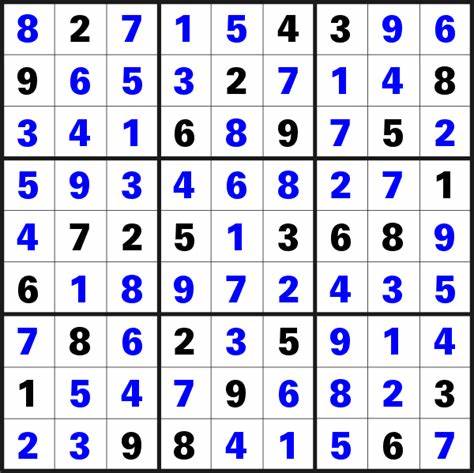
**Sudoku Solver Report**

**Title:** Sudoku Solver Using Backtracking and Visualization  
**Name:** Aryan Tomar  
**Roll No.:** 202401100300072  
**Date:** 11-03-2025

**1. Introduction**

Sudoku is a logic-based number puzzle where the goal is to fill a 9x9 grid so that each column, each row, and each of the nine 3x3 subgrids contains all the numbers from 1 to 9. It is widely known for its challenging nature and requires strategic thinking and pattern recognition. The aim of this project is to implement a Sudoku solver using the backtracking algorithm, with additional visualization for improved understanding.

**Sample Sudoku Puzzle:** 

**Sample Solved Puzzle:** 

### 2. Methodology

The solution follows a structured approach using the backtracking algorithm:

* **Input Board:** The puzzle is provided as a 9x9 grid with empty cells marked as zeroes.
* **Backtracking Algorithm:**
  + Traverse the grid to locate empty cells.
  + For each empty cell, test values from 1 to 9.
  + If the number adheres to Sudoku rules, assign it to the cell and proceed.
  + If no valid number is found, backtrack by resetting the cell to zero and retrying different numbers.
* **Visualization:** The matplotlib library is employed to visualize the puzzle both before and after solving to enhance clarity.

### 3. Code

import matplotlib.pyplot as plt

import numpy as np

# Function to check if a number can be placed in a specific cell

def is\_valid(board, row, col, num):

    # Check row

    if num in board[row]:

        return False

    # Check column

    if num in [board[r][col] for r in range(9)]:

        return False

    # Check 3x3 subgrid

    start\_row, start\_col = (row // 3) \* 3, (col // 3) \* 3

    for r in range(start\_row, start\_row + 3):

        for c in range(start\_col, start\_col + 3):

            if board[r][c] == num:

                return False

    return True

# Function to solve the Sudoku puzzle using backtracking

def solve\_sudoku(board):

    for row in range(9):

        for col in range(9):

            if board[row][col] == 0:  # Empty cell

                for num in range(1, 10):

                    if is\_valid(board, row, col, num):

                        board[row][col] = num

                        if solve\_sudoku(board):

                            return True

                        board[row][col] = 0  # Backtrack if the number is incorrect

                return False  # No valid number found, backtrack

    return True

# Function to print the Sudoku board in console

def print\_board(board):

    for row in board:

        print(" ".join(str(num) if num != 0 else '.' for num in row))

# Function to visualize the Sudoku board using matplotlib

def plot\_sudoku(board, title):

    plt.figure(figsize=(6, 6))

    plt.imshow(np.array(board), cmap="cool", vmin=0, vmax=9)

    plt.title(title)

    plt.colorbar()

    plt.show()

# Sample Sudoku puzzle (0 represents empty cells)

board = [

    [5, 3, 0, 0, 7, 0, 0, 0, 0],

    [6, 0, 0, 1, 9, 5, 0, 0, 0],

    [0, 9, 8, 0, 0, 0, 0, 6, 0],

    [8, 0, 0, 0, 6, 0, 0, 0, 3],

    [4, 0, 0, 8, 0, 3, 0, 0, 1],

    [7, 0, 0, 0, 2, 0, 0, 0, 6],

    [0, 6, 0, 0, 0, 0, 2, 8, 0],

    [0, 0, 0, 4, 1, 9, 0, 0, 5],

    [0, 0, 0, 0, 8, 0, 0, 7, 9]

]

print("Original Sudoku:")

print\_board(board)

plot\_sudoku(board, "Original Sudoku")

# Solve the Sudoku puzzle and display the result

if solve\_sudoku(board):

    print("\nSolved Sudoku:")

    print\_board(board)

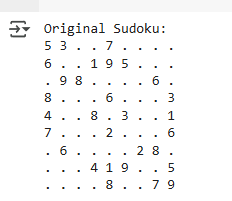
    plot\_sudoku(board, "Solved Sudoku")

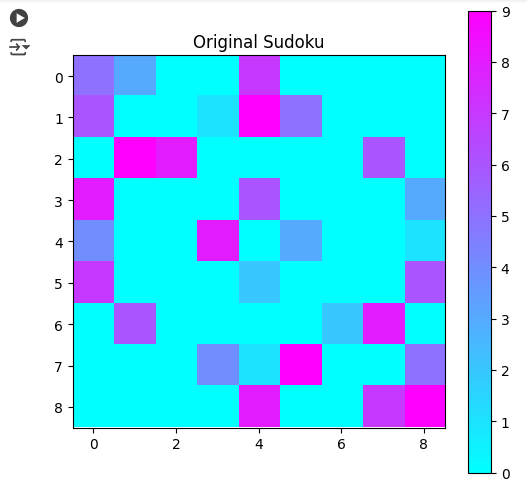
else:

    print("\nNo solution exists")

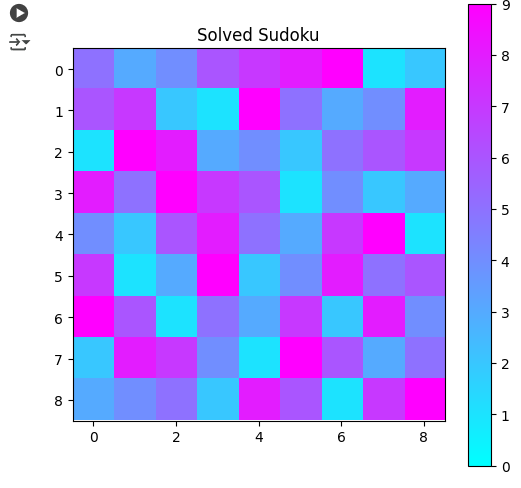
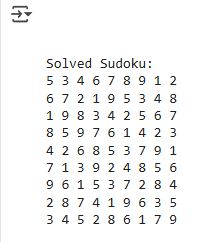
### 4. Output/Result

**Original Puzzle Screenshot:**



****

**Solved Puzzle Screenshot:**



### 5. References/Credits

* Wikipedia for Sudoku puzzle images and examples.
* matplotlib library for visualization.
* Original algorithm concept adapted from common backtracking principles.