PROJECT REPORT

CS 579 – Online Social Network Analysis

Spring 2024

- PROJECT 2 – Final Report –

Analysis of Labor Market Dynamics Using the O*NET Dataset

Illinois Institute of Technology

By

Akshat Behera – A20516439

Rushikesh Kadam – A20517258

INDEX

1. Introduction
2. Data - Description
3. Project Process
4. Project Results
5. Discussion of Project
6. Future Work
7. References

1. Introduction

The objective of this project was to develop a comprehensive model of the labor market, focusing on the interrelationships between occupations and their required abilities, skills, and technology skills. By utilizing network analysis and predictive modeling techniques using data from the O*NET database, we aimed to uncover the underlying structure of the labor market, identify key skills and technologies, and assess the impact of technological advancements on various occupations. This analysis was intended to help stakeholders, including educators and policymakers, better understand and adapt to the rapidly changing demands of the workforce.

In pursuit of this objective, the project employed advanced community detection algorithms to delve deep into the structure of the labor market. By analyzing and visualizing the relationships between occupations and their requisite competencies, the project identified distinct communities within the labor market. These communities, or clusters of occupations that share similar skill sets and technological requirements, highlight the interconnected nature of modern job roles. Understanding these connections is essential for informing targeted educational programs and workforce training strategies designed to bridge potential skill gaps and enhance overall workforce preparedness amidst rapid technological change. We also simulated scenarios and modeled indirect relationships to understand how new technologies might influence various occupations.

Integrating both quantitative and qualitative data from the O*NET database, this project mapped out the complex network of occupational requirements and identified pivotal skills and technologies across various industries. The analysis revealed how different job roles are interconnected and demonstrated the transferability of certain skills beyond their traditional occupational categories. This in-depth understanding is designed to help stakeholders make informed decisions about educational programming, job training, and economic policy, aligning them with the evolving needs of the workforce. For example, insights from this project could guide educators in incorporating digital literacy into curricula across all fields of study to prepare students for the technologically driven job market.

Furthermore, the project utilized machine learning techniques, including logistic regression, SVM, random forest, and gradient boosting, to predict future trends and shifts in occupational skill requirements. This predictive analysis is crucial for proactive strategy development, enabling stakeholders to foresee and prepare for changes that affect job competencies. By integrating both quantitative and qualitative data from the O*NET database, the project not only illuminated how different job roles are interconnected but also emphasized the transferability and versatility of specific skills across various industries. By understanding these trends, stakeholders can better prepare for the future, ensuring that the workforce is equipped not only to meet current demands but also to adapt to future challenges.

Overall, the insights derived from this project are intended to equip policymakers, educators, and business leaders with the necessary tools to develop strategies that support continuous learning and skill adaptation, ensuring the workforce remains robust and adaptable in a rapidly evolving economic landscape.

2. Data Description

The O*NET dataset, provided by the U.S. Department of Labor, served as the data source for this project. It includes meticulously structured data detailing essential occupational requirements across three main dimensions—abilities, skills, and technology skills, encapsulated in three separate files: Abilities.xlsx, Skills.xlsx, and Technology_Skills.xlsx. Each file offers an exhaustive list of occupations with corresponding attributes such as the importance and level of each skill or ability required.

The datasets renowned for their standardization and reliability, were directly downloaded from the O*NET Website[8], which provides public access to up-to-date occupational data in Excel format. This method ensured that the data was not only reliable but also reflective of current labor market conditions, making it an ideal resource for our analytical objectives.

Given the extensive nature of the O*NET database, it was feasible to utilize the complete dataset without the need for sampling. The dataset includes three main files relevant to this project:

Abilities.xlsx (*90,792 instances and 15 columns*): This file contains data about the abilities required for various occupations. Abilities are defined as enduring attributes of the individual that influence performance. Examples include cognitive abilities like verbal comprehension and sensory abilities like night vision.

Skills.xlsx (*61,110 instances and 15 columns*): This file details the specific skills required across different occupations. Skills are seen as developed capacities that facilitate learning or the more rapid acquisition of knowledge.

Technology_Skills.xlsx (*32,435 instances and 7 columns*): This file enumerates the technologies and tools associated with each occupation.

Each entry in these files includes multiple dimensions of data, such as the importance and level of skills or abilities, formatted across various fields:

O*NET-SOC Code: The occupational code associated with each job role.

Title: The name of the occupation.

Element ID and Element Name: Identifiers and names for each skill or ability.

Scale ID and Scale Name: The type of measurement scale used (e.g., importance, level).

Data Value: The numerical value representing the degree to which a skill or ability is required.

N-Sample Size & Standard Error: Stats. Metrics that give insights into the reliability of the data.

Lower CI Bound and Upper CI Bound: The confidence interval bounds for the data value, indicating the range within which the true value is likely to lie.

Recommend Suppress and Not Relevant: Fields indicating whether data should be suppressed due to reliability issues or if it's not applicable to the occupation.

3. Project Process

The project process involved several detailed steps from data loading to complex network analysis and visualization. Below, we break down the significant parts of the code, highlighting the challenges faced and the solutions implemented at each stage.

a. Data Loading

Importing Neccessary Libraries

In [1]:
```
import pandas as pd
import networkx as nx
from networkx.algorithms import community
import matplotlib.pyplot as plt
from matplotlib import cm
import random
import community as community_louvain
import seaborn as sns


from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, roc_curve, auc
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
```

Import O*NET Datasets:

In [2]:
```
abilities   = pd.read_excel('Abilities.xlsx')
skills      = pd.read_excel('Skills.xlsx')
tech_skills = pd.read_excel('Technology_Skills.xlsx')
```

We began by importing the necessary libraries as shown and loading the datasets that we worked on (Abilities.xlsx; Skills.xlsx and Technology_Skills.xlsx) using Pandas, which is effective for handling and manipulating large datasets.

Initially, ensuring data integrity during loading was crucial, as discrepancies could lead to incorrect analyses later.

By using pandas, we can handle various data inconsistencies such as missing values or format discrepancies seamlessly. So, we converted our .xlsx files into dataframes namely “abilities”, “skills” and “tech_skills” will serve as the foundation for further analysis and processing within the project.

b. Data Preprocessing / Feature Engineering

```
: # Lowercase and strip whitespace for string columns in each DataFrame
for df in [abilities, skills, tech_skills]:
    for col in df.columns:
        if df[col].dtype == 'object':
            df[col] = df[col].str.lower().str.strip()
```

To maintain consistency across dataframes, we standardized text entries and filled missing values.

```
# Preprocessing - normalize text and handle missing values
def preprocess(df):
    # Normalize text data to lower case
    text_columns = df.select_dtypes(include=['object']).columns
    df[text_columns] = df[text_columns].apply(lambda x: x.str.lower().str.strip())

    # Handle missing values if any - This will depend on your specific use-case
    # For this demonstration, we'll fill missing values with a placeholder
    df.fillna('missing', inplace=True)

    return df

abilities = preprocess(abilities)
skills = preprocess(skills)
tech_skills = preprocess(tech_skills)
```

The data from O*NET contained textual information that varied in formatting due to the nature of how it was collected and recorded. This variability could lead to discrepancies in data analysis, such as the same skill or technology being recognized as different due to case differences or extra spaces

Additionally, handling missing values is a common issue in data preprocessing, as incomplete data can skew the analysis or lead to erroneous conclusions.

For this, we then utilized pandas library functions like str.lower() and str.strip() to adjust text data across all object type columns in the dataframes and applied the fillna('missing') method to all dataframes, ensuring no data point was left blank.

- ***Feature Engineering and Data Aggregation (For Machine Learning Models)***

Here, we transform raw data into a format suitable for machine learning analysis, specifically focusing on identifying occupations that are experiencing growth due to technological advancements.

```
# We are taking column 'Hot Technology' being 'Y' is indicative of a growing occupation
# Encoding this as a binary feature

tech_skills_df['Hot_Tech'] = tech_skills_df['Hot Technology'].apply(lambda x: 1 if x == 'Y' else 0)

# Aggregating tech skills data at the occupation level to find the sum of 'Hot Tech' skills
tech_skills_aggregated = tech_skills_df.groupby('O*NET-SOC Code')['Hot_Tech'].sum().reset_index()

# For skills and abilities, Creating a combined importance and level value
# It's simple heuristic.

skills_df['Combined_Value'] = skills_df['Data Value'] * (skills_df['Scale ID'] == 'IM')
abilities_df['Combined_Value'] = abilities_df['Data Value'] * (abilities_df['Scale ID'] == 'IM')

skills_pivot = skills_df.pivot_table(index='O*NET-SOC Code', columns='Element Name', values='Combined_Value', aggfunc='mean')
abilities_pivot = abilities_df.pivot_table(index='O*NET-SOC Code', columns='Element Name', values='Combined_Value', aggfunc='
                                           mean')

# Combining all features into one DataFrame, using occupation code as an index
features_combined = skills_pivot.join(abilities_pivot, how='outer').join(tech_skills_aggregated.set_index('O*NET-SOC Code'),
                                                                         on='O*NET-SOC Code')

# Handling missing values
features_combined.fillna(0, inplace=True)
```

```
# Defining a simple threshold for an occupation to be considered in growth
growth_threshold = tech_skills_aggregated['Hot_Tech'].median()

# Binary target variable based on the threshold
features_combined['Target_Growth'] = (features_combined['Hot_Tech'] >= growth_threshold).astype(int)
```

One of the challenges was to accurately represent the significance of 'Hot Technology' in predicting occupational growth, as not all technologies are equally impactful.

Another challenge was Aggregating technology skills data by occupation to reflect the total technological intensity required in each occupation needed careful handling to avoid skewing the data.

Also, combining multiple datasets with varying structures and handling missing values required a robust approach to maintain data integrity.

First for Binary Encoding we converted the 'Hot Technology' column into a binary feature, where 'Y' (Yes) was encoded as 1 and everything else as 0. This simplification allowed us to use this feature as a direct indicator of technology-driven growth in occupations.

For the next challenge we used pivot tables to aggregate and restructure the skills and abilities data according to the importance levels associated with each occupation. This method helped maintain clarity and consistency in how the data was processed and interpreted.

Finally, by joining the skill and ability pivot tables with the aggregated technology skills data and filling missing values with zeros, we ensured that the final dataset was complete and ready for analysis. This step was crucial for the accurate application of machine learning models.

c. Network Construction – Building the Network

This code segment constructs a bipartite graph representing the relationships between occupations, skills, and technologies. Occupations, skills, and technologies are represented as nodes in the graph, with edges indicating associations between them. The importance of skills and abilities is considered by adding weighted edges based on the 'Data Value' column. Additionally, a 'hot' attribute is included for technology nodes based on the 'Hot Technology' column in the data.

```
# Creating a bipartite graph
G = nx.Graph()

# Adding occupation nodes
occupations = set(abilities['O*NET-SOC Code'].unique()) \
              .union(skills['O*NET-SOC Code'].unique()) \
              .union(tech_skills['O*NET-SOC Code'].unique())
G.add_nodes_from(occupations, bipartite=0, type='occupation')

# Adding skills nodes
skills_abilities = set(skills['Element Name'].unique()).union(abilities['Element Name'].unique())
G.add_nodes_from(skills_abilities, bipartite=1, type='skill')

# Adding technology nodes
technologies = set(tech_skills['Example'].unique())
G.add_nodes_from(technologies, bipartite=1, type='technology')

# Adding edges for abilities and skills
# We are adding importance as weight only for simplification.
for _, row in abilities.iterrows():
    if row['Scale ID'] == 'IM':
        G.add_edge(row['O*NET-SOC Code'], row['Element Name'], weight=row['Data Value'])

for _, row in skills.iterrows():
    if row['Scale ID'] == 'IM':
        G.add_edge(row['O*NET-SOC Code'], row['Element Name'], weight=row['Data Value'])

# Adding edges for technologies with 'hot' attribute
for _, row in tech_skills.iterrows():
    hot_tech = row['Hot Technology'] == 'Y'
    G.add_edge(row['O*NET-SOC Code'], row['Example'], hot_tech=hot_tech)
```

Combining data from three different sources (abilities, skills, and technology skills) into a coherent graph structure posed significant challenges due to differing data schemas and content.

The construction of a graph with potentially thousands of nodes and edges required efficient data handling to prevent performance bottlenecks. Accurately representing the relationship between occupations and skills/technologies, particularly ensuring that the weights and types of relationships were meaningful, was critical.

To handle the integration of diverse datasets, node sets for occupations, skills, and technologies were created by extracting unique entries from each dataset and treating them as distinct categories within the graph (bipartite structure). This clear distinction helped in managing data from different sources effectively.

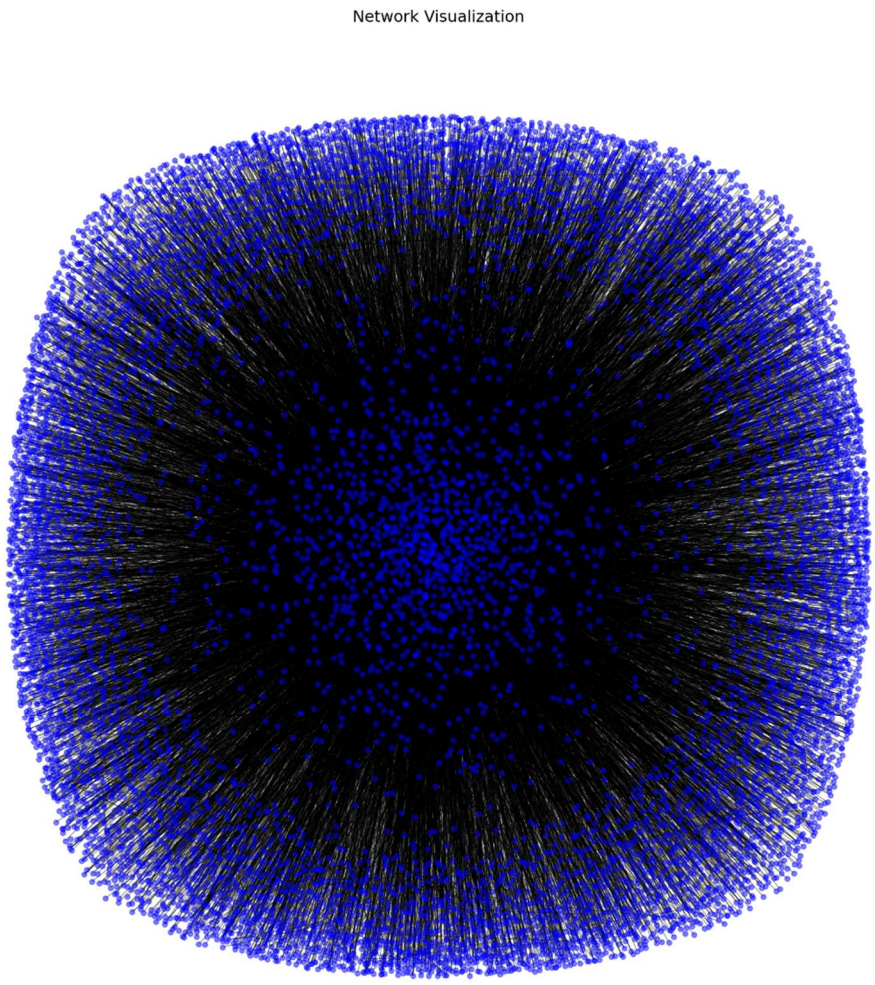
NetworkX was used to create and manipulate the bipartite graph due to its robust handling of large and complex network data. This tool efficiently manages large datasets, ensuring that the graph structure remains manageable and analyzable.

Weights were assigned to edges based on the 'Data Value' where applicable, especially for skills and abilities marked as 'important' (Scale ID = 'IM'). For technology skills, a binary attribute ('hot_tech') indicated whether the technology was considered a hot technology, thereby influencing its relevance to the occupation.

The creation of this bipartite graph facilitated a structured analysis of how occupations are connected to specific skills and technologies, highlighting the dependencies and importance of various attributes within the labor market

- ***Visualizations***

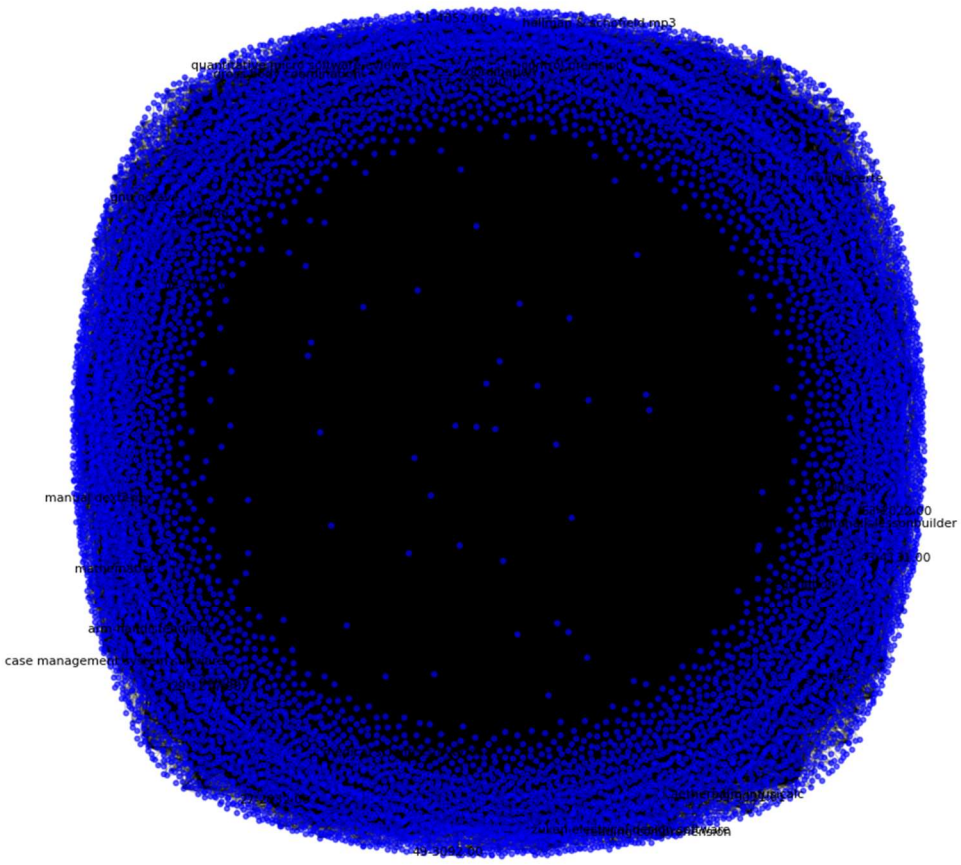
```
#Visualization of the network:

# Increasing figure size
plt.figure(figsize=(20, 20))

# Visualizing the bipartite network with a layout suitable for large graphs
pos = nx.spring_layout(G, k=0.15, iterations=20)

# Drawing the graph without labels for clarity
nx.draw(G, pos, node_size=50, node_color='blue', alpha=0.5)

# Drawing labels for a subset of nodes (e.g., only for 'hot' tech skills)
hot_tech_skills = [node for node, data in G.nodes(data=True) if data.get('hot_tech')]
nx.draw_networkx_labels(G, pos, labels={node: node for node in hot_tech_skills}, font_size=8, font_color='red')

plt.title('Network Visualization', fontsize=22)
plt.show()
```



The output is a visual representation of a bipartite graph showing connections between occupations and 'hot' technology skills. The edges indicate the importance of specific skills/abilities for different occupations and the utilization of 'hot' technologies within them. The layout is organized for clarity, providing an overview of the relationships between occupations, skills, and technologies.

