```
# Import libraries
from itertools import combinations
import networkx as nx
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline
import numpy as np
import datetime, time, os, random, copy

class Graph(nx.Graph):
    def __init__(self, grph_nm=""):
        super().__init__()
        self.graph_name = f"Graph--{grph_nm}"
        self.average_degree = 0
        self.component_proba_dict = {i: [int(), float()] for i in list(np.linspace(0.001, 0.040, 20))}

    def __str__(self):
        obj_str = f"-------------- {self.graph_name} --------------\n"
        obj_str += f"Graph component count:\t{nx.number_connected_components(self)}\n"
        obj_str += f"Graph degree Average:\t{self.degree}\n"
        obj_str += f"Graph edge count:\t{self.number_of_edges()}\n"
        obj_str += f"Graph edge dict:\n"
        for node, edges in self._adj.items():
            obj_str += f"\t{node}:\t{edges}\n"
        return obj_str

    def genRandomListOfNodes(self, max=10, min=2):
        '''
        Generates list of different set pairs for each node that exsist
        in a randomly sized list of nodes that is greater than 'min' and
        equal to or less than the passed agrument 'max' value.
        @param max Represents the max number of nodes that can be
        generated from random graph generation of 'human network(s)'
        (By default max is set to 10).
        @param min represents the min numbers of nodes that can be
        generated for the instance of graph (By default min is set to
        2).
        '''
        # Randomnly selcected number of nodes to construct node_list.
        n = random.randint(min, max)
        # Temp node list to use when generating all possible node pairs.
        tmp_node_list=[]
        # Iterate through each and add each node to the temp list and
        # add the nodes to this instance of a graph.
        for node in range(0, n):
            tmp_node_list.append(f"n_{node}")
            self.add_node(f"n_{node}")
            # Generates every possible pair combinations for hte list of
            # nodes that exsist in this instance of a generated graph.
            n_lst=list(combinations(tmp_node_list, 2))
        # Returning list of all possible nodes pairs of every node that
        # exsists in this instance of a generated graph.
        return n_lst, n

    def genSetListOfNodes(self, sn):
        '''
        Generates list of all possible pair combinations that can be
        formed within a list of nodes of a specified length.
        @param sn This parameter will set the length of the list holding
        all the nodes that exsist with in this instance of a generated
        graph.
        '''
        # Temp node list to use when generating all possible node pairs.
        tmp_node_list=[]
        for node in range(0, sn):
            tmp_node_list.append(f"n_{node}")
            self.add_node(f"n_{node}")
        n_lst=list(combinations(tmp_node_list, 2))
        # Returning list of nodes represented as n_i nodes
        return n_lst, sn

    def randomEdgeConnect(self, prob):
        '''
        Generates proability of an edge forming AT RANDOM for every possbile
        exsisting pair in the graph.
        @param prob The parameter represnts the weighted value for whether a edge
        will form between that pair of nodes or not.
        '''
        edge_probability = np.random.choice([0, 1], 1, p=[1-prob, prob])
        return edge_probability == [1]

    def showGraph(self):
        '''
        Show a visual representation of the graph and edges using 'networkx'.  Also
        prints out each graphs information
        '''
        # Show instance of graph information.
        # print(self)
        # Set tittle of graph
        plt.title(f"{self.graph_name}-Kamada Kawai Layout")
        # Set graph visual representation attributes.
        nx.draw(self, pos=nx.kamada_kawai_layout(self,
                                        scale=1, center=None, dim=2),
                                        with_labels=False,
                                        node_size=300, width=2.5,
                                        node_shape="8", node_color="#000000",
                                        font_color="#FFFFFF")
        # Show every other nth graph.
        plt.show()

    def calculateAvgComponentDegree(self):
        '''
        Calculate the avg. degree for this instance of graph.
