Interpretability	Focuses on the structural layout of the graph.	Focuses on preserving the spatial relationships of data points.
Advantages	- Creates intuitive and visually appealing graphs. - Captures network structure effectively.	- Allows visualization of high-dimensional data. - Maintains relative distances for interpretability.
Disadvantages	- Not suitable for non-relational data. - Sensitive to parameter tuning and initial layout.	- May not fully preserve distances in low dimensions. - Sensitive to outliers.

2. Types of MDS : Metric vs. Non-metric MDS :

- Metric MDS and non-metric MDS are the two primary forms of multidimensional scaling.
- **Metric MDS :** Metric MDS maintains precisely in the lower-dimensional space the initial distances or differences between data points. Metric MDS's optimisation goal is to minimise the difference between the initial pairwise distances and the distances in the condensed space. Metric MDS uses the classical MDS method, which is popular for displaying data in euclidean space.
- **Non-metric MDS :** Rather than focusing on the precise distances, non-metric MDS places more of an emphasis on maintaining the ranking or ordinal connections of the pairwise distances. The optimal monotonic transformation for the given dissimilarity rankings is found via non-metric MDS. This MDS type is more adaptable and can work with non-euclidean data.

Force-Directed Techniques in Multidimensional Scaling (MDS)

Force-directed techniques are an intuitive and graph-based approach used in **multidimensional scaling (MDS)** to visualize high-dimensional data in lower dimensions, typically 2D or 3D. These techniques treat the points in the lower-dimensional space as objects subjected to forces, aiming to position them in a way that preserves the pairwise distances from the high-dimensional space as closely as possible.

Key Concepts in Force-Directed Techniques

1. **Force Model:**
   - The technique models the dataset as a graph, where:
     - Nodes represent the data points.
     - Edges encode the pairwise distances between the points in the original high-dimensional space.
   - Forces are applied between the nodes:
     - **Attractive forces:** Draw nodes together if their pairwise distance in the low-dimensional space is less than the original distance.
     - **Repulsive forces:** Push nodes apart if their pairwise distance in the low-dimensional space is greater than the original distance.
2. **Objective:**
   The goal is to minimize the **stress function**, which quantifies the difference between the distances in the original high-dimensional space and the reduced low-dimensional space:

$$\text{Stress} = \sqrt{\sum_{i \neq j} \left(d_{ij}^{\text{original}} - d_{ij}^{\text{low-dim}} \right)^2}$$

   Where:
   - d_{ij}^{original} = distance between points i and j in the original space.
   - $d_{ij}^{\text{low-dim}}$ = distance between points i and j in the low-dimensional space.
3. **Iterative Optimization:**
   The layout is adjusted iteratively by simulating the forces and moving the nodes in each step:
   - Nodes are repositioned until the system reaches equilibrium, i.e., when the forces are balanced and stress is minimized.

Steps in Force-Directed MDS

1. **Initialize Positions:**
   - Randomly assign initial positions to the points in the low-dimensional space.
2. **Compute Pairwise Distances:**
   - Calculate the distances between all pairs of points in both the high-dimensional and low-dimensional spaces.
3. **Simulate Forces:**
   - Compute the attractive and repulsive forces based on the differences between the original and current distances.
4. **Update Positions:**
   - Move the points in the direction dictated by the forces to reduce the stress.
5. **Iterate Until Convergence:**
   - Repeat steps 3 and 4 until the stress function stops decreasing significantly, indicating equilibrium.

Advantages of Force-Directed Techniques

1. **Intuitive Visualization:**
   - These methods produce layouts that are visually interpretable and easy to understand.
2. **Dynamic Adjustment:**
   - They provide a flexible framework for iterative refinement of the layout.
3. **Scalability:**
   - Can handle small to medium-sized datasets effectively.
4. **Insight into Relationships:**
   - Preserves the relationships between data points, making it easier to spot clusters or patterns.

Disadvantages of Force-Directed Techniques

1. **High Computational Cost**:
   - For n data points, computing pairwise distances and simulating forces requires $O(n^2)$ operations, making it computationally expensive for large datasets.
2. **Local Minima**:
   - The iterative optimization process may get stuck in local minima, leading to suboptimal layouts.
3. **Sensitivity to Initialization**:
   - The final layout may depend heavily on the initial random positions of the points.
4. **Dimensional Collapse**:
   - Some variance in the data might be lost when compressing into 2D or 3D.
5. **Difficulties with Overlapping Points**:
   - Dense datasets may result in overlapping points, making interpretation harder.

Disadvantages of Force-Directed Techniques

1. **High Computational Cost**:
   - For n data points, computing pairwise distances and simulating forces requires $O(n^2)$ operations, making it computationally expensive for large datasets.
2. **Local Minima**:
   - The iterative optimization process may get stuck in local minima, leading to suboptimal layouts.
3. **Sensitivity to Initialization**:
   - The final layout may depend heavily on the initial random positions of the points.
4. **Dimensional Collapse**:
   - Some variance in the data might be lost when compressing into 2D or 3D.
5. **Difficulties with Overlapping Points**:
   - Dense datasets may result in overlapping points, making interpretation harder.