Visualizing general network with all occupations, skills and technologies.

```
# Defining figure size
plt.figure(figsize=(12, 12))

# Using a simple layout and scale it up to space out nodes
pos = nx.spring_layout(G, k=0.75, iterations=20)

# Drawing the network
nx.draw_networkx_nodes(G, pos, node_size=10, node_color='blue', alpha=0.6)
nx.draw_networkx_edges(G, pos, alpha=0.2)

# Converting sets to lists
occupations_list = list(occupations)
skills_abilities_list = list(skills_abilities)
technologies_list = list(technologies)

# Combining lists
subset_of_nodes = random.sample(occupations_list, 10) + random.sample(skills_abilities_list, 10) + random.sample(technologies_lis


labels = {node: node for node in subset_of_nodes}
nx.draw_networkx_labels(G, pos, labels, font_size=8, font_color='black')

# Removing the axis
plt.axis('off')

# Show plot
plt.show()
```



The above output shows a subset of nodes from the bipartite graph representing relationships between occupations, skills, and technologies. Employing the spring layout algorithm, nodes are positioned to maximize clarity, with edges drawn faintly to avoid clutter. Blue nodes denote occupations, while labels highlight a diverse selection of occupations, skills, and technologies.

By converting sets to lists and randomly sampling a subset from each category, the visualization offers a glimpse into the interconnectedness of these elements.

The below step involved creating separate bipartite graphs for skills, abilities, and technology requirements associated with occupations.

```
# Function to create a bipartite graph from data
def create_bipartite_graph(df, node_set_1_name, node_set_2_name, attr_name=None):
    B = nx.Graph()

    # Extracting unique identifiers for node sets
    node_set_1 = df[node_set_1_name].unique()
    node_set_2 = df[node_set_2_name].unique()

    # Adding nodes with the bipartite attribute
    B.add_nodes_from(node_set_1, bipartite=0)
    B.add_nodes_from(node_set_2, bipartite=1)

    # Adding edges with optional attribute
    for _, row in df.iterrows():
        if attr_name and attr_name in row:
            B.add_edge(row[node_set_1_name], row[node_set_2_name], **{attr_name: row[attr_name]})
        else:
            B.add_edge(row[node_set_1_name], row[node_set_2_name])

    return B
```

```
# Create different bipartite graphs
G_skills = create_bipartite_graph(skills, 'O*NET-SOC Code', 'Element Name', 'Data Value')
G_abilities = create_bipartite_graph(abilities, 'O*NET-SOC Code', 'Element Name', 'Data Value')
G_tech = create_bipartite_graph(tech_skills, 'O*NET-SOC Code', 'Example', 'Hot Technology')
```

The above, code is a function, create_bipartite_graph, to generate bipartite graphs from input data. Using this function, three bipartite graphs are created: G_skills from 'skills', G_abilities from 'abilities', and G_tech from 'tech_skills'. These graphs represent relationships between different sets of nodes, such as occupation codes and skill names, with optional edge attributes like 'Data Value' and 'Hot Technology'.

Each type of data (skills, abilities, technology) required its own specific handling to ensure that the graphs accurately reflected the relationships and were useful for analysis.

Also, iterating over large datasets to create graphs can be computationally expensive, especially when dealing with multiple large data frames and complex graph structures.

Even correctly incorporating attributes like 'Data Value' and 'Hot Technology' as edge properties in the graph required careful coding to ensure that attributes were correctly applied and interpreted.

By developing a function that could be applied to any dataframe with the appropriate parameters, we were able to streamline the creation of multiple bipartite graphs, reducing redundancy in the code and ensuring consistency across different datasets.

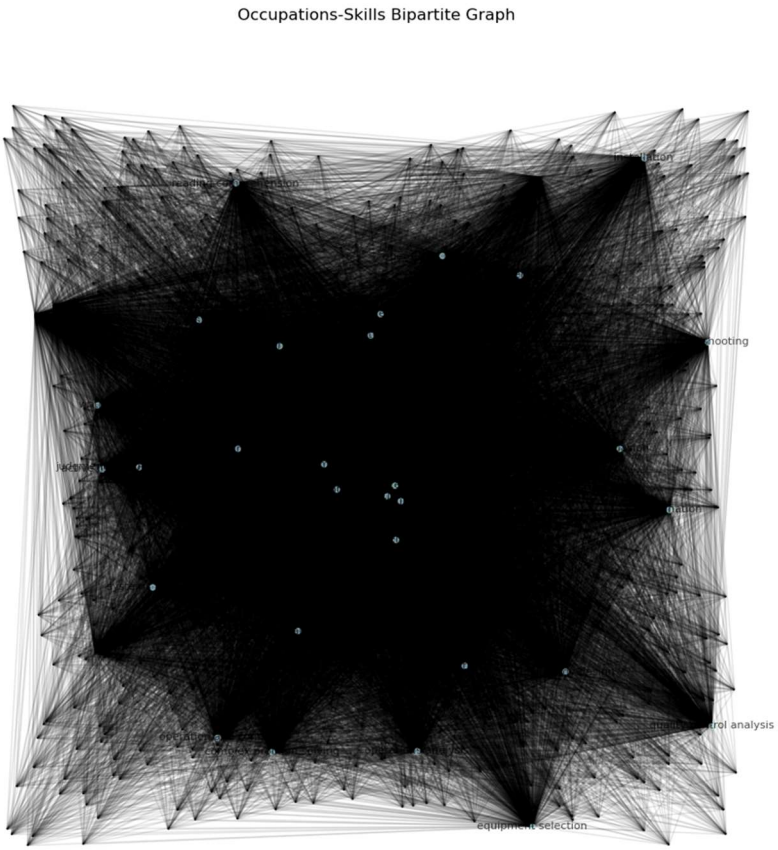
The function uses NetworkX efficiently by adding nodes and edges in a structured manner, minimizing overhead and improving performance.

Also, the function was designed in a way to optionally include any additional attributes as edge properties if they were present, making the graph creation flexible and adaptable to different kinds of input data.

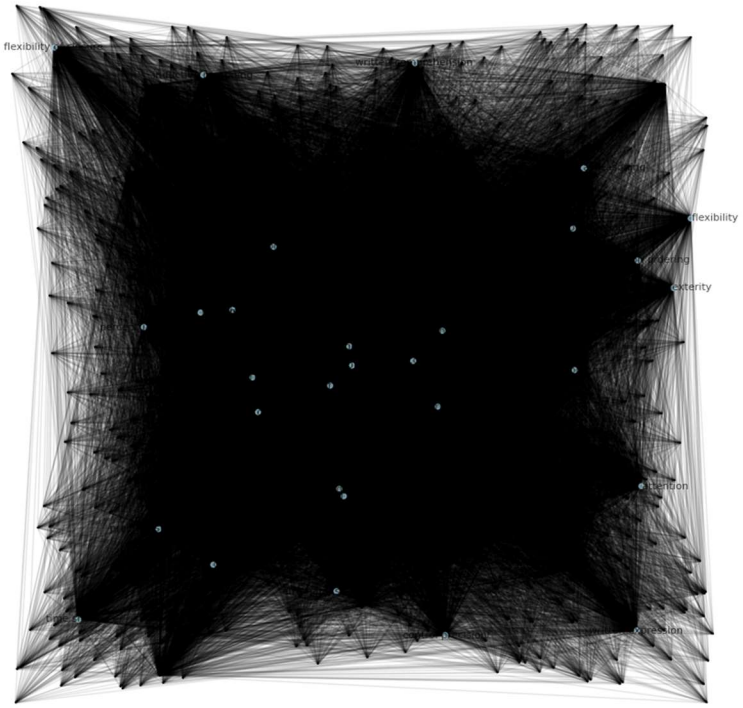
- ***Visualizations***

Representing complex network structures in a clear and understandable manner can be challenging, especially when dealing with large graphs that contain many nodes and edges.

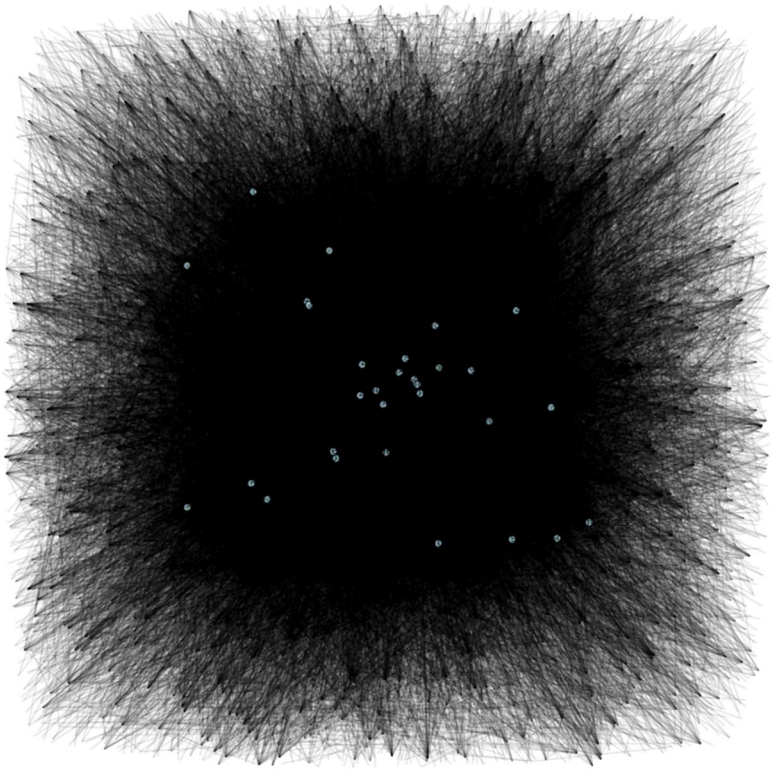
So, using spring_layout with appropriate parameters helps in spreading out the nodes reasonably, making the visualization clearer and more informative.



Occupations-Abilities Bipartite Graph



Occupations-Technology Skills Bipartite Graph



d. Community Detection

Using community detection algorithms helped us identify clusters within our network, which could represent closely related occupations.

In this section we are defining a function, detect_communities, to identify communities within a bipartite graph. It first projects the graph onto the occupation nodes and then applies community detection using the greedy modularity optimization method. Communities are detected separately for each graph: G_skills, G_abilities, and G_tech. The identified communities are printed, displaying the first five occupations in each community.

```
def detect_communities(B):
    # Project bipartite graph to a unipartite graph on occupations
    occupations = {n for n, d in B.nodes(data=True) if d['bipartite'] == 0}
    projected_B = nx.bipartite.weighted_projected_graph(B, occupations)

    # Detecting communities
    return list(community.greedy_modularity_communities(projected_B))

# Detecting communities in each graph
communities_skills = detect_communities(G_skills)
communities_abilities = detect_communities(G_abilities)
communities_tech = detect_communities(G_tech)
```

Community detection in large networks can be both computationally expensive and complex to interpret.

We utilized the greedy modularity community detection algorithm from NetworkX, which offers a good balance between computational efficiency and the quality of results.

As it's a group-based community detection algorithm, it was chosen for its efficiency and effectiveness in detecting communities within large networks. This method approximates the maximization of modularity, effectively revealing densely connected groups of nodes.

```
#Mapping each occupation code to its corresponding title.
soc_code_to_title = pd.concat([abilities[['O*NET-SOC Code', 'Title']], skills[['O*NET-SOC Code', 'Title']]]).drop_duplicates().s
```

```
def print_communities_with_titles(communities, soc_code_to_title):
    for i, community in enumerate(communities):
        print(f"Community {i+1}: {list(community)[:5]}")
        titles = [soc_code_to_title.get(code, "Unknown title") for code in community]
        print(f"Community {i+1}: {list(titles)[:5]}")
        #print(', '.join(titles))
        print("")

# Printing communities for each graph
print("Skills Communities:")
print_communities_with_titles(communities_skills, soc_code_to_title)
print("")

print("Abilities Communities:")
print_communities_with_titles(communities_abilities, soc_code_to_title)
print("")

print("Technology Skills Communities:")
print_communities_with_titles(communities_tech, soc_code_to_title)
print("")
```

We get top 5 communities withing every dataset labeled with title along with the soc code for better understanding.

The function print_communities_with_titles(communities, soc_code_to_title) iterates through each detected community. For each community, it prints the top 5 nodes (occupations) based on their titles instead of codes. It retrieves the corresponding titles from the soc_code_to_title dictionary using the occupation codes within the community. Finally, it prints each community along with its corresponding occupation titles.

Output:

```
Skills Communities:
Community 1: ['47-2021.00', '19-1029.04', '39-1022.00', '17-2141.00', '47-5071.00']
Community 1: ['brickmasons and blockmasons', 'biologists', 'first-line supervisors of personal service workers', 'mechanical en
gineers', 'roustabouts, oil and gas']


Abilities Communities:
Community 1: ['47-2021.00', '19-1029.04', '39-1022.00', '17-2141.00', '47-5071.00']
Community 1: ['brickmasons and blockmasons', 'biologists', 'first-line supervisors of personal service workers', 'mechanical en
gineers', 'roustabouts, oil and gas']


Technology Skills Communities:
Community 1: ['39-1022.00', '19-1029.04', '17-2141.00', '41-9091.00', '29-2057.00']
Community 1: ['first-line supervisors of personal service workers', 'biologists', 'mechanical engineers', 'door-to-door sales w
orkers, news and street vendors, and related workers', 'ophthalmic medical technicians']

Community 2: ['47-2021.00', '53-5011.00', '15-1252.00', '47-5071.00', '15-2051.01']
Community 2: ['brickmasons and blockmasons', 'sailors and marine oilers', 'Unknown title', 'roustabouts, oil and gas', 'busines
s intelligence analysts']

Community 3: ['49-9063.00']
Community 3: ['musical instrument repairers and tuners']

Community 4: ['47-5043.00']
Community 4: ['roof bolters, mining']
```

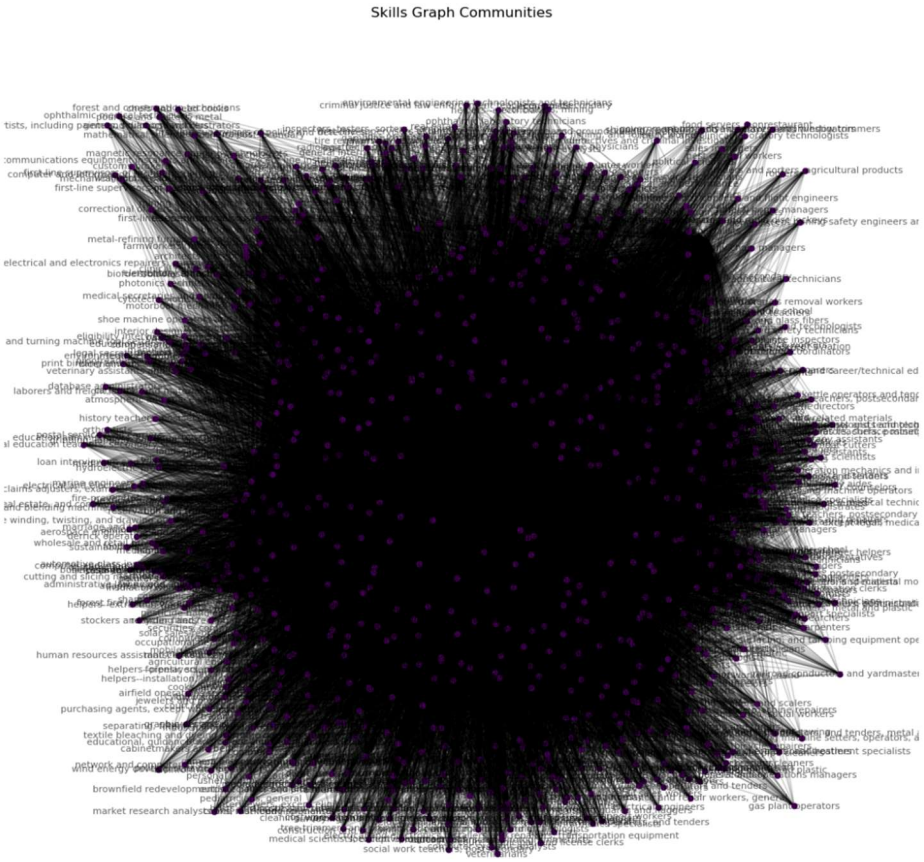
- ***Visualizations*:**

Visualization of every community with labels as titles instead of soc codes for better understanding

```
def visualize_communities(B, communities, title, soc_code_to_title):
    plt.figure(figsize=(14, 14))
    # Creating a color map for the communities
    color_map = cm.get_cmap('viridis', len(communities))
    # Creating a position layout for our graph
    pos = nx.spring_layout(B, k=0.1, iterations=5)

    # Draw the nodes and color them based on which community they belong to
    for i, community in enumerate(communities):
        # Geting a list of titles for the nodes in this community
        community_titles = {node: soc_code_to_title[node] for node in community if node in soc_code_to_title}
        nx.draw_networkx_nodes(B, pos, community_titles.keys(), node_size=20, node_color=[color_map(i)])

    # Drawing the edges of the graph
    nx.draw_networkx_edges(B, pos, alpha=0.2)  # Lower alpha to make edges less dominant

    # Drawing inglabels for each node
    labels = {node: soc_code_to_title[node] for node in B.nodes() if node in soc_code_to_title}
    nx.draw_networkx_labels(B, pos, labels, font_size=8, alpha=0.6)

    plt.title(title)
    plt.axis('off')
    plt.show()


# Update the communities if necessary to include only SOC codes
communities_skills = [set(soc_code_to_title.keys()).intersection(community) for community in communities_skills]
communities_abilities = [set(soc_code_to_title.keys()).intersection(community) for community in communities_abilities]
communities_tech = [set(soc_code_to_title.keys()).intersection(community) for community in communities_tech]
```

```
#Visualization of communities in Skills graph
visualize_communities(G_skills, communities_skills, "Skills Graph Communities", soc_code_to_title)
```