        '''
        # Try and find the degree, if fail occurs due to divide by zero fail and
        # return 0 as that is the avg. degree value for the graph.
        try:
            self.average_degree = nx.number_of_nodes(self)/nx.number_of_edges(self)
            return self.average_degree
        except :
            return 0

# Create different amount of different sized graphs up to either
# specified size or randomnly between the minimum, 2, and max size
# of the graphs you wanted generated.
amount_of_graphs = 10
max_size_of_graphs = 100
min_size_of_graphs = 2
set_size_of_graphs = max_size_of_graphs

list_of_graphs = []

# Iterate through and create instances of graph, then append them to a list of
# graphs to latter use for iterating through 'Graph' objects.
for grph_indx in range(1, amount_of_graphs+1):
    temp_graph_obj = Graph(grph_nm=f"{grph_indx}")
    # Iterate through each probability between 1 and 0 for edge forming between
    # two nodes.
    for edge_prob in temp_graph_obj.component_proba_dict.keys():
        # node_pair_set_list, amount_of_nodes =
        # temp_graph_obj.genRandomListOfNodes(max=max_size_of_graphs, min=min_size_of_graphs)
        node_pair_set_list, amount_of_nodes = temp_graph_obj.genSetListOfNodes(set_size_of_graphs)
        # For each pair we evaluate if we create an edge inbetween them
        for pair_nodes in node_pair_set_list:
            # Connect edges inbetween each node by selcting at random.
            create_edge_bool = temp_graph_obj.randomEdgeConnect(edge_prob)
            if create_edge_bool:
                temp_graph_obj.add_edge(pair_nodes[0],
                            pair_nodes[1],
                            weight=1)

        # Keep track of each graphs different probabilities components
        temp_graph_obj.component_proba_dict.get(edge_prob)[0] = nx.number_connected_components(temp_graph_obj)

        # Keep track of each graphs different average node degree
        temp_graph_obj.component_proba_dict.get(edge_prob)[1] = temp_graph_obj.calculateAvgComponentDegree()

        # Show visualization of graph
        if grph_indx % 100 == 0:
            temp_graph_obj.showGraph()

    list_of_graphs.append(temp_graph_obj)

# Show line graph of plotted p values and number of different component in the x-axis
# and to the y-axis respectively.
for graph in list_of_graphs:
    # Create graph for components for each edge pobability and tne number of componenets
    # as the probabaility of edges forming goes up starting from 0 to 1 in 0.1 increments.
    xx = [proab_graph_var for proab_graph_var in graph.component_proba_dict.keys()]
    yy = [num_componts[0] for num_componts in graph.component_proba_dict.values()]
    # Create 'numpy' array to build regression model.
    x = np.array(xx)
    y = np.array(yy)
    # Using 'scipy'and 'numpy' to build regression model type.
    X_Y_Spline = make_interp_spline(x, y)
    X_= np.linspace(x.min(), x.max(), 500)
    Y_ = X_Y_Spline(X_)
    # Set graph attributes
    ax = plt.axes()
    ax.set_title(graph.graph_name, c='black')
    ax.set_ylabel(f"Number of Components", c='black')
    ax.set_xlabel("Edge Probability (p={0.001-0.040})", c='black')
    ax.set_ylim(0, max_size_of_graphs)
    ax.set_xlim(0, 0.04)
    ax.set_facecolor('black')
    ax.tick_params(axis='x', colors='red')
    ax.tick_params(axis='y', colors='red')
    # Plot and show graphs
    plt.plot(X_, Y_, 'g')
    plt.plot(xx, yy, 'wo')
    plt.show(ax)

# Show line graph of plotted p values and number of different component
# to the x-axis and to the y-axis respectively.
for graph in list_of_graphs:
    graph.calculateAvgComponentDegree()
    # Create graph for components for each edge pobability.
    xxx = [proab_graph_var for proab_graph_var in graph.component_proba_dict.keys()]
    yyy = [num_components[1] for num_components in graph.component_proba_dict.values()]
    xxxx = np.array(xxx)
    yyyy = np.array(yyy)
    X_Y_Spline = make_interp_spline(xxxx, yyyy)
    XX_= np.linspace(xxxx.min(), yyyy.max(), 10000)
    YY_ = X_Y_Spline(XX_)
    ax = plt.axes()
    # # ax.text(1, 1, f"Edge Proabability: {edge_prob}", c='white')
    ax.set_title(graph.graph_name, c='blue')
    ax.set_ylabel(f"Avg. Node Degree", c='blue')
    ax.set_ylim(0, nx.number_of_nodes(graph))
    ax.set_xlabel("Edge Probability (p={0.001-0.020})", c='blue')
    ax.set_xlim(0, 0.09)
    ax.set_facecolor('black')
    ax.tick_params(axis='x', colors='red')
    ax.tick_params(axis='y', colors='red')
    plt.plot(XX_, YY_, 'g')
    plt.plot(xxx, yyy, 'c')
    plt.show(ax)
```