```python
import numpy as np
import matplotlib.pyplot as plt
import networkx as nx
import pandas as pd
from tabulate import tabulate


plt.rcParams["figure.figsize"] = (11, 7)


def friedkin_johnsen(Lam, A, x0, k, node_of_intrst, t_step_propganda_mtrx, plot_result = False) :
    n = A.shape[0] # assuming everything is dimensioned right
    I = np.eye(n)
    xx = np.zeros((n,k))
    xx[:,0] = x0
    for i in range(1,k) :
        xx[:,i] = Lam@A@xx[:,i-1] + (I-Lam)@x0
        t_step_propganda_mtrx.append(xx[0:-2,i].sum())
    if plot_result:
        plt.plot(xx.T, label=["Fake Node", "Node 0", "Node 1", "Node 2", "Node 3",
                              "Node 4", "Node 5", "Node 6", "Node 7", "Node 8", "Node 9"])
        plt.legend(bbox_to_anchor=(0.1, 1.15), loc='upper left', borderaxespad=0, ncol=6)
        plt.title(f"{node_of_intrst} ---0.5---> 'FAKE' ")
        plt.get_current_fig_manager().set_window_title(f"Results from Table 1. {node_of_intrst}")
        plt.show()


    return xx, t_step_propganda_mtrx

def draw_from_matrix(A,draw_labels=False, drw_method='arc3, rad = 0.1') :
    G = nx.from_numpy_matrix(np.matrix(A), create_using=nx.DiGraph)
    layout = nx.spring_layout(G,seed=0)
    nx.draw(G, layout, node_size=750, with_labels=True, font_weight='bold', font_size=15, connectionstyle=f"{drw_method}")
    if draw_labels :
        labels = nx.get_edge_attributes(G, "weight")
        nx.draw_networkx_edge_labels(G, pos=nx.circular_layout(), edge_labels=labels, label_pos=.33);

def addNodeToNetwork(og_network, og_opinions):
    new_network = []
    # Add new col to end of each row
    for row in range(len(og_network)):
        new_col = og_network[row].copy()
        new_col.append(0.0)
        new_network.append(new_col)
    new_row = [0.0 for col in range(len(new_network)+1)]
    mod_new_network = new_network.append(new_row)
    new_opinions = og_opinions.copy()
    new_opinions.append(1.0)
    new_network[-1][-1] = 1.0
    return new_network, new_opinions

def addBadEdgeToNetwork(adjusted_network, node, bad_node):
    for edge in range(len(adjusted_network[node])):
        adjusted_network[node][edge] = adjusted_network[node][edge] * 0.5
    adjusted_network[node][bad_node] = 0.5
    return adjusted_network

def timestepsToPTable(t_step_matrix, p_t_stp_lst):
    node_lst = [node for node in range(len(t_step_matrix))]
    time_step_dict = {f"{node}" : [] for node in range(len(t_step_matrix))}

    for t_stp in range(len(t_step_matrix[0])):
        for node in range(len(t_step_matrix)):
            time_step_dict.get(f"{node}").append(t_step_matrix[node][t_stp])

    time_step_dict.update({"p(t)": p_t_stp_lst})

    time_table = pd.DataFrame(time_step_dict, index=[f"t={t_stp}" for t_stp in range(len(t_step_matrix[0]))])
    tabHeaders = [f"node_{node}" for node in range(len(t_step_matrix))]
    tabHeaders[-1] = f"node_fake"
    tabHeaders.append("P(t)")

    return tabulate(time_table, headers=tabHeaders, tablefmt="fancy_grid")

def networkPropagandaModel(og_network, og_opnions, num_iterations, bad_node_neighbor, t_step_propganda_lst, draw_network=False, plot_result=False):
    # Add node to network
    mod_network, mod_opnions = addNodeToNetwork(og_network, og_opnions) # Adds self-pointing edge to
    # Multiplies the nodes row (it's edges) by 0.5 and add 0.5 edge from bad node
    # to the one of the nodes of the original network.
    prop_network = addBadEdgeToNetwork(mod_network, bad_node_neighbor, len(mod_network)-1)

    prop_lambda_diag_lst = np.diag(mod_opnions).tolist()
    t_step_propganda_lst.append(sum(mod_opnions[0:-2]))
    result , result_propganda_val = friedkin_johnsen(np.array(prop_lambda_diag_lst), np.array(prop_network), np.array(mod_opnions), num_iterations, bad_node_neighbor, t_step_propganda_lst, plot_result)
    return result, result_propganda_val

def plotProagandaValOverTime(p_of_t_matrix, node_of_intrest):
    plt.plot(p_of_t_matrix.T, label=f"Propaganda-Value")
    plt.legend(bbox_to_anchor=(0.1, 1.15), loc='upper left', borderaxespad=0, ncol=6)
    plt.title(f"Overall Propaganda Value when:\n{node_of_intrest} ---0.5---> 'FAKE' ")
    plt.get_current_fig_manager().set_window_title(f"Overall Propaganda Value")
    plt.show()

def plotAllProagandaValOverTime(all_prop_val):
    plt.plot(all_prop_val.T, label=["Node 0", "Node 1", "Node 2", "Node 3", "Node 4",
                                    "Node 5", "Node 6", "Node 7", "Node 8", "Node 9"])
    plt.legend(bbox_to_anchor=(0.1, 1.15), loc='upper left', borderaxespad=0, ncol=6)
    plt.title(f"All possible 'FAKE' neighbor node")
    plt.get_current_fig_manager().set_window_title(f"All possible 'FAKE' neighbor node")
    plt.show()
```

```python

proj_03_adj_ntwrk = [[0.0, 0.9, 0.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                     [0.0, 0.0, 0.2, 0.1, 0.4, 0.3, 0.0, 0.0, 0.0, 0.0],
                     [0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0],
                     [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                     [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.9, 0.0, 0.1, 0.0],
                     [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0],
                     [0.4, 0.0, 0.0, 0.2, 0.0, 0.0, 0.0, 0.4, 0.0, 0.0],
                     [0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0],
                     [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0],
                     [0.0, 0.0, 0.8, 0.0, 0.1, 0.0, 0.0, 0.0, 0.1, 0.0]]

node_opnins = [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 0.95]

TRAINING_ITERATIONS = 10


p_t_steps=[]

overall_p_vals = []

# Iterate through each row of the matrix that represent each possible neighbor,
# that the bad node's one edge could be connected to.
for possible_neigh in range(len(proj_03_adj_ntwrk)):
    print(f"Tabel 1.{possible_neigh}")
    result, p_t_steps = networkPropagandaModel(proj_03_adj_ntwrk, node_opnins, TRAINING_ITERATIONS, possible_neigh, p_t_steps, plot_result=True)
    print("------------------------------------------------------------")
    print(f" Node {possible_neigh} Directly influenced by 'fake node' test results")
    print("------------------------------------------------------------")
    plotProagandaValOverTime(np.array(p_t_steps), possible_neigh)
    temp_lst = p_t_steps.copy()
    print(timestepsToPTable(result, temp_lst))
    overall_p_vals.append(temp_lst)
    p_t_steps.clear()
    print("------------------------------------------------------------\n\n")

plotAllProagandaValOverTime(np.array(overall_p_vals))
```