**TerminalYatra-NCR-Pathfinder-CLI**

A Python CLI for the Delhi Metro network. 🚇 Find the shortest route by distance or the most convenient path with the fewest interchanges using Dijkstra's algorithm. Provides clear directions, calculates fares, and generates a full, color-coded network map with networkx and matplotlib.

**✨ Features Dual Pathfinding Algorithms:**

• Find the shortest path based on geographical distance.

• Find the most convenient path with the minimum number of interchanges.

• Detailed Travel Directions: Get clear, step-by-step instructions for your journey, including line changes.

• Fare Calculation: Automatically calculates the fare for the generated route based on the total distance.

• Interactive Station Selection: View a numbered list of all stations and select them by name or number.

• Full Network Visualization: Generate and save a high-resolution, color-coded map of the entire Delhi Metro network.

• Beautified CLI: A clean, user-friendly terminal interface with colors and formatted tables, powered by Colorama and Rich.

**🛠️ Tech Stack**

• Python: The core programming language.

• NetworkX: A powerful library for the creation, manipulation, and study of complex networks. Used for all pathfinding algorithms and graph modeling.

• Matplotlib: Used for generating the 2D plot of the metro network.

• Colorama & Rich: Used to create a beautiful and intuitive command-line user interface.

**🧠 How It Works:**

The Core Concepts This project isn't just a list of stations; it's a smart model of the entire Delhi Metro network. Here’s a simple breakdown of the ideas that power it.

**Modeling the Metro as a Graph** 🧩 At its heart, this project treats the entire Delhi Metro like a giant connect-the-dots puzzle. This is a concept from a field of computer science called graph theory.

* Stations become the dots (called nodes).
* Tracks between stations become the lines connecting the dots (called edges).
* The distance (in km) is the "length" of each line (called edge weight).

By representing the network this way, we can use powerful algorithms to analyze it and find the best routes.

**Finding the Shortest Path (by Distance) 🗺️** This feature is like asking a GPS for the quickest driving route.

Algorithm: It uses the classic and highly efficient Dijkstra's algorithm.

Goal: To find the path that covers the fewest kilometers from your start to your destination. The app looks at the "length" of every track segment and calculates the route that minimizes the total distance traveled.

**Finding the Easiest Path (Minimum Interchanges)** 🚶‍♂️ Sometimes the shortest path has too many line changes. This clever feature answers the question: "How can I get there with the fewest interruptions?"

Concept: To solve this, the algorithm plays a "game" on a special, imaginary version of the map where changing lines is very "expensive."

How it works:

* Traveling between stations on the same line costs just 1 point.
* Changing lines at an interchange station costs a huge 100 points.
* By finding the path with the lowest total score, the algorithm naturally discovers the route that avoids the expensive 100-point penalties, giving you the path with the minimum number of interchanges.

**Drawing the Map: Connections, Not Geography** 🎨 You might notice that a straight line (like the Red Line) appears curved on the generated map. This is intentional! The map's goal is to show the network's structure, not its real-world geography.

* The map drawing algorithm (kamada\_kawai\_layout) acts like a physics simulation:
* Springs: Each metro line is like a spring, trying to pull its connected stations together.
* Magnets: Interchange stations (like Kashmere Gate or Rajiv Chowk) act like powerful magnets, pulling multiple lines toward a central point.

This is why a straight line gets bent—the algorithm pulls it towards the interchanges it connects to. It prioritizes showing the network's overall connectivity over perfect geographical shapes, giving a much clearer, schematic view of how the entire system links together.

**The Data Source: delhi\_metro\_edges.csv** 📂 The entire project is powered by a single, simple CSV file that acts as the blueprint for the metro network. Each row represents a track segment: station\_u,station\_v,distance\_km.

* The station naming convention is key:
* Regular Stations: station-name\_line-color (e.g., saket\_yellow). This tells the program the station's name and its line.
* Interchange Stations: station-name\_ (interchange line1 line2 ...) (e.g., rajiv chowk\_ (interchange yellow blue)). This format signals that the station is an interchange, allowing the program to identify all its connecting lines automatically.

This self-contained naming scheme is very efficient, as it allows all necessary information to be stored directly in the station's name, eliminating the need for extra data files.