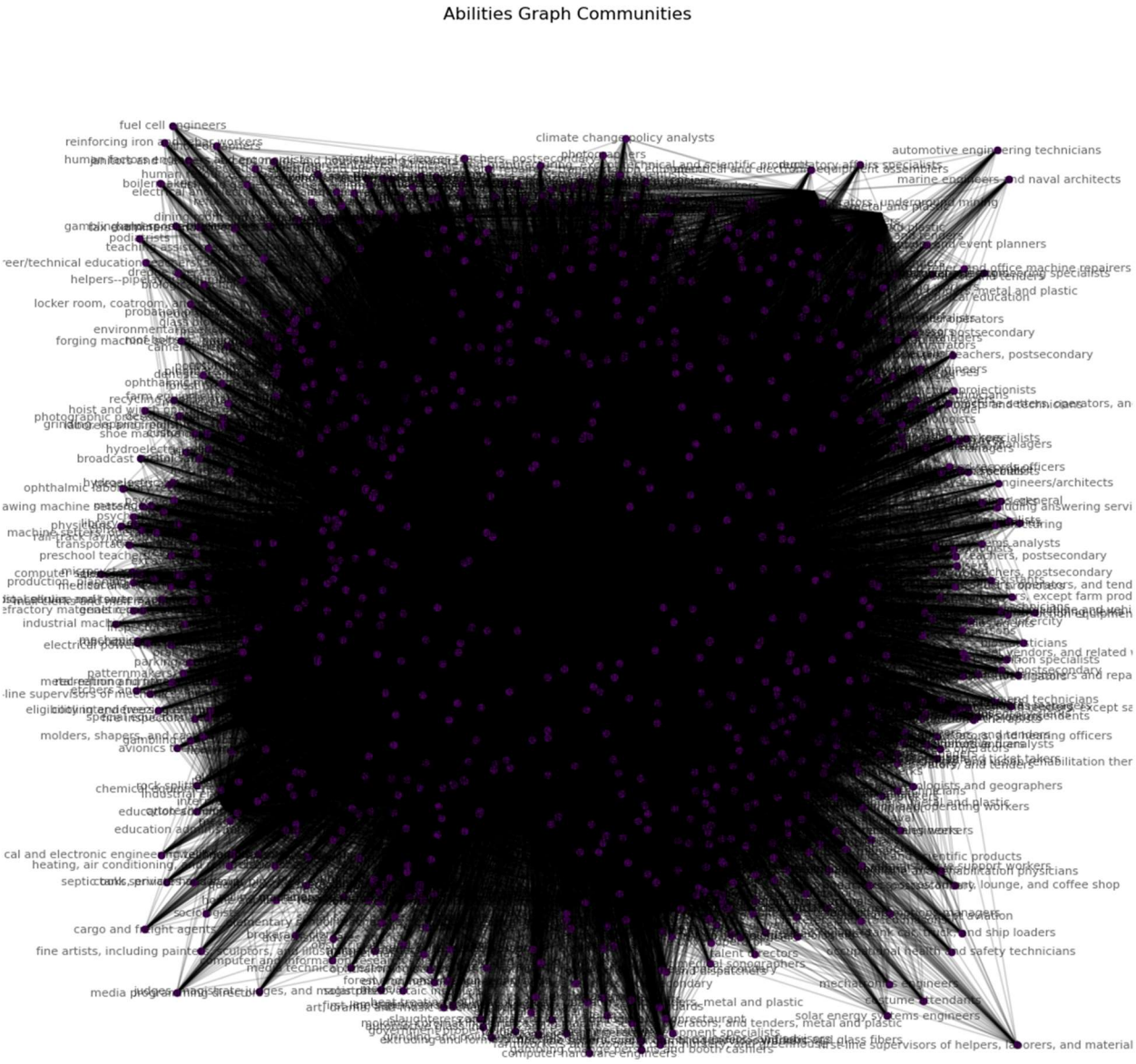
The Skill graph below shows the interconnectivity of various occupations based on shared skills. Structured as a dense, circular network, each node represents an occupation, and the connecting edges highlight shared or similar skills. Highlighted occupations such as "forest and conservation technicians," "medical scientists," and "social work teachers" show the range of specialized roles within this skills-based network.



The visualization shown below depicts the interconnectedness of occupations based on shared abilities. This graph is structured as a large, dense network where each node represents an occupation, and the edges between them indicate shared or similar abilities. The nodes are scattered around the edge of the circular layout, which helps in identifying the clusters or communities of occupations that have closely related abilities**.**

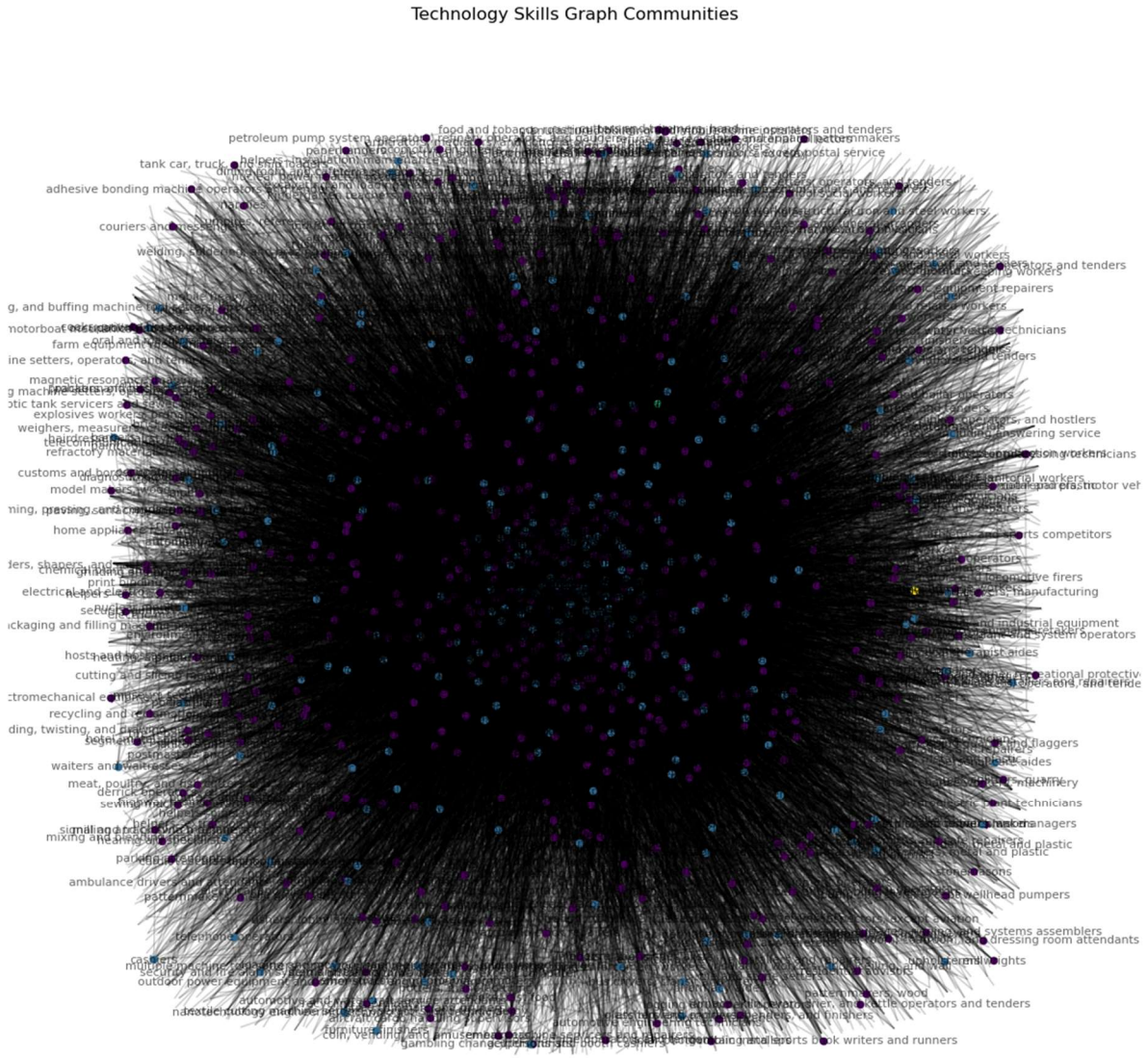
Notably, the graph is quite dense with many overlapping edges, suggesting a complex set of relationships where many occupations share abilities with multiple other occupations.

Specific occupation labels such as "climate change policy analysts," "automotive engineering technicians," and "marine engineers and naval architects" are visible, providing examples of specialized roles within the network.



Similarly to the previous visualization on abilities, the graph shown below represents the interconnectedness of occupations based on shared technology skills. This graph, like the previous one, is structured as a large, dense network where each node represents an occupation, and the edges between them indicate shared or similar technology skills.

Specific occupation labels such as "petroleum pump system operators," "food and tobacco roasting, baking, and drying machine operators," and "lathe and turning machine tool setters, operators, and tenders" are visible.



Analyzing Communities

This function provides insights into the detected communities within a graph. It shows the total number of communities detected and calculates the overall modularity of the partition. Additionally, it identifies the top central occupations within each community based on degree centrality and then this funtion is applied to analyze the communities detected in the graph 'occupation_projected'.

```python
# 'communities' is the result from community detection

def analyze_communities(G, communities):
    print(f"Total communities detected: {len(communities)}")
    modularity = community.modularity(G, communities)
    print(f"Overall modularity of the partition: {modularity:.4f}")

    for i, comm in enumerate(communities):
        subgraph = G.subgraph(comm)
        density = nx.density(subgraph)
        print(f"Community {i+1}: {len(comm)} nodes, Density: {density:.4f}")

        # Calculating degree centrality within the community
        centrality = nx.degree_centrality(subgraph)
        top_central_nodes = sorted(centrality.items(), key=lambda item: item[1], reverse=True)[:5]
        print(f"Top central occupations in Community {i+1}: {top_central_nodes}")

analyze_communities(occupation_projected, communities)
```

```
Total communities detected: 4
Overall modularity of the partition: 0.0325
Community 1: 626 nodes, Density: 0.9752
Top central occupations in Community 1: [('19-4012.00', 0.9968), ('21-1015.00', 0.9968), ('19-1023.00', 0.9968), ('11-9179.01',
0.9968), ('11-9013.00', 0.9968)]
Community 2: 295 nodes, Density: 0.9008
Top central occupations in Community 2: [('15-1232.00', 0.9795918367346939), ('43-9061.00', 0.9727891156462585), ('11-9111.00',
0.9727891156462585), ('15-1211.00', 0.9727891156462585), ('29-9021.00', 0.9693877551020408)]
Community 3: 1 nodes, Density: 0.0000
Top central occupations in Community 3: [('49-9063.00', 1)]
Community 4: 1 nodes, Density: 0.0000
Top central occupations in Community 4: [('47-5043.00', 1)]
```

e. Identifying Influencers

This part of the code is focused on identifying the top influencers within the skills, abilities, and technology skills networks. Influencers in this context refer to the nodes that have the highest degree centrality, indicating they are most connected within the network, and therefore potentially the most impactful or required across multiple occupations.

```python
def identify_influencers(B):
    # We will project onto the non-occupation nodes to find influential skills/abilities/technologies
    non_occupations = {n for n, d in B.nodes(data=True) if d['bipartite'] == 1}
    projected_B = nx.bipartite.weighted_projected_graph(B, non_occupations)

    # Calculating degree centrality
    centrality = nx.degree_centrality(projected_B)
    return sorted(centrality.items(), key=lambda x: x[1], reverse=True)[:10]

#Top 10 influencers in each graph
influencers_skills = identify_influencers(G_skills)
influencers_abilities = identify_influencers(G_abilities)
influencers_tech = identify_influencers(G_tech)
```

The function first identifies non-occupation nodes in the graph, which likely represent skills, abilities, or technologies. It then projects the bipartite graph onto these non-occupation nodes to create a projected graph.

Using the projected graph, it calculates the degree centrality for each node, representing the proportion of other nodes it is connected to. Finally, it sorts the centrality values in descending order and returns the top 10 influencers.

The code then applies this function to three different graphs (G_skills, G_abilities, and G_tech) to identify the top 10 influencers in each graph based on their degree centrality. These influencers represent skills, abilities, or technologies that are highly connected within their respective networks, indicating their significance or importance.

G_skills representing the relationships between occupations and their required skills.

G_abilities representing the relationships between occupations and required abilities.

G_tech representing the relationships between occupations and technology skills

Output:

```
: #Top 10 influecing skills within a graph
  influencers_skills

: [('equipment maintenance', 1.0),
   ('active listening', 1.0),
   ('negotiation', 1.0),
   ('judgment and decision making', 1.0),
   ('systems analysis', 1.0),
   ('equipment selection', 1.0),
   ('speaking', 1.0),
   ('troubleshooting', 1.0),
   ('complex problem solving', 1.0),
   ('science', 1.0)]

: #Top 10 influecing abilities within a graph
  influencers_abilities

: [('written comprehension', 1.0),
   ('control precision', 1.0),
   ('selective attention', 1.0),
   ('stamina', 1.0),
   ('trunk strength', 1.0),
   ('night vision', 1.0),
   ('oral comprehension', 1.0),
   ('problem sensitivity', 1.0),
   ('deductive reasoning', 1.0),
   ('fluency of ideas', 1.0)]

: #Top 10 influecing tech skills within a graph
  influencers_tech

: [('microsoft excel', 0.9811299176578225),
   ('microsoft office software', 0.9717520585544374),
   ('microsoft word', 0.9356129917657823),
   ('microsoft powerpoint', 0.9006175663311986),
   ('microsoft outlook', 0.774245196706313),
   ('microsoft access', 0.7291857273559013),
   ('web browser software', 0.5853156450137237),
   ('microsoft project', 0.5433440073193047),
   ('sap software', 0.5379688929551693),
   ('microsoft visio', 0.4539112534309241)]
```

Projecting the bipartite graph onto non-occupation nodes (skills, abilities, technologies) to analyze their centrality within the network presents computational and structural challenges.

So we used NetworkX to project the graph onto non-occupation nodes effectively, focusing on skills, abilities, or technologies only. This projection highlighted the interconnectedness and influence of these nodes across occupations.

- **Visualizations:**

Below is the code where we identify and visualize the most influential skills, abilities, and technologies within a network of occupations by projecting a bipartite graph onto non-occupation nodes and calculating degree centrality.

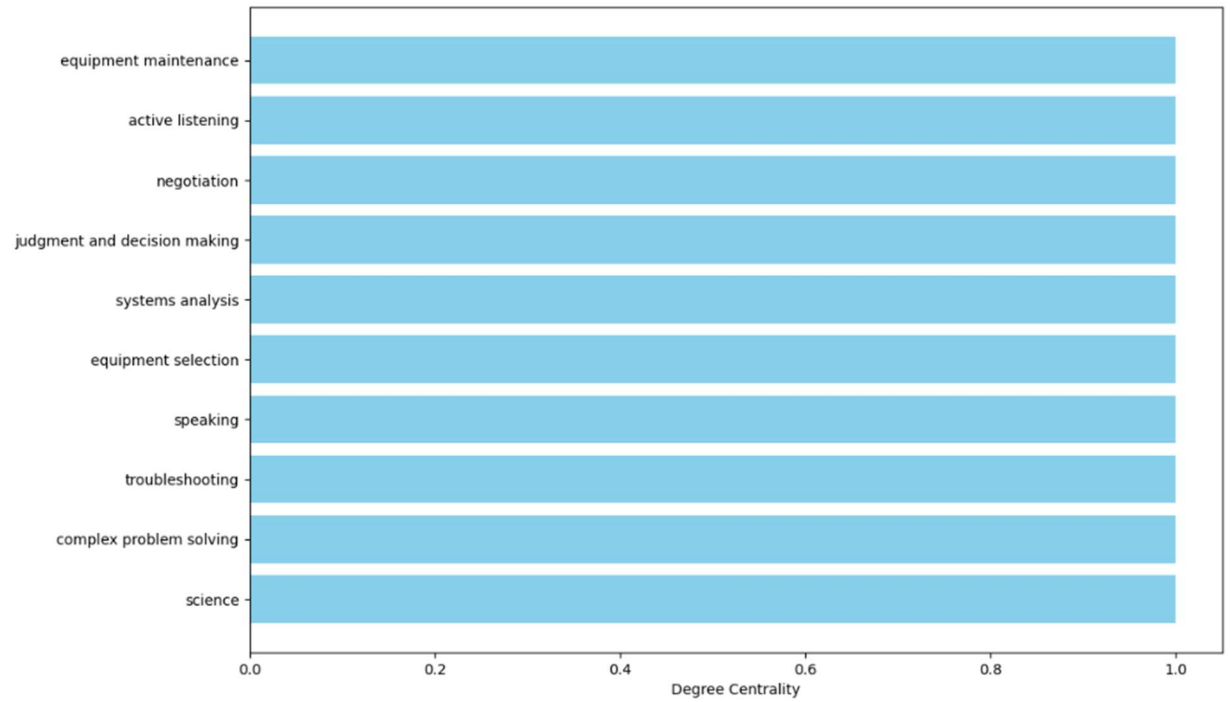
It focuses on extracting the top influencers based on their connectivity within the graph, sorting and mapping these to more readable labels and then visualizing the results in a bar chart.

