```
import random
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
from io import StringIO
import contextlib
import traceback
import subprocess
import requests

# Define possible shapes and colors (fallbacks in case AI model fails)
shapes_2d = ['circle', 'square', 'triangle']
shapes_3d = ['sphere', 'cube', 'tetrahedron']
colors = ['red', 'blue', 'green', 'yellow', 'purple', 'orange']

# Function to call the AI model (Ollama or similar) and get shape specifications
def call_ai_model():
    try:
        # Example API call to a hypothetical endpoint for generating image specifications
        # This should be replaced with actual AI model call
        response = requests.post(
            "https://api.ollama.com/generate",
            json={"prompt": "Generate specifications for a shape drawing"}
        )
```

```
        response.raise_for_status()
        result = response.json()

        # Assuming the result contains dimension, shape, color, and additional attributes
        dimension = result.get("dimension", random.choice(['2D', '3D']))
        shape = result.get("shape", random.choice(shapes_2d if dimension == '2D' else shapes_3d))
        color = result.get("color", random.choice(colors))
        return dimension, shape, color
    except Exception as e:
        print(f"Error calling AI model: {e}")
        # Fallback to random selection
        return random.choice(['2D', '3D']), random.choice(shapes_2d if dimension == '2D' else
shapes_3d), random.choice(colors)

def generate_python_code(dimension, shape, color):
    code = f"# Generated code to draw a {dimension} shape\n"
    code += "import matplotlib.pyplot as plt\n"
    if dimension == '3D':
        code += "from mpl_toolkits.mplot3d import Axes3D\nimport numpy as np\n"
    code += "fig = plt.figure()\n"
    if dimension == '3D':
        code += "ax = fig.add_subplot(111, projection='3d')\n"
    else:
        code += "ax = fig.add_subplot(111)\n"
```

```
    if shape == 'circle':
        code += f"circle = plt.Circle((0.5, 0.5), 0.3, color='{color}')\nax.add_artist(circle)\n"
    elif shape == 'square':
        code += f"square = plt.Rectangle((0.2, 0.2), 0.6, 0.6, color='{color}')\nax.add_artist(square)\n"
    elif shape == 'triangle':
        code += f"triangle = plt.Polygon([[0.5, 0.9], [0.1, 0.1], [0.9, 0.1]], color='{color}')\nax.add_artist(triangle)\n"
    elif shape == 'sphere':
        code += f"u, v = np.mgrid[0:2*np.pi:30j, 0:np.pi:20j]\nx = np.cos(u) * np.sin(v)\ny = np.sin(u) * np.sin(v)\nz = np.cos(v)\n"
        code += f"ax.plot_surface(x, y, z, color='{color}')\n"
    elif shape == 'cube':
        code += "r = [-0.5, 0.5]\n"
        code += f"for s, e in combinations(np.array(list(product(r, r, r))), 2):\n"
        code += f"    if np.sum(np.abs(s - e)) == r[1] - r[0]:\n"
        code += f"        ax.plot3D(*zip(s, e), color='{color}')\n"
    elif shape == 'tetrahedron':
        code += "verts = [[1, 1, 1], [-1, -1, 1], [-1, 1, -1], [1, -1, -1]]\n"
        code += f"faces = [[0, 1, 2], [0, 2, 3], [0, 1, 3], [1, 2, 3]]\n"
        code += f"ax.add_collection3d(Poly3DCollection([np.array([verts[face[0]], verts[face[1]], verts[face[2]]]) for face in faces], facecolors='{color}'))\n"

    code += "ax.set_aspect('equal', 'box')\n"
    code += "ax.set_axis_off()\n" if dimension == '2D' else "ax.set_box_aspect([1,1,1])\n"
    code += "plt.show()\n"
```

```
    return code

def execute_code_and_capture_output(code):
    output = StringIO()
    try:
        with contextlib.redirect_stdout(output):
            exec(code)
    except Exception:
        return None, traceback.format_exc()
    return output.getvalue(), None

def verify_output(output, expected_shape, expected_color):
    # Basic verification (e.g., check if output contains expected terms)
    verification_passed = True
    verification_passed &= expected_shape in output
    verification_passed &= expected_color in output
    return verification_passed

def install_dependencies():
    subprocess.check_call([sys.executable, "-m", "pip", "install", "matplotlib",
"mpl_toolkits.mplot3d", "numpy", "requests"])

def main():
    install_dependencies()
```

```
    dimension, shape, color = call_ai_model()
    print(f"Generated {dimension} shape: {shape} with color: {color}")
    code = generate_python_code(dimension, shape, color)
    print("Generated Python Code:")
    print(code)

    output, error = execute_code_and_capture_output(code)
    if error:
        print("Error in executing generated code:")
        print(error)
    else:
        print("Execution output:")
        print(output)

        # Verification
        verification_passed = verify_output(output, shape, color)
        if verification_passed:
            print("Verification passed: The output matches the expected shape and color.")
        else:
            print("Verification failed: The output does not match the expected shape and color.")

if __name__ == "__main__":
    main()
```

.....

Control Flow Diagram

```
graph TD
    A[Start] --> B[Call AI Model for Shape and Color]
    B --> C{AI Model Response}
    C -->|Success| D[Generate Python Code]
    C -->|Failure| E[Fallback to Random Selection]
    D --> F[Install Dependencies]
    E --> D
    F --> G[Execute Python Code and Capture Output]
    G --> H{Error?}
    H -->|Yes| I[Print Error]
    H -->|No| J[Print Execution Output]
    J --> K[Verify Output]
    K --> L{Verification Passed?}
    L -->|Yes| M[Print Verification Passed]
    L -->|No| N[Print Verification Failed]
    M --> O[End]
    N --> O[End]
```