Visualizing top influencing skills, abilities, tech skills based on degree cetrality

```
def identify_influencers2(B, top_n=10, label_mapping=None):
    try:
        # We will project onto the non-occupation nodes to find influential skills/abilities/technologies
        non_occupations = {n for n, d in B.nodes(data=True) if d['bipartite'] == 1}
        projected_B = nx.bipartite.weighted_projected_graph(B, non_occupations)

        # Calculating degree centrality
        centrality = nx.degree_centrality(projected_B)

        # Sorting and getting the top N influencers
        top_influencers = sorted(centrality.items(), key=lambda x: x[1], reverse=True)[:top_n]

        # Maping SOC codes to titles if a mapping is provided
        if label_mapping:
            top_influencers = [(label_mapping.get(node, node), centrality) for node, centrality in top_influencers]

        # Visualizing the top N influencers
        visualize_top_influencers(top_influencers)

        return top_influencers
    except KeyError as e:
        print(f"Error: Missing node attribute {e}")

def visualize_top_influencers(top_influencers):
    nodes, centralities = zip(*top_influencers)
    plt.figure(figsize=(12, 8))
    plt.barh(range(len(nodes)), centralities, color='skyblue')
    plt.xlabel('Degree Centrality')
    plt.yticks(range(len(nodes)), nodes)
    plt.gca().invert_yaxis()  # Highest values at the top
    plt.show()
```

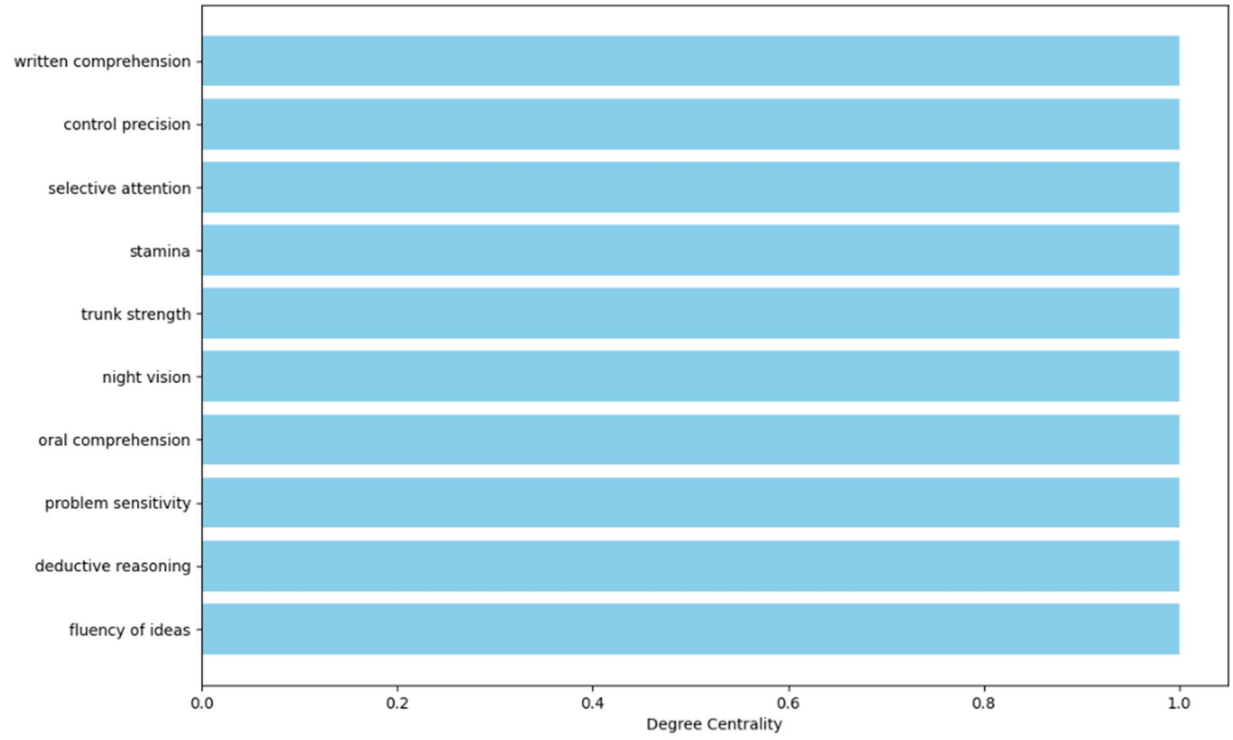
Influencers for G_Skills:

```
influencers_skills = identify_influencers2(G_skills, top_n=10, label_mapping=soc_code_to_title)
```



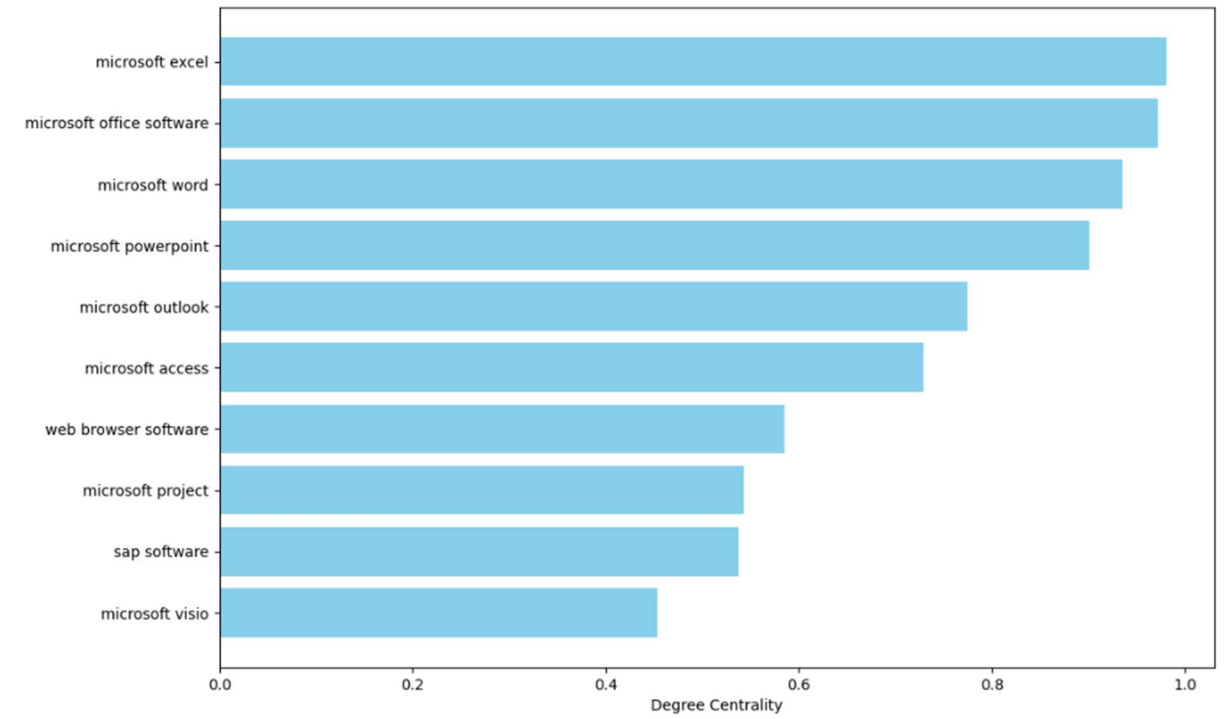
Influencers for G_Abilities:

```
influencers_abilities = identify_influencers2(G_abilities, top_n=10, label_mapping=soc_code_to_title)
```



Influencers for G_TechSkills:

```
influencers_tech = identify_influencers2(G_tech, top_n=10, label_mapping=soc_code_to_title)
```



f. Simulate Scenarios

We used 3 applicable network models to simulate different scenarios, such as:

- Random Model

This random graph model function generates a random graph using the Erdős-Rényi model with parameters n and p and returns the resulting graph.

```
def random_graph_model(n, p):
    """Generate a random graph using the Erdős-Rényi model."""
    G_random = nx.erdos_renyi_graph(n, p, seed=20)
    return G_random
```

Below code generates a bipartite graph from a given base graph by assigning nodes to two groups. It returns the resulting bipartite graph.

```python
def create_bipartite_graph(base_graph):
    """Generate a bipartite graph from a random graph by assigning nodes to two groups."""
    B = nx.Graph()
    top_nodes = {n for n, d in base_graph.nodes(data=True) if n % 2 == 0}
    bottom_nodes = set(base_graph) - top_nodes

    # Adding nodes with the bipartite attribute
    B.add_nodes_from(top_nodes, bipartite=0)
    B.add_nodes_from(bottom_nodes, bipartite=1)

    # Adding edges from the base graph
    for u, v in base_graph.edges():
        if u in top_nodes and v in bottom_nodes:
            B.add_edge(u, v)
        elif v in top_nodes and u in bottom_nodes:
            B.add_edge(v, u)

    return B
```

This simulate_Rand function adds a new technology node to the bipartite graph B and connects it with a random sample of nodes from one of the bipartite groups. It prints the edges added between occupations and the new technology node. Then, it visualizes the subgraph around the new technology.

```python
def simulate_Rand(B, new_tech_name, soc_code_to_title):
    """Adding a new technology node and connect it with a random sample of nodes from one bipartite group."""
    B.add_node(new_tech_name, bipartite=1, type='technology')
    occupations = [n for n, d in B.nodes(data=True) if d['bipartite'] == 0]
    sample_occupations = random.sample(occupations, 10)

    for occ in sample_occupations:
        B.add_edge(occ, new_tech_name)
        print(f"Added edge between {soc_code_to_title.get(occ, occ)} and {new_tech_name}")

    visualize_technology_subgraph(B, new_tech_name)
```

- Visualizations

Visualization of the subgraph for each node:

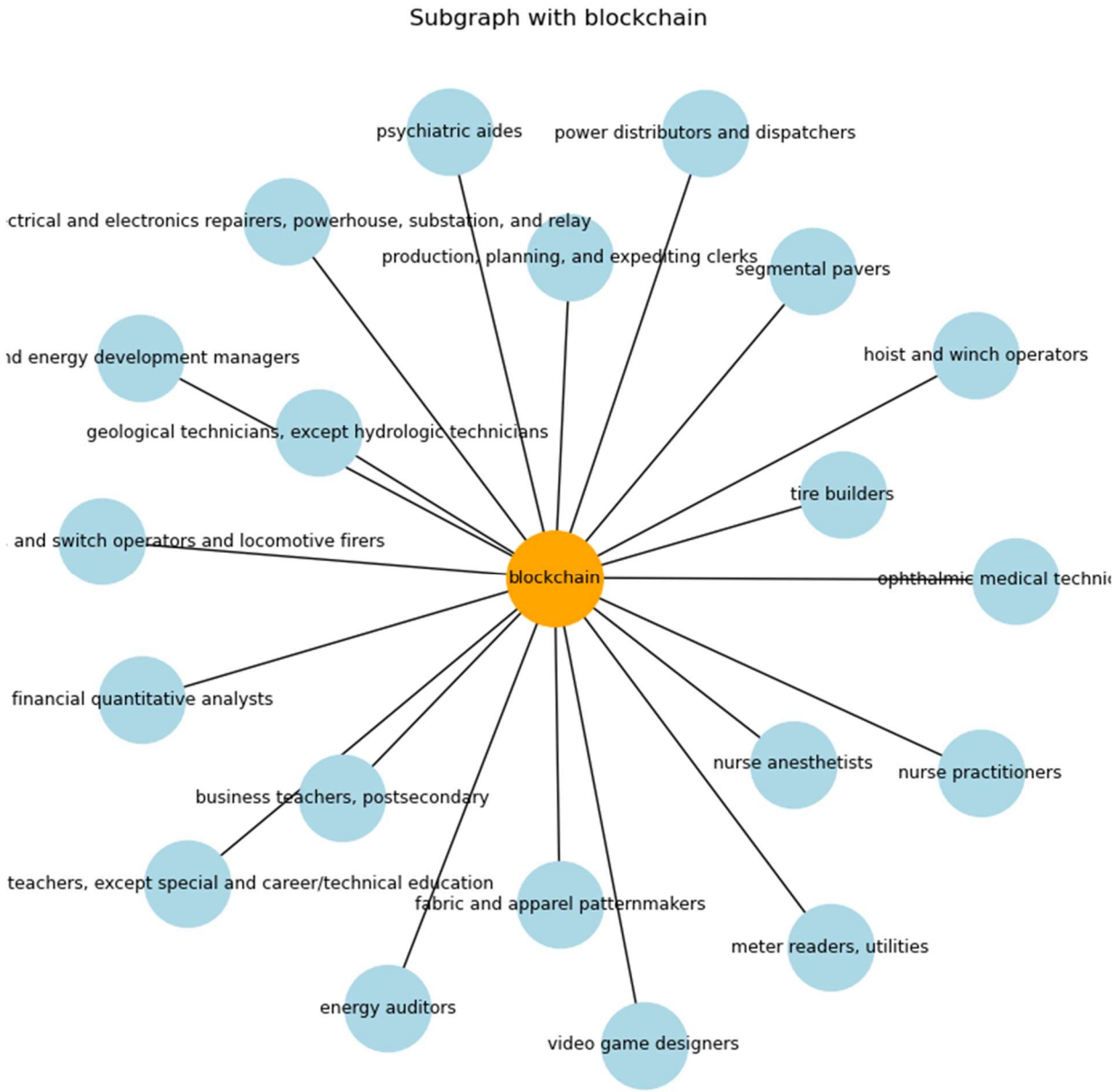
```
: def visualize_technology_subgraph(B, tech_node):
    """Visualizing the immediate subgraph around the new technology node."""
    connected_nodes = [tech_node] + list(B.neighbors(tech_node))
    sub_B = B.subgraph(connected_nodes)

    plt.figure(figsize=(8, 8))
    pos = nx.spring_layout(sub_B)

    labels = {node: (soc_code_to_title.get(node, node) if node in occupations else node) for node in connected_nodes}

    nx.draw(sub_B, pos, with_labels=False, node_size=2000, node_color='lightblue')

    nx.draw_networkx_labels(sub_B, pos, labels=labels, font_size=9)

    nx.draw_networkx_nodes(sub_B, pos, nodelist=[tech_node], node_size=2500, node_color='orange')
    plt.title(f"Subgraph with {tech_node}")
    plt.show()

soc_code_to_title = pd.concat([abilities[['O*NET-SOC Code', 'Title']], skills[['O*NET-SOC Code', 'Title']]]).drop_duplicates().s

n = 10  # Number of nodes
p = 0.5  # Probability of edge creation
base_graph = random_graph_model(n, p)
B = create_bipartite_graph(base_graph)
```

SOC Code replaced with occupation titles for better understanding

Random Graph Model on New Technology “Blockchain” -

```
#Running simulations for visualizing the results for blockchain skill
simulate_Rand(G_skills, 'blockchain', soc_code_to_title)
```

```
Added edge between psychiatric aides and blockchain
Added edge between energy auditors and blockchain
Added edge between middle school teachers, except special and career/technical education and blockchain
Added edge between electrical and electronics repairers, powerhouse, substation, and relay and blockchain
Added edge between power distributors and dispatchers and blockchain
Added edge between wind energy development managers and blockchain
Added edge between nurse anesthetists and blockchain
Added edge between tire builders and blockchain
Added edge between video game designers and blockchain
Added edge between production, planning, and expediting clerks and blockchain
```



Random Graph Model on New Technology “Artificial Intelligence” -

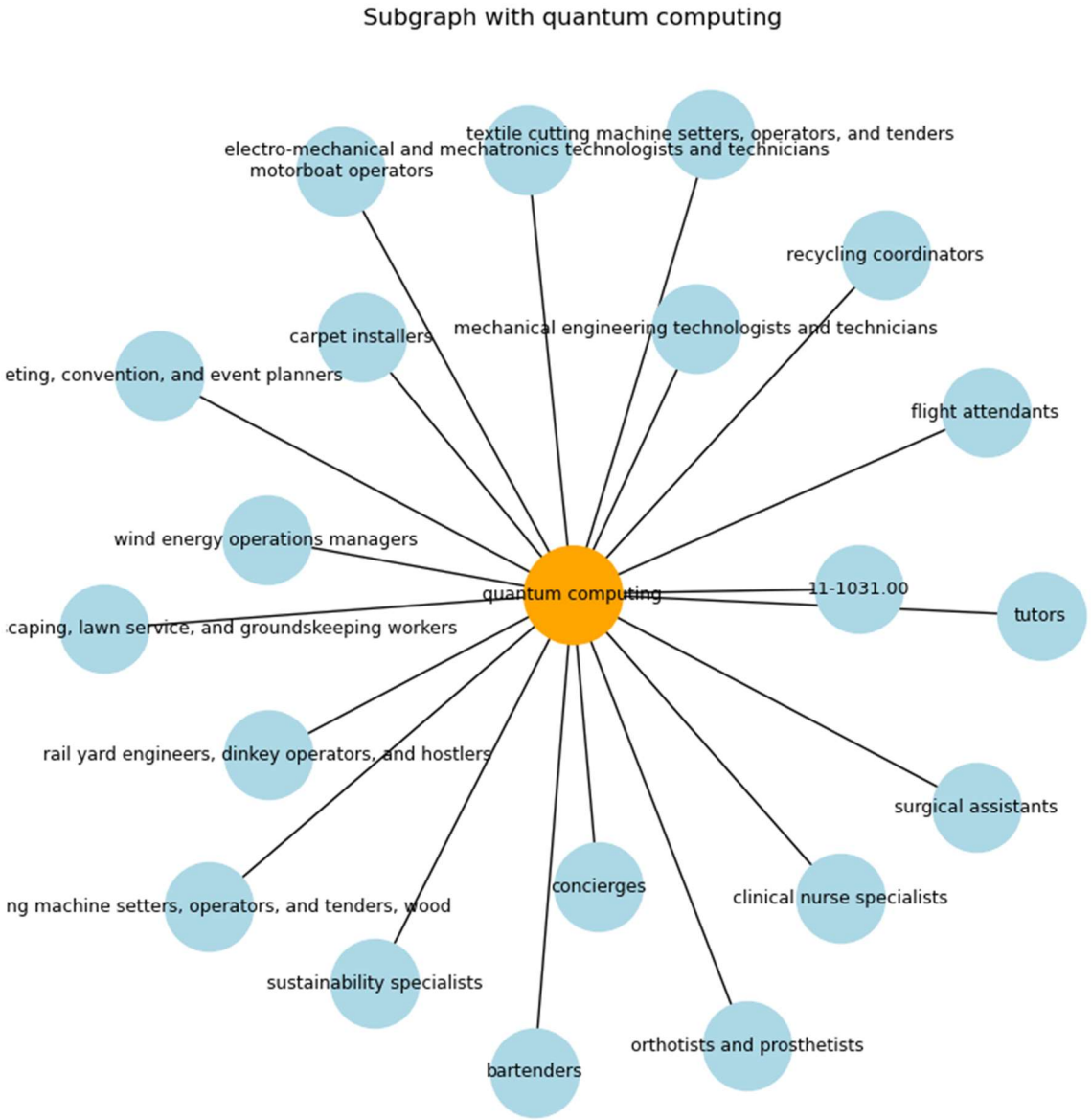
```
#Running simulations for visualizing the results for artificial intelligence ability

simulate_Rand(G_abilities, 'artificial intelligence', soc_code_to_title)
```

```
Added edge between postal service mail carriers and artificial intelligence
Added edge between computer systems analysts and artificial intelligence
Added edge between logging equipment operators and artificial intelligence
Added edge between mechanical drafters and artificial intelligence
Added edge between art directors and artificial intelligence
Added edge between rail yard engineers, dinkey operators, and hostlers and artificial intelligence
Added edge between farmworkers and laborers, crop, nursery, and greenhouse and artificial intelligence
Added edge between advertising sales agents and artificial intelligence
Added edge between motorboat operators and artificial intelligence
Added edge between surveyors and artificial intelligence
```

Random Graph Model on New Technology “Quantum Computing”

```
#Running simulations for visualizing the results for quantum computing tech skill

simulate_Rand(G_tech, 'quantum computing', soc_code_to_title)
```

```
Added edge between textile cutting machine setters, operators, and tenders and quantum computing
Added edge between clinical nurse specialists and quantum computing
Added edge between surgical assistants and quantum computing
Added edge between meeting, convention, and event planners and quantum computing
Added edge between carpet installers and quantum computing
Added edge between orthotists and prosthetists and quantum computing
Added edge between flight attendants and quantum computing
Added edge between motorboat operators and quantum computing
Added edge between rail yard engineers, dinkey operators, and hostlers and quantum computing
Added edge between bartenders and quantum computing
```



- Small World Model

Here, we create the small-world model using NetworkX's “watts_strogatz_graph” function, where each node is connected to “k” nearest neighbors, then rewire each edge with probability “p” to a random node.

Managing the complexity and computational overhead of creating and manipulating large-scale networks.

Also, ensuring a representative sample of occupations is chosen for connecting the new technology node, which can affect the simulation's realism.

So for that, we used the Watts-Strogatz model helped reduce complexity by generating a small-world network that maintains characteristics of real-world social networks but with a manageable number of connections.

Moreover, using random.sample function allowed for the random but uniform selection of occupation nodes to connect with the new technology, simulating a realistic spread.

```
def create_small_world_network(num_nodes, k, p):
    """
    Creating a small-world network using the Watts-Strogatz model.

    Parameters:
        - num_nodes: Total number of nodes in the network.
        - k: Each node is connected to k nearest neighbors in a ring topology.
        - p: Probability of rewiring each edge.

    Returns:
        - G: The generated small-world network.
    """
    return nx.watts_strogatz_graph(num_nodes, k, p)
```

This function simulates the addition of a new technology node to a bipartite graph and connects it with a random sample of occupations. It then visualizes the immediate subgraph around the new technology node, illustrating its connections and related nodes.

```
def simulate_SWN(B, new_tech_name):
    # Addinf the new technology to the graph
    B.add_node(new_tech_name, bipartite=1, type='technology')

    # Connecting it with a random sample of occupations
    sample_occupations = random.sample(list(occupations),10)  # Exclude the new tech node
    for occ in sample_occupations:
        B.add_edge(occ, new_tech_name)
        print(f"Added edge between {soc_code_to_title.get(occ, occ)} and {new_tech_name}")

    # Visualizing the subgraph for the new technology
    visualize_technology_subgraph(B, new_tech_name)
```

```python
#Visualization of subgraph

def visualize_technology_subgraph(B, tech_node):
    # Extracting the immediate subgraph around the new technology
    connected_nodes = [tech_node] + list(B.neighbors(tech_node))
    sub_B = B.subgraph(connected_nodes)

    labels = {node: (soc_code_to_title.get(node, node) if node in occupations else node) for node in connected_nodes}

    # Visualization setup
    plt.figure(figsize=(8, 8))
    pos = nx.spring_layout(sub_B)

    # Drawing the nodes and edges
    nx.draw(sub_B, pos, with_labels=False, node_size=2000, node_color='lightblue')

    # Drawing custom labels
    nx.draw_networkx_labels(sub_B, pos, labels=labels, font_size=9)

    # Highlighting the new technology node
    nx.draw_networkx_nodes(sub_B, pos, nodelist=[tech_node], node_size=2500, node_color='orange')

    plt.title(f"S.W.N - Subgraph with {tech_node}")
    plt.show()
```

```python
num_nodes = 20  # Total number of nodes
k = 4           # Each node is connected to k nearest neighbors
p = 0.3         # Probability of rewiring each edge

# Creating the small-world network
small_world_network = create_small_world_network(num_nodes, k, p)
```

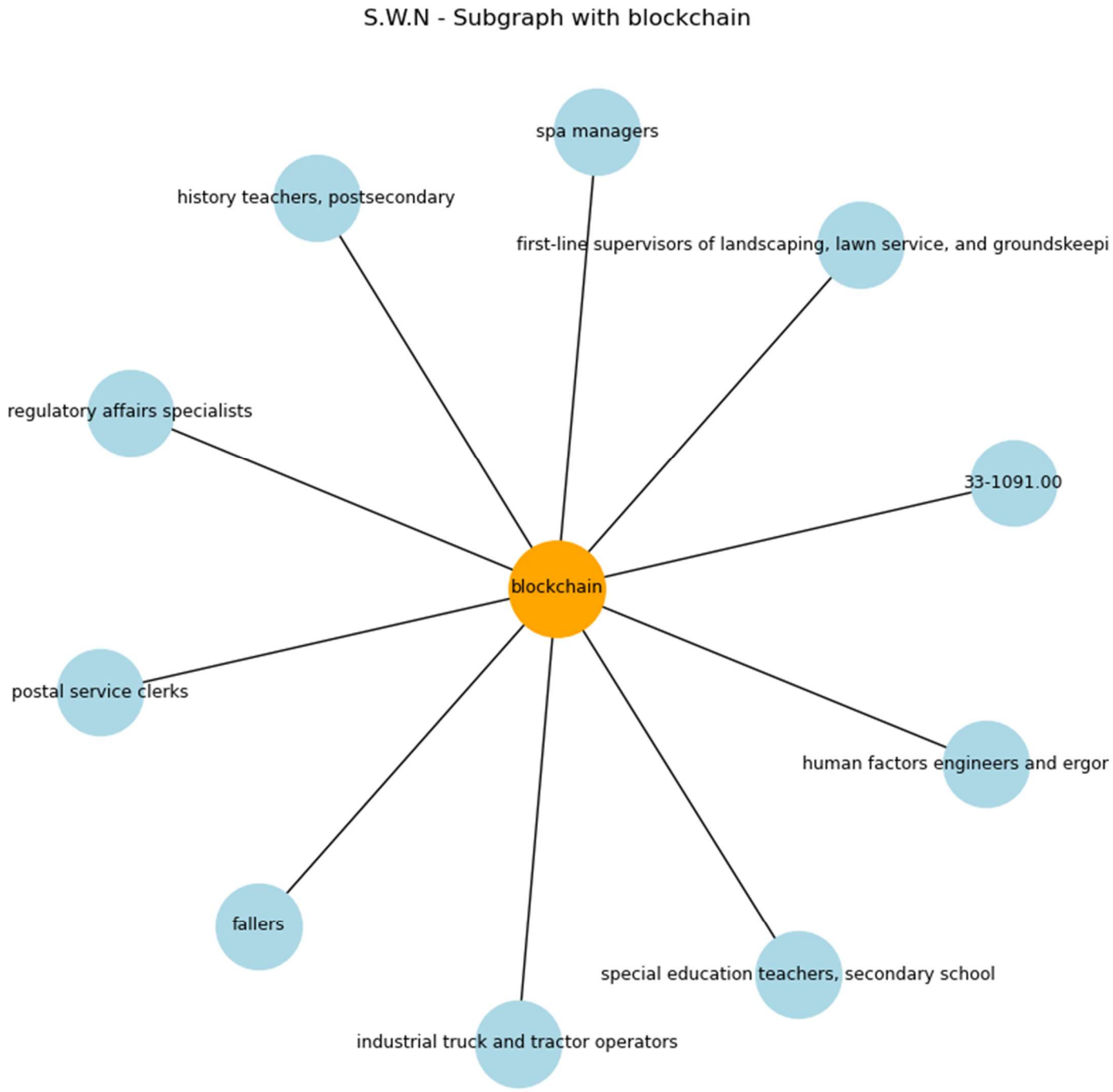
- Visualizations

Small World Model on New Technology “Blockchain”

```python
# Simulating the the model for "blockchain" skill
simulate_SWN(small_world_network, "blockchain")
```

```
Added edge between first-line supervisors of landscaping, lawn service, and groundskeeping workers and blockchain
Added edge between special education teachers, secondary school and blockchain
Added edge between human factors engineers and ergonomists and blockchain
Added edge between fallers and blockchain
Added edge between regulatory affairs specialists and blockchain
Added edge between 33-1091.00 and blockchain
Added edge between history teachers, postsecondary and blockchain
Added edge between spa managers and blockchain
Added edge between industrial truck and tractor operators and blockchain
Added edge between postal service clerks and blockchain
```

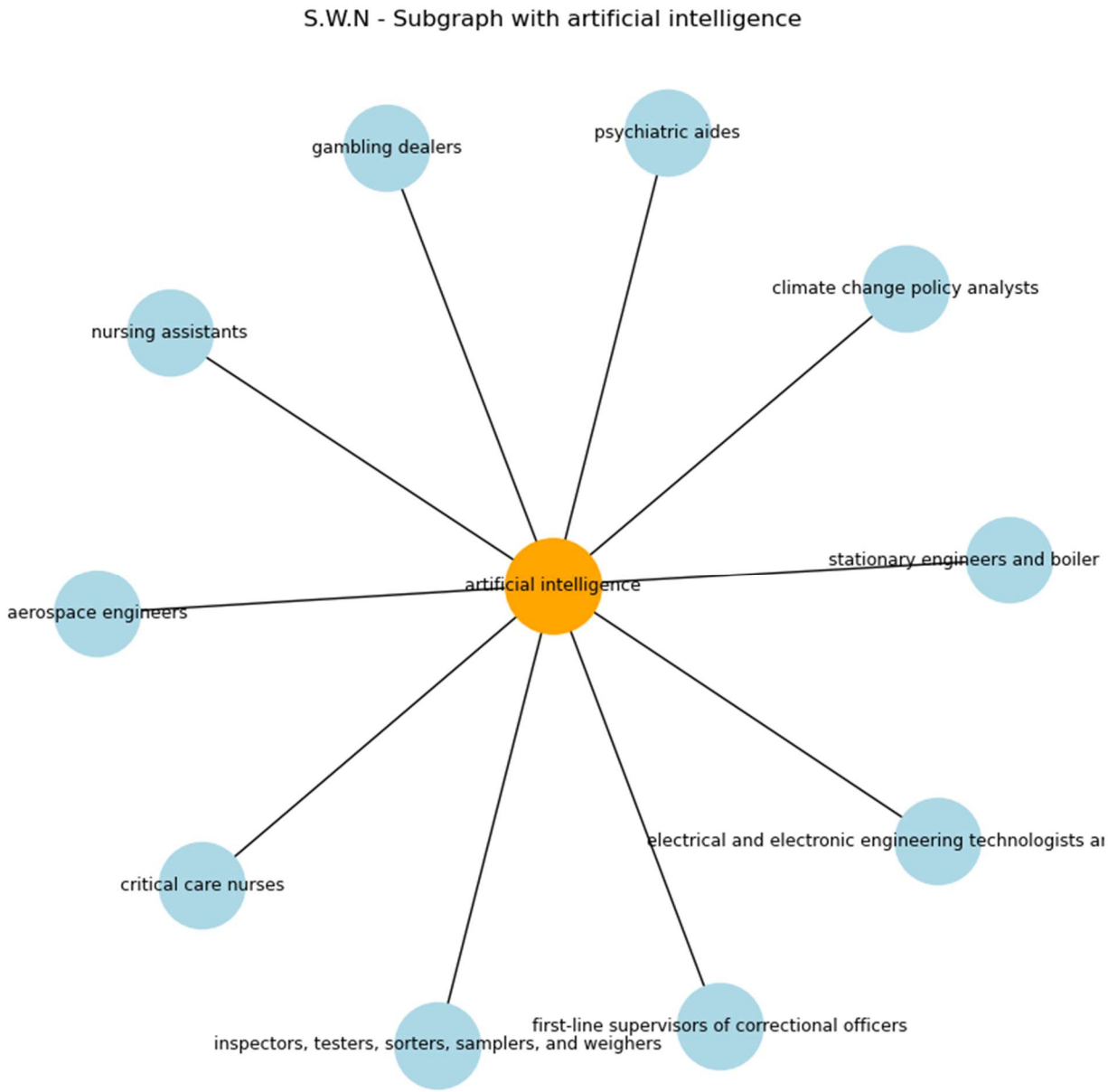
Output:


S.W.N - Subgraph with blockchain
spa managers
history teachers, postsecondary
first-line supervisors of landscaping, lawn service, and groundskeepi
regulatory affairs specialists
33-1091.00
blockchain
postal service clerks
human factors engineers and ergor
fallers
special education teachers, secondary school
industrial truck and tractor operators

Small World Model on New Technology “Artificial Intelligence”

```
# Simulating the the model for "artificial intelligence" ability

simulate_SWN(small_world_network, "artificial intelligence")
```

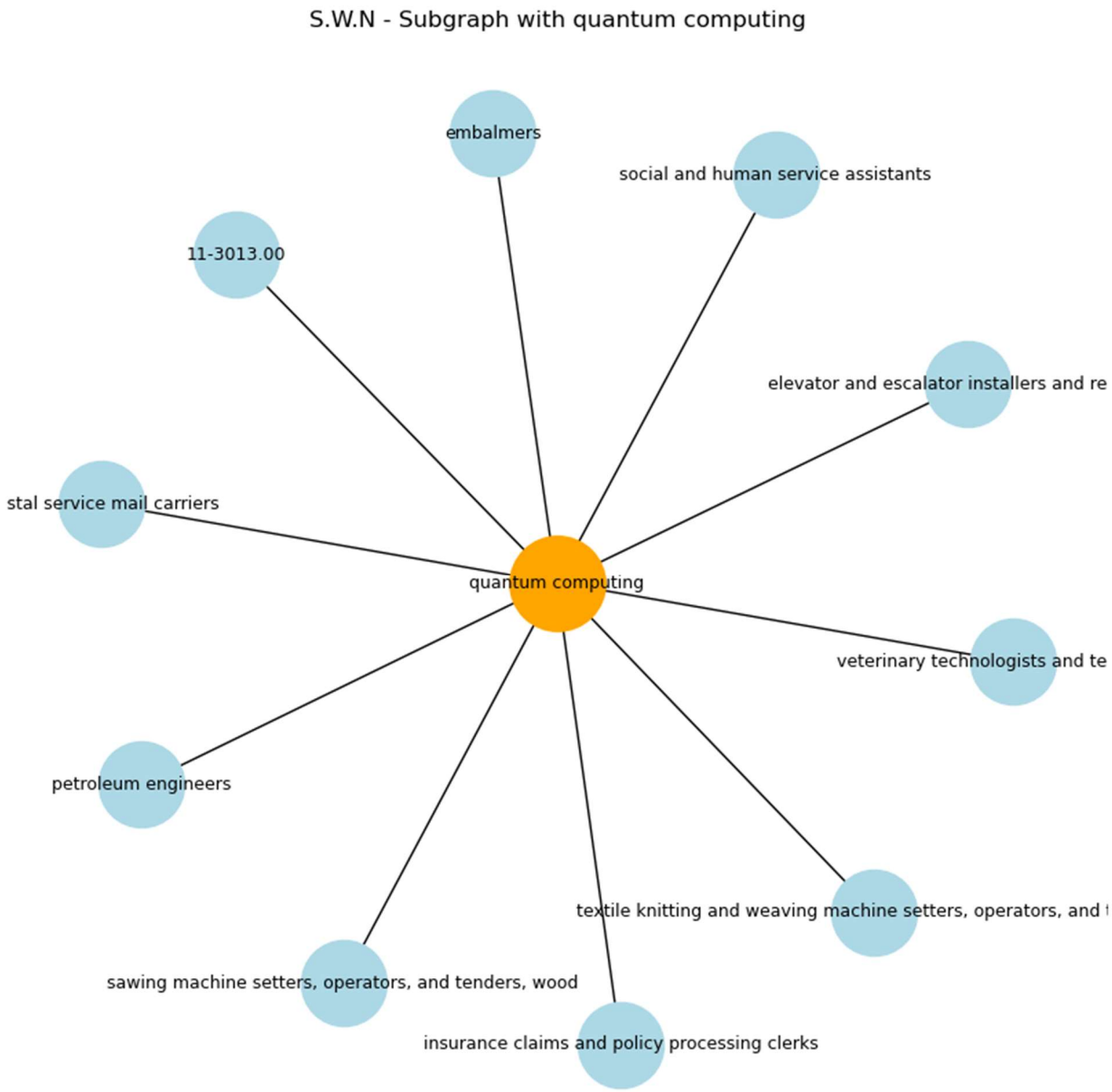
```
Added edge between aerospace engineers and artificial intelligence
Added edge between inspectors, testers, sorters, samplers, and weighers and artificial intelligence
Added edge between gambling dealers and artificial intelligence
Added edge between climate change policy analysts and artificial intelligence
Added edge between electrical and electronic engineering technologists and technicians and artificial intelligence
Added edge between first-line supervisors of correctional officers and artificial intelligence
Added edge between nursing assistants and artificial intelligence
Added edge between psychiatric aides and artificial intelligence
Added edge between critical care nurses and artificial intelligence
Added edge between stationary engineers and boiler operators and artificial intelligence
```



Small World Model on New Technology “Quantum Computing”

```
# Simulating the the model for "quantum computing" tech skill

simulate_SWN(small_world_network, "quantum computing")
```

```
Added edge between insurance claims and policy processing clerks and quantum computing
Added edge between textile knitting and weaving machine setters, operators, and tenders and quantum computing
Added edge between elevator and escalator installers and repairers and quantum computing
Added edge between postal service mail carriers and quantum computing
Added edge between veterinary technologists and technicians and quantum computing
Added edge between embalmers and quantum computing
Added edge between petroleum engineers and quantum computing
Added edge between social and human service assistants and quantum computing
Added edge between sawing machine setters, operators, and tenders, wood and quantum computing
Added edge between 11-3013.00 and quantum computing
```



- Preferential Attachment Model

This function generates a network based on the Barabási-Albert preferential attachment model, which is commonly used to simulate networks that exhibit scale-free properties. This model is particularly relevant for understanding how new nodes (e.g., technologies) are integrated into existing networks.

Handling the complexity and computational demands of simulating large networks was challenging.

So we used, NetworkX's efficient implementation of the Barabási-Albert model helps manage the computational load.

```
def create_preferential_attachment_network(num_nodes, m):
    """
    Using the Barabási-Albert preferential attachment model to create the network

    Parameters:
        - num_nodes: Total number of nodes in the network.
        - m: Number of edges to attach from a new node to existing nodes.

    Returns:
        - G: The generated preferential attachment network.
    """
    return nx.barabasi_albert_graph(num_nodes, m)
```

```
def simulate_PAN(B, new_tech_name):
    # Adding the new technology to the graph
    B.add_node(new_tech_name, bipartite=1, type='technology')

    # Connectting it with a random sample of occupations
    sample_occupations = random.sample(list(occupations),10)  # Exclude the new tech node
    for occ in sample_occupations:
        B.add_edge(occ, new_tech_name)
        print(f"Added edge between {soc_code_to_title.get(occ, occ)} and {new_tech_name}")

    # Visualization of the subgraph
    visualize_technology_subgraph(B, new_tech_name)
```

This function will visualize the subgraph for the technology node

```
def visualize_technology_subgraph(B, tech_node):
    # Extracting the immediate subgraph around the new technology
    connected_nodes = [tech_node] + list(B.neighbors(tech_node))
    sub_B = B.subgraph(connected_nodes)

    labels = {node: (soc_code_to_title.get(node, node) if node in occupations else node) for node in connected_nodes}

    # Visualization setup
    plt.figure(figsize=(8, 8))
    pos = nx.spring_layout(sub_B)

    # Drawing the nodes and edges
    nx.draw(sub_B, pos, with_labels=False, node_size=2000, node_color='lightblue')

    nx.draw_networkx_labels(sub_B, pos, labels=labels, font_size=9)

    # Highlighting the new technology node
    nx.draw_networkx_nodes(sub_B, pos, nodelist=[tech_node], node_size=2500, node_color='orange')

    plt.title(f"P.A.N - Subgraph with {tech_node}")
    plt.show()
```

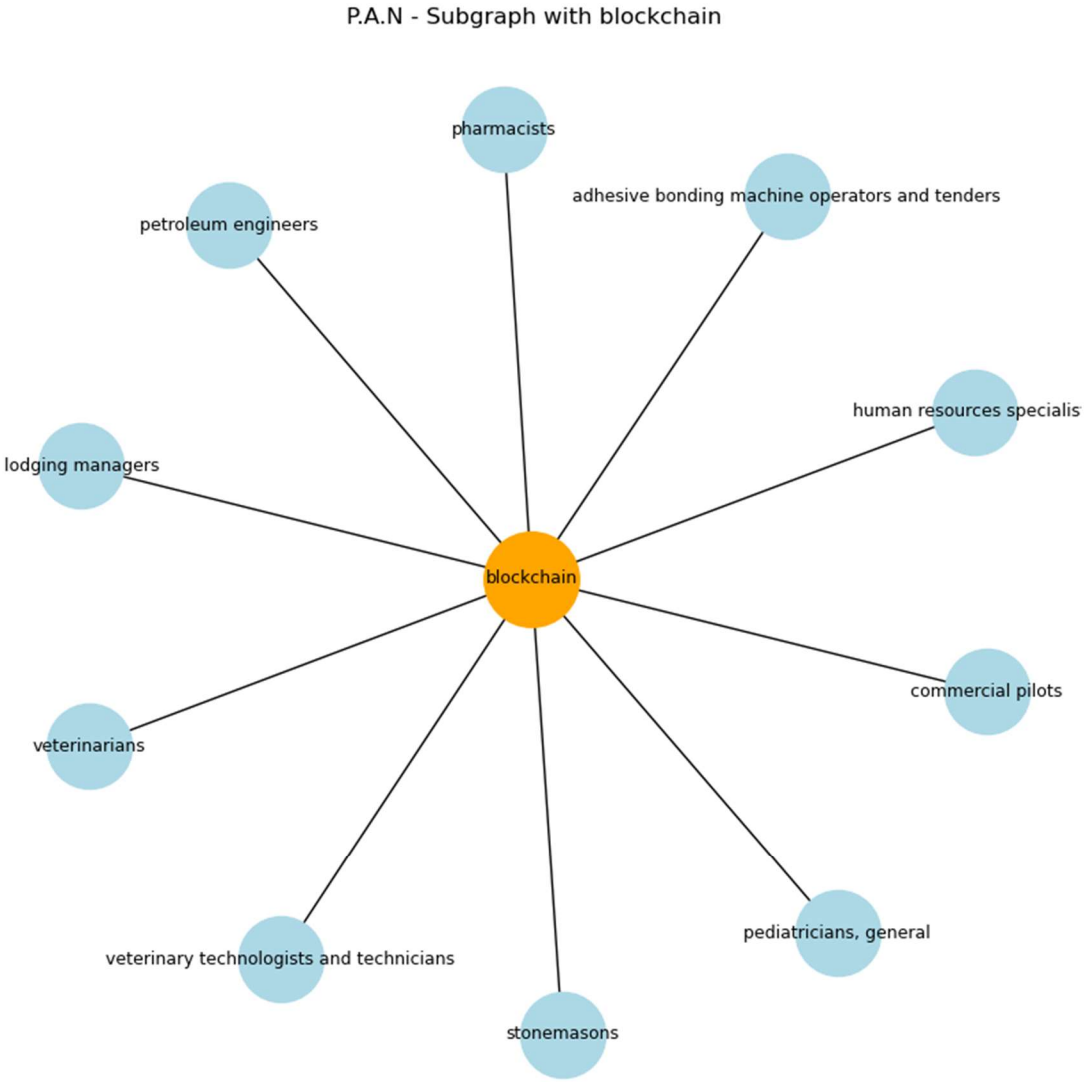
```
num_nodes = 20  # Total number of nodes
m = 3           # Number of edges to attach from a new node to existing nodes

# Creating the preferential attachment network
preferential_attachment_network = create_preferential_attachment_network(num_nodes, m)
```

- Visualizations

Preferential Attachment Model on New Technology “Blockchain”

```
# Simulation for "blockchain"
simulate_PAN(preferential_attachment_network, "blockchain")
```

```
Added edge between pediatricians, general and blockchain
Added edge between veterinarians and blockchain
Added edge between human resources specialists and blockchain
Added edge between pharmacists and blockchain
Added edge between petroleum engineers and blockchain
Added edge between commercial pilots and blockchain
Added edge between stonemasons and blockchain
Added edge between veterinary technologists and technicians and blockchain
Added edge between adhesive bonding machine operators and tenders and blockchain
Added edge between lodging managers and blockchain
```



Preferential Attachment Model on New Technology “Artificial Intelligence”

```
#Simulation for "artificial intelligence"
simulate_PAN(preferential_attachment_network, "artificial intelligence")
```

```
Added edge between architectural and civil drafters and artificial intelligence
Added edge between umpires, referees, and other sports officials and artificial intelligence
Added edge between adapted physical education specialists and artificial intelligence
Added edge between mixing and blending machine setters, operators, and tenders and artificial intelligence
Added edge between sports medicine physicians and artificial intelligence
Added edge between upholsterers and artificial intelligence
Added edge between special education teachers, preschool and artificial intelligence
Added edge between training and development specialists and artificial intelligence
Added edge between radio, cellular, and tower equipment installers and repairers and artificial intelligence
Added edge between geoscientists, except hydrologists and geographers and artificial intelligence
```

Preferential Attachment Model on New Technology “Quantum Computing”

```
#Simulation for "quantum computing"
simulate_PAN(preferential_attachment_network, "quantum computing")
```

```
Added edge between paving, surfacing, and tamping equipment operators and quantum computing
Added edge between fine artists, including painters, sculptors, and illustrators and quantum computing
Added edge between rail-track laying and maintenance equipment operators and quantum computing
Added edge between regulatory affairs specialists and quantum computing
Added edge between first-line supervisors of retail sales workers and quantum computing
Added edge between transportation vehicle, equipment and systems inspectors, except aviation and quantum computing
Added edge between tool and die makers and quantum computing
Added edge between choreographers and quantum computing
Added edge between political science teachers, postsecondary and quantum computing
Added edge between drilling and boring machine tool setters, operators, and tenders, metal and plastic and quantum computing
```

g. Modeling Non-Obvious Connections

We create and use the function that creates a network G_indirect representing indirect relationships between occupations based on shared requirements. It adds occupation nodes and connects them based on shared skills and abilities. If two occupations share a skill or ability, they are indirectly connected in the graph.

Determining how to connect occupations based on shared skills or abilities indirectly required a thoughtful approach to truly reflect the subtle relationships and influences within the occupational data.

So by initializing the edges with a weight and then incrementally increasing this weight for additional shared connections, the model could dynamically illustrate the strength of indirect connections between occupations based on shared skills or abilities.

```
# creating a network with indirect relationships
G_indirect = nx.Graph()

# Adding all occupation nodes
occupations = pd.concat([abilities, skills, tech_skills])['O*NET-SOC Code'].unique()
G_indirect.add_nodes_from(occupations, type='occupation')

# Adding connections based on shared requirements indirectly
# Here, for simplicity, we are considering skills and abilities combined
all_skills_abilities = pd.concat([skills['Element Name'], abilities['Element Name']]).unique()

for skill in all_skills_abilities:
    occ_with_skill = skills[skills['Element Name'] == skill]['O*NET-SOC Code'].unique()

    # Connecting these occupations indirectly
    for i in range(len(occ_with_skill)):
        for j in range(i + 1, len(occ_with_skill)):
            if not G_indirect.has_edge(occ_with_skill[i], occ_with_skill[j]):
                G_indirect.add_edge(occ_with_skill[i], occ_with_skill[j], weight=1)
            else:
                # If the edge exists, increasing the weight
                G_indirect[occ_with_skill[i]][occ_with_skill[j]]['weight'] += 1
```

- Simulate Scenario / Scenario: Impact of a New Skill

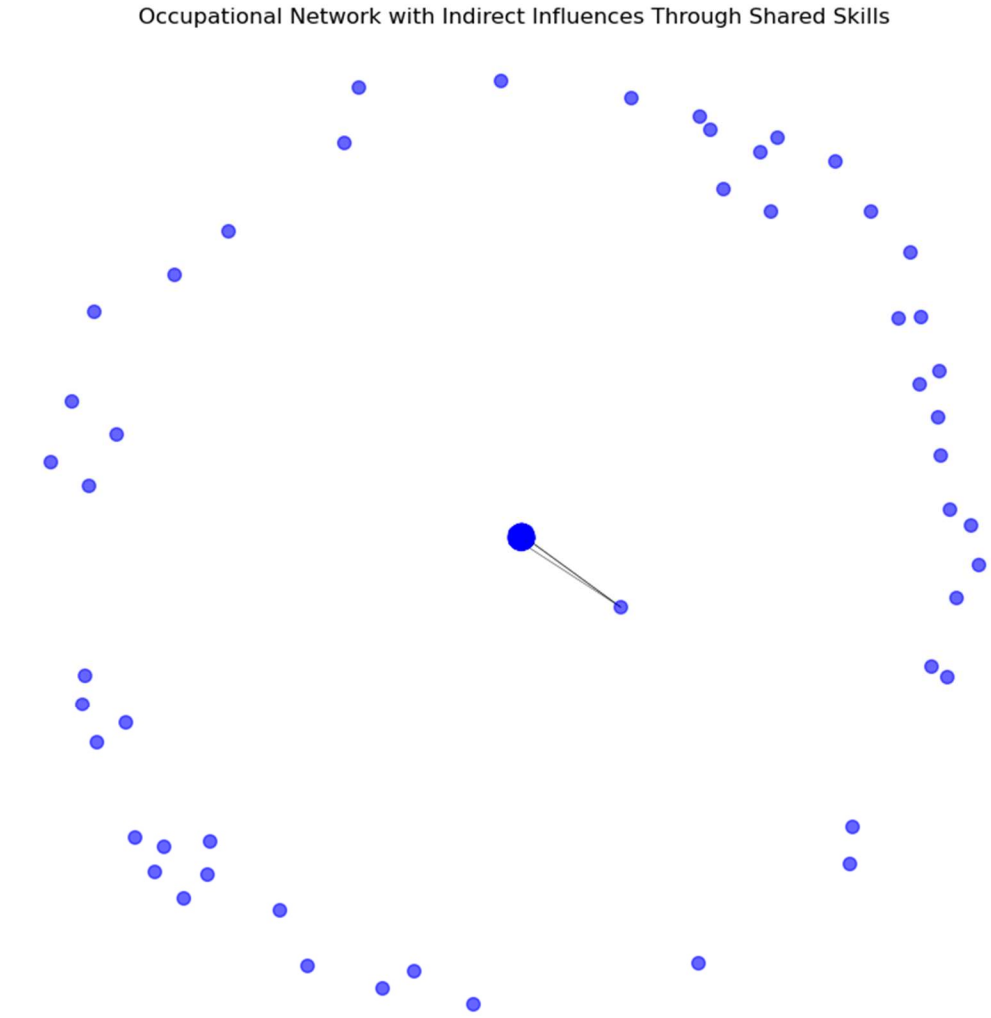
The function, simulate_new_skill_impact, simulates the impact of introducing a new critical skill, such as "Data Science," to a network represented by graph. It adds the skill node to the network and connects it with selected occupations (affected_occupations). It also updates indirect connections by increasing the weights of edges between occupations that share the new skill.

```
def simulate_new_skill_impact(graph, new_skill, affected_occupations):
    # Adding new skill to the network
    graph.add_node(new_skill, type='skill')

    # Connecting the new skill with selected occupations
    for occ in affected_occupations:
        graph.add_edge(occ, new_skill, weight=2)  # Stronger initial weight due to high relevance

    # Updating the indirect connections
    for i in range(len(affected_occupations)):
        for j in range(i + 1, len(affected_occupations)):
            if not graph.has_edge(affected_occupations[i], affected_occupations[j]):
                graph.add_edge(affected_occupations[i], affected_occupations[j], weight=1)
            else:
                graph[affected_occupations[i]][affected_occupations[j]]['weight'] += 1

# Simulating introducing "Data Science" as a new critical skill
simulate_new_skill_impact(G_indirect, 'Data Science', ['15-1253.00', '15-2041.00', '11-3021.00'])
```

- Visualize

For visualization, the function (visualize_graph_with_weights) plots a network graph with nodes representing occupations and edges representing indirect connections through shared skills. The thickness of the edges corresponds to the weight of the connections, allowing for a visual understanding of the strength of these indirect influences.

```
def visualize_graph_with_weights(graph, title):
    plt.figure(figsize=(10, 10))
    pos = nx.spring_layout(graph, k=0.15, iterations=50)

    #Drawing nodes
    nx.draw_networkx_nodes(graph, pos, node_color='blue', node_size=50, alpha=0.6)

    #Drawing edges based on weight
    edges = nx.get_edge_attributes(graph, 'weight')
    nx.draw_networkx_edges(graph, pos, edgelist=edges.keys(), width=[w * 0.15 for w in edges.values()])

    plt.title(title)
    plt.axis('off')
    plt.show()

#Visualization of the graph with updated indirect connections
visualize_graph_with_weights(G_indirect, 'Occupational Network with Indirect Influences Through Shared Skills')
```

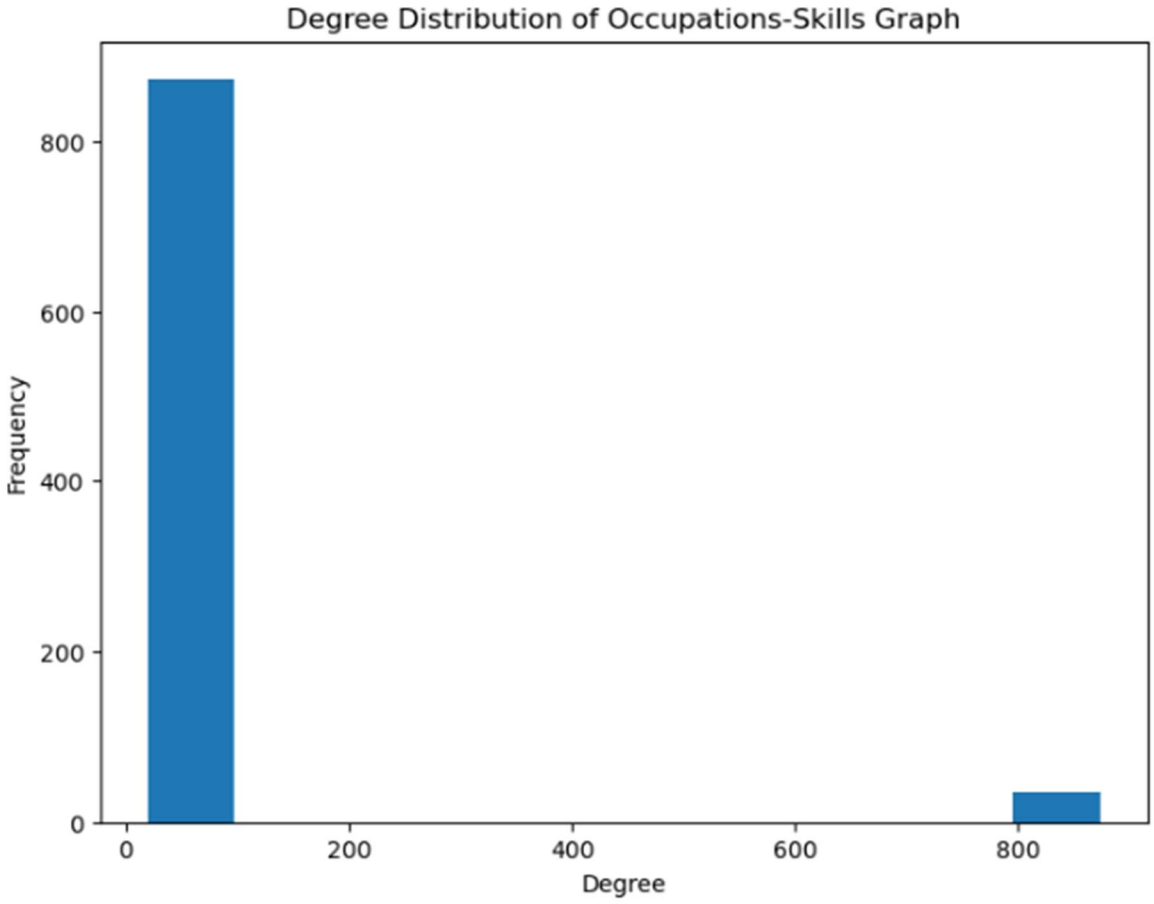


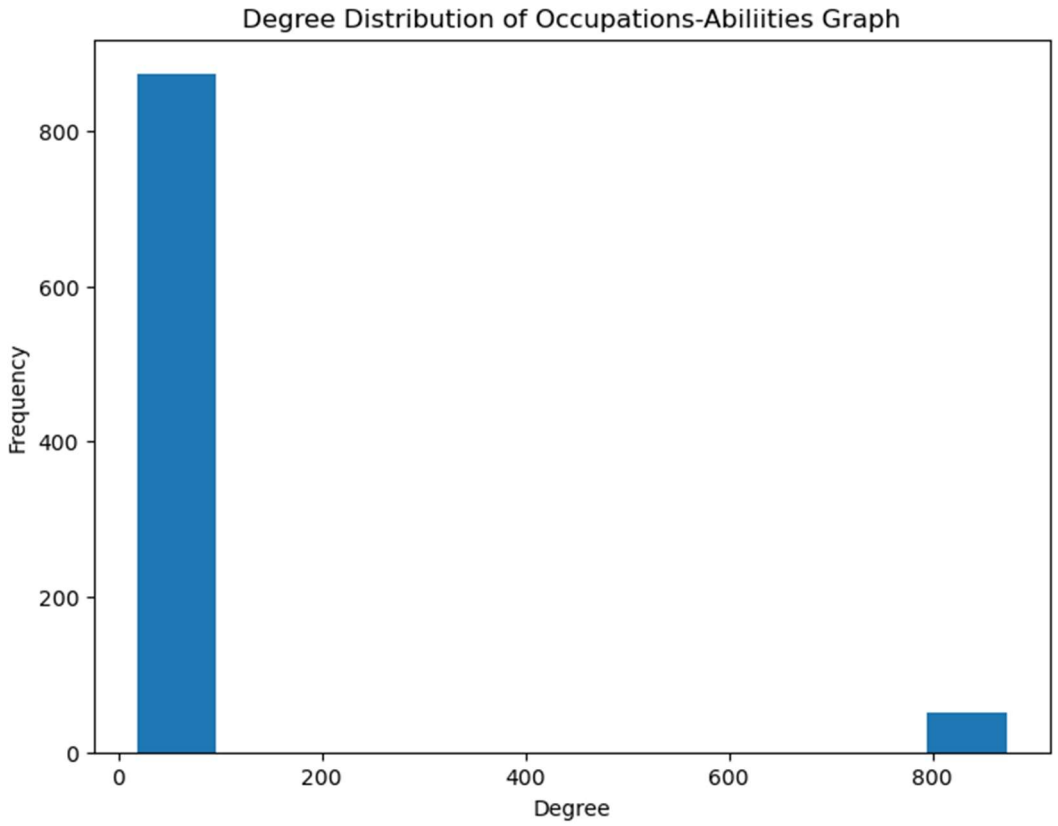
h. Visualizations Degree Distribution – Clustering Coefficients

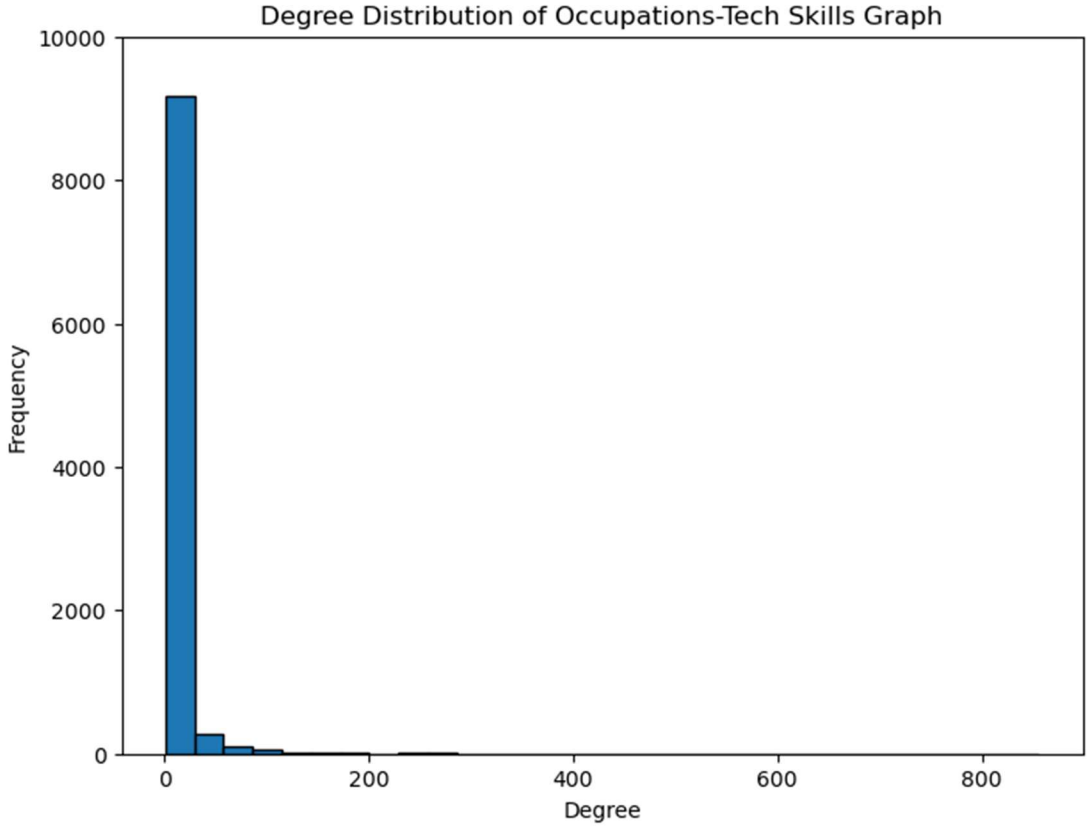
Degree Distribution for Occupations vs Skills - Network Graph

```
# Function for visualizing Degree Distribution Plot
def degree_distribution_plot(B, title):
    degree_sequence = sorted([d for n, d in B.degree()], reverse=True)  # Degree sequence
    plt.figure(figsize=(8, 6))
    plt.hist(degree_sequence, bins='auto')
    plt.title(title)
    plt.xlabel('Degree')
    plt.ylabel('Frequency')
    plt.show()
```

```
#Degree Distribution plot of Occupations-Skills Graph

degree_distribution_plot(G_skills, 'Degree Distribution of Occupations-Skills Graph')
```



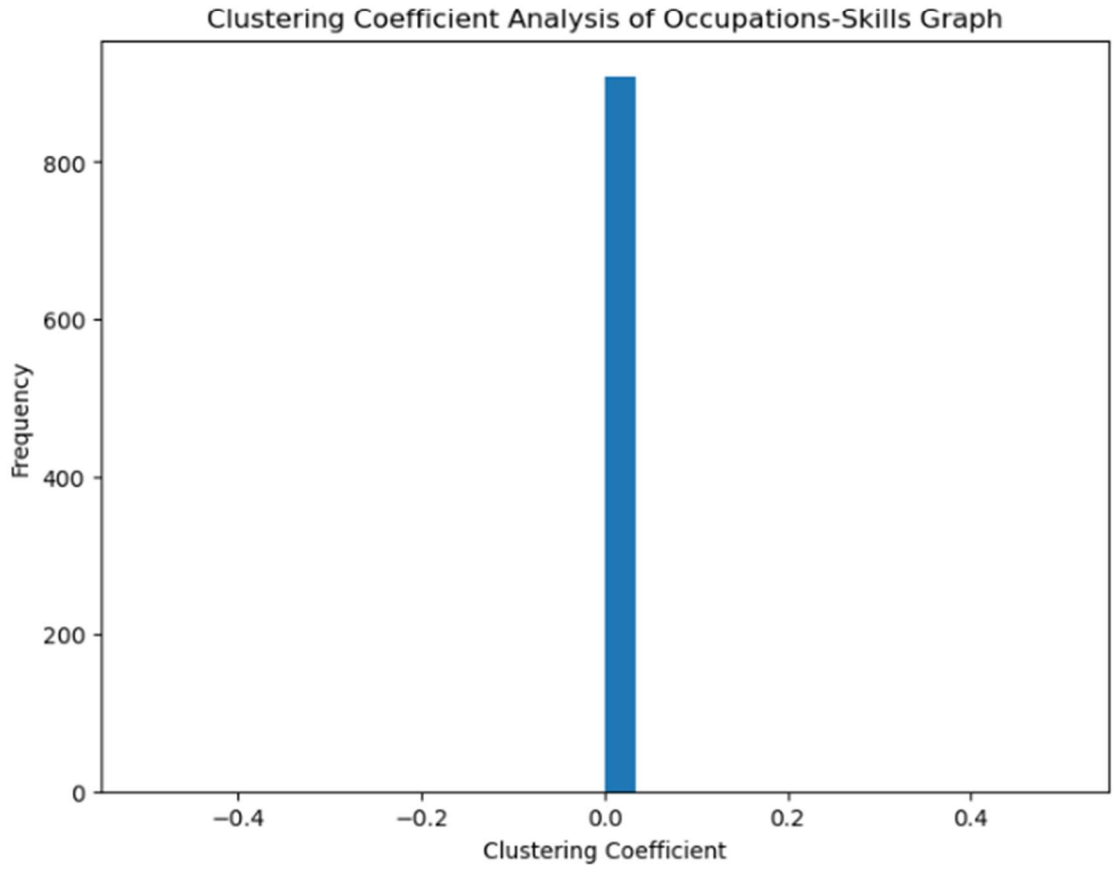

Degree Distribution of Occupations-Abiliities Graph
Frequency
Degree
0
200
400
600
800

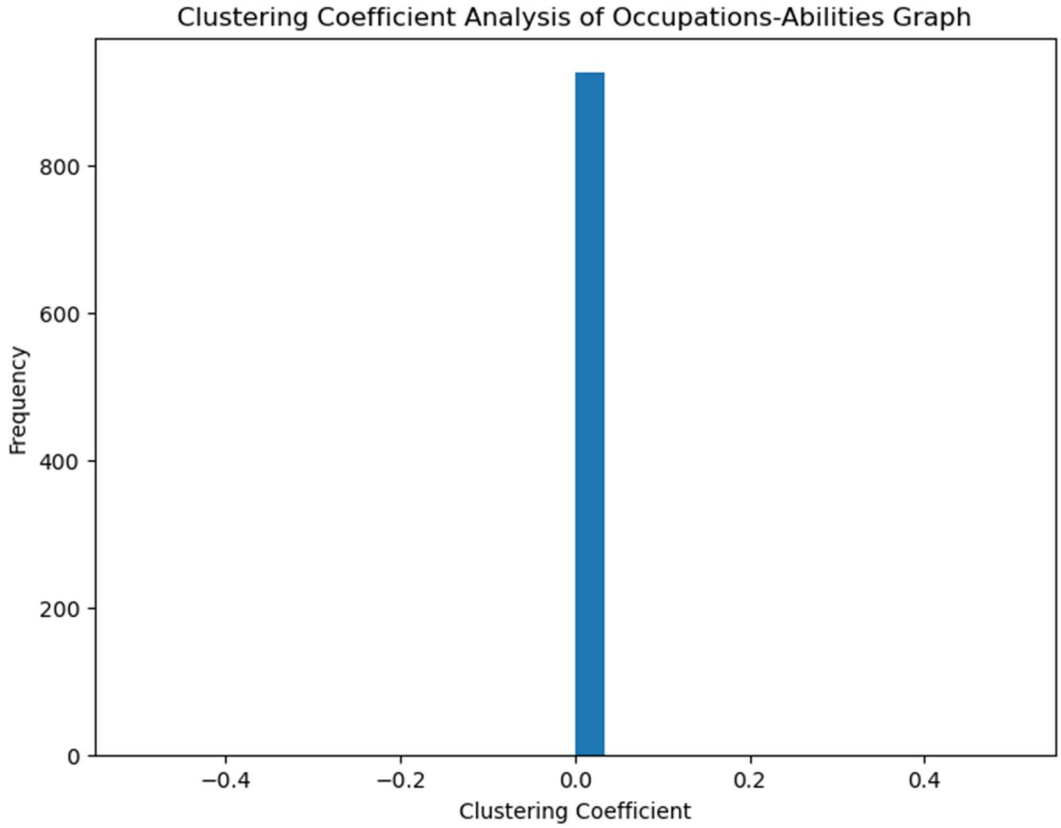

Degree Distribution of Occupations-Tech Skills Graph
Frequency
Degree
0
2000
4000
6000
8000
10000
200
400
600
800

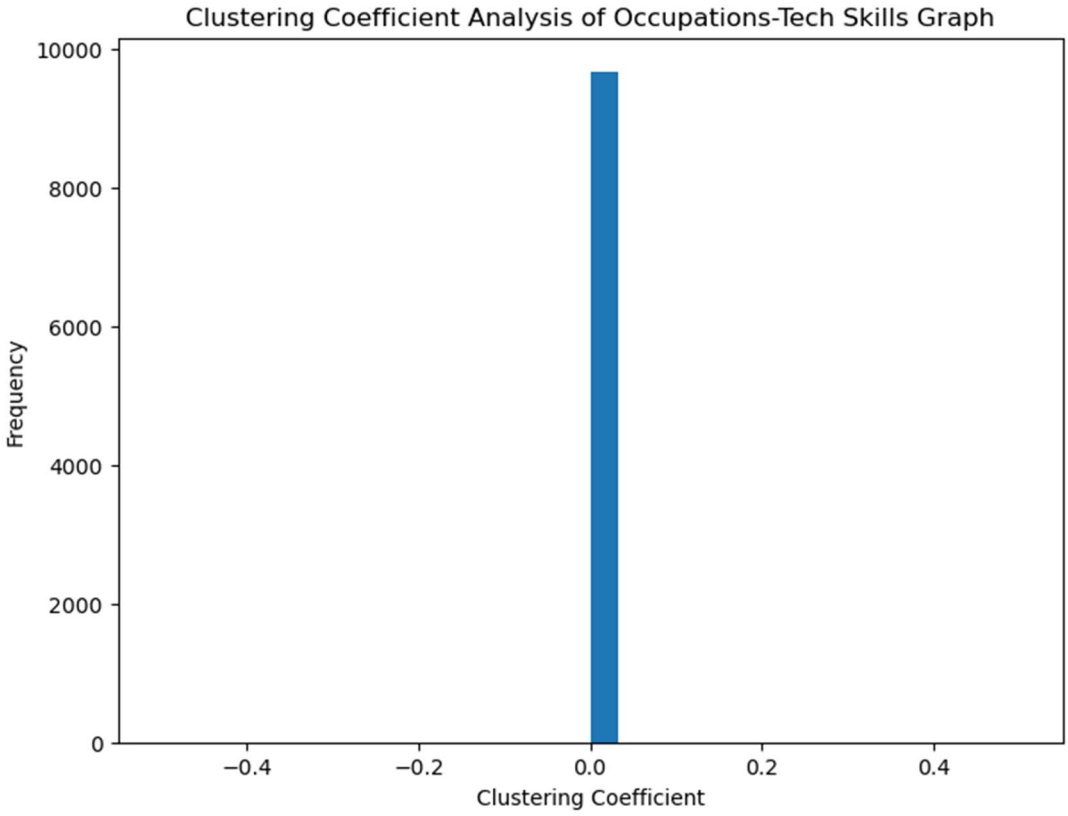
Clustering Coefficient for Occupations vs Skills - Network Graph

```
#Function to Analyze the clustering coefficient distribution of the graph

def clustering_coefficient_analysis(B, title):
    # Calculate the clustering coefficient for the graph
    clustering_coeffs = nx.clustering(B)
    plt.figure(figsize=(8, 6))
    plt.hist(list(clustering_coeffs.values()), bins=30)
    plt.title(title)
    plt.xlabel('Clustering Coefficient')
    plt.ylabel('Frequency')
    plt.show()
```

```
#Simulating Clustering Coefficient Analysis of Occupations-Skills Graph

clustering_coefficient_analysis(G_skills, 'Clustering Coefficient Analysis of Occupations-Skills Graph')
```




Clustering Coefficient Analysis of Occupations-Abilities Graph
Frequency
0
200
400
600
800
−0.4
−0.2
0.0
0.2
0.4
Clustering Coefficient

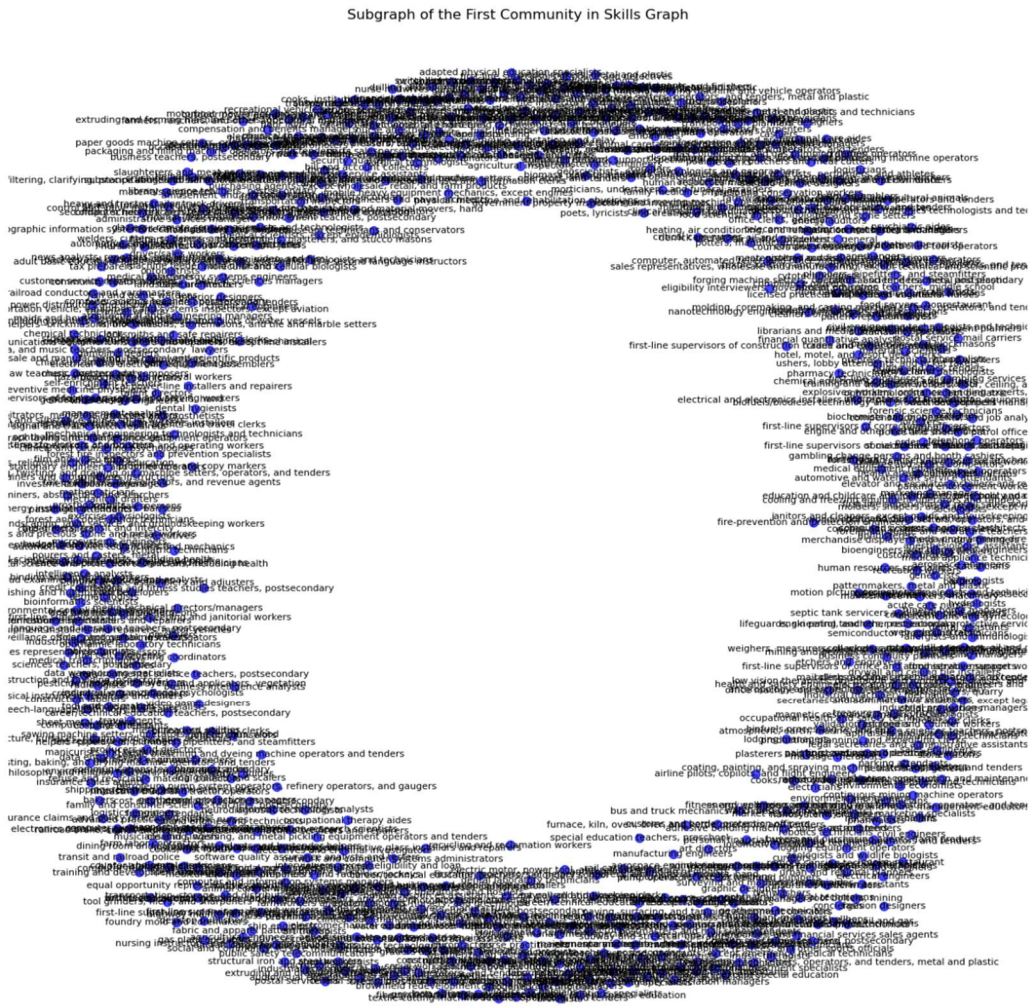

Clustering Coefficient Analysis of Occupations-Tech Skills Graph
Frequency
0
2000
4000
6000
8000
10000
−0.4
−0.2
0.0
0.2
0.4
Clustering Coefficient

i. Visualizations – Communities Subgraph

The function "visualize_subgraph()" is designed to visualize specific communities within a network graph. It helps in understanding how occupations or skills grouped into communities are interconnected, providing insights into clusters that share similar characteristics or requirements.

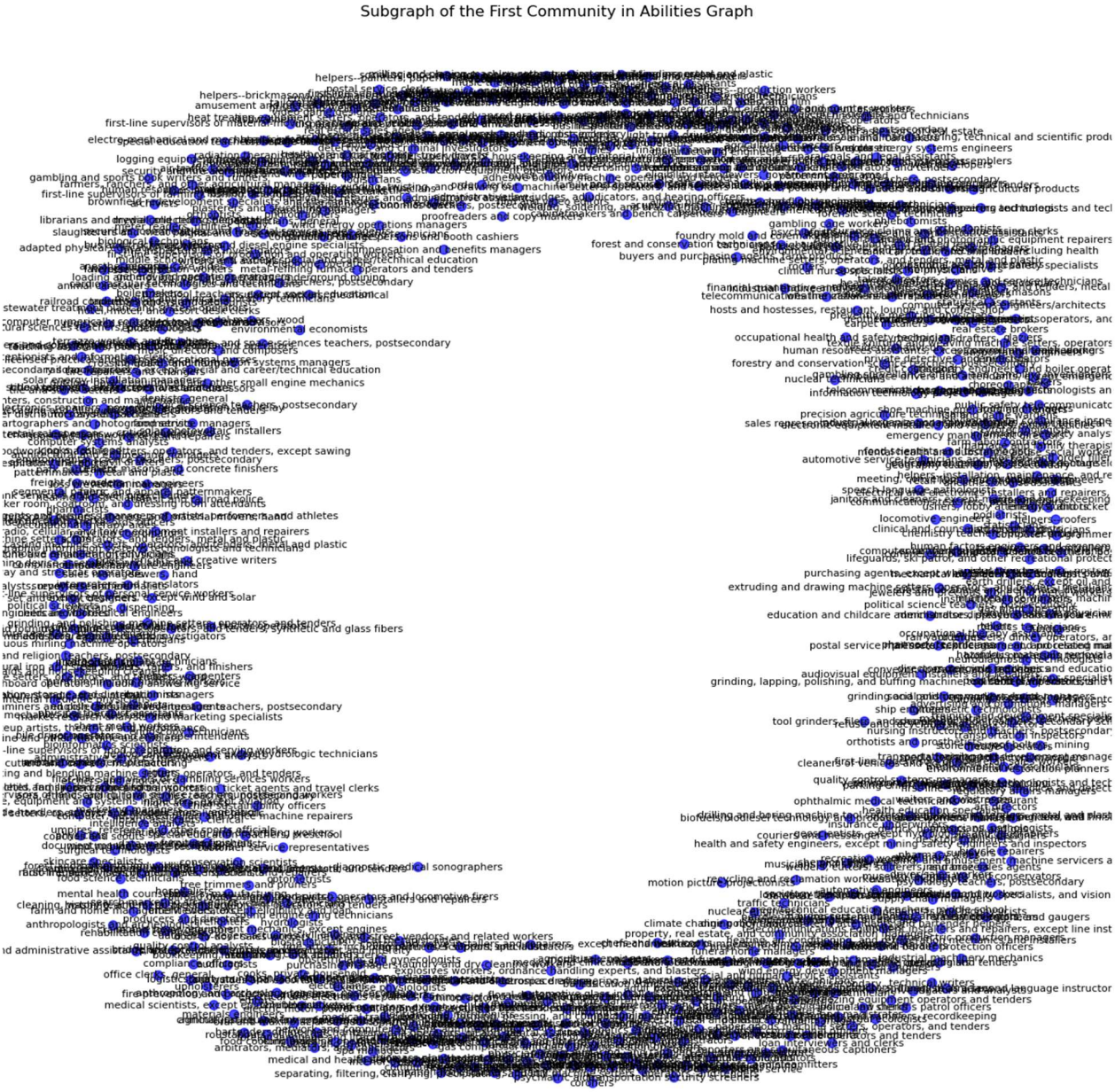
```
def visualize_subgraph(B, community, title, soc_code_to_title):
    sub_B = B.subgraph(community)
    plt.figure(figsize=(12, 12))
    pos = nx.spring_layout(sub_B, k=0.1, iterations=20)

    # Mapping the community's O*NET-SOC Codes to Titles
    labels = {node: soc_code_to_title.get(node, node) for node in community}

    # Drawing the subgraph
    nx.draw(sub_B, pos, with_labels=False, node_size=50, alpha=0.7, node_color='blue')
    nx.draw_networkx_labels(sub_B, pos, labels=labels, font_size=8)

    plt.title(title)
    plt.axis('off')
    plt.show()
```

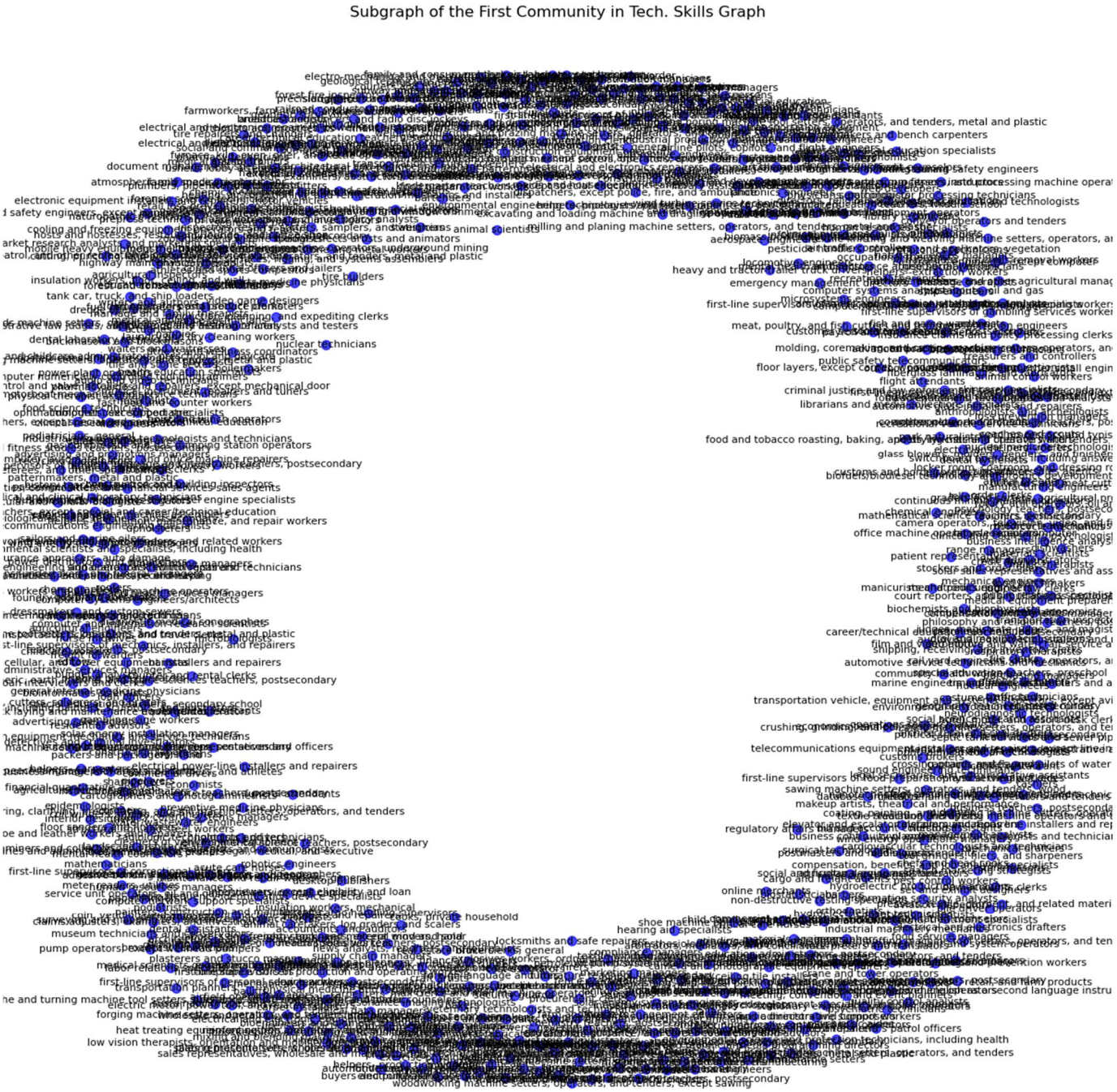
```
# Visualizing the community in Skills Graph with Titles instead of SOC Codes

if communities_skills:  # Ensure there is at least one community
    community_skills_titles = [soc_code_to_title.get(node, 'Unknown') for node in communities_skills[0]]
    visualize_subgraph(G_skills, communities_skills[0], 'Subgraph of the First Community in Skills Graph', soc_code_to_title)
```



Achieving an optimal layout that clearly delineates connections and clusters within the community can be computationally intensive.

Here we used a force-directed layout that helps naturally separate nodes that are not closely connected, improving visual clarity and focusing attention on the structure of connections within the community.





j. Heatmap Correlation – Code - Visualizations (In Result)

The below code provided describes the process of creating a heatmap to visualize the importance of top skills across top occupations using data from the O*NET database.

The skills and abilities data are concatenated to form a single DataFrame. This allows for a unified view of both skills and abilities across all occupations.

Then, the data is grouped by O*NET-SOC Code and Element Name, and the mean of Data Value is calculated. This aggregation helps in deriving a single representative value for each skill and occupation combination, essential for accurate visualization.

Here, the top 10 occupations and skills with the highest average Data Value are identified. These represent the occupations that place the highest importance on certain skills and the skills that are most valued across occupations, respectively.

Then, the DataFrame is filtered to include only the top 10 occupations and top 10 skills. The filtered data is pivoted to create a matrix suitable for heatmap visualization, where each cell in the matrix represents the normalized importance of a skill within an occupation.

HeatMap for top 10 occupations vs top 10 important skills

```
# Merging the titles into the DataFrame
df_heatmap_data = df_heatmap_data.merge(skills[['O*NET-SOC Code', 'Title']].drop_duplicates(), on='O*NET-SOC Code')

df_heatmap_data = df_heatmap_data.groupby(['Title', 'Element Name']).agg({'Data Value': 'mean'}).reset_index()

# Identifying top 10 occupations with the highest average 'Data Value'
top_occupations_titles = (df_heatmap_data.groupby('Title')['Data Value']
                          .mean()
                          .sort_values(ascending=False)
                          .head(10)
                          .index)

# Identifying top 10 skills with the highest average 'Data Value'
top_skills = (df_heatmap_data.groupby('Element Name')['Data Value']
              .mean()
              .sort_values(ascending=False)
              .head(10)
              .index)

# Filtering the data to include only the top 10 occupations and top 10 skills
filtered_data = df_heatmap_data[df_heatmap_data['Title'].isin(top_occupations_titles) &
                                df_heatmap_data['Element Name'].isin(top_skills)]

heatmap_data = filtered_data.pivot(index='Title', columns='Element Name', values='Data Value')

# Normalizing the data values for better color mapping
heatmap_data_normalized = heatmap_data.apply(lambda x: (x - x.min()) / (x.max() - x.min()), axis=1)

# Generating and visualizing the heatmap with titles:
plt.figure(figsize=(12, 10))  # Adjust the figure size as necessary
sns.heatmap(heatmap_data_normalized, cmap='coolwarm', annot=True, fmt=".2f", linewidths=.5)
plt.title('Heatmap of Top 10 Skills Importance Across Top 10 Occupations')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()  # Adjust the plot to ensure everything fits without overlapping
plt.show()
```

Combining different types of data like skills and abilities for a unified analysis can be challenging. Concatenating and grouping the data helps standardize and simplify the analysis.

Managing and visualizing a potentially large and complex dataset is another hurdle, which can be addressed by focusing on the top 10 skills and occupations to limit the scope and improve clarity.

Ensuring the visualization accurately reflects differences in skill importance across occupations is crucial, and normalizing the data within each row makes relative comparisons meaningful and visually discernible.

Similarly, for 10 Top Abilities v/s 10 Top Occupations

```
#Visualizing heatmap for top 10 Abilities with respect to top 10 the occupations

abilities_grouped = abilities.groupby(['Title', 'Element Name']).agg({'Data Value': 'mean'}).reset_index()

# Identifying top 10 occupations with the highest average 'Data Value'
top_occupations_titles = (abilities_grouped.groupby('Title')['Data Value']
                          .mean()
                          .sort_values(ascending=False)
                          .head(10)
                          .index)

# Identifying top 10 abilities with the highest average 'Data Value'
top_abilities = (abilities_grouped.groupby('Element Name')['Data Value']
                 .mean()
                 .sort_values(ascending=False)
                 .head(10)
                 .index)

# Filtering the data to include only the top 10 occupations and top 10 abilities
filtered_abilities = abilities_grouped[abilities_grouped['Title'].isin(top_occupations_titles) &
                                       abilities_grouped['Element Name'].isin(top_abilities)]

abilities_heatmap_data = filtered_abilities.pivot(index='Title', columns='Element Name', values='Data Value')

# Normalizing the data values for better color mapping
abilities_heatmap_data_normalized = abilities_heatmap_data.apply(lambda x: (x - x.min()) / (x.max() - x.min()), axis=1)

# Generating the heatmap
plt.figure(figsize=(12, 10))  # Adjust the figure size as necessary
sns.heatmap(abilities_heatmap_data_normalized, cmap='coolwarm', annot=True, fmt=".2f", linewidths=.5)
plt.title('Heatmap of Top 10 Abilities Importance Across Top 10 Occupations')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()  # Adjust the plot to ensure everything fits without overlapping
plt.show()
```

And, for 10 Top Tech_Skills v/s 10 Top Occupations

```
# Visualizing heatmap for top 10 Tech skills with respect to top 10 occupations:

# Merging the title from the tech_skills DataFrame directly
tech_skills_with_title = tech_skills.copy()

# Verifying the merge by checking the 'Title' column's presence
print(tech_skills_with_title.head())

# Finding the top 10 occupations based on how many different tech skills are listed
top_occupations_by_tech_skills = (tech_skills_with_title.groupby('Title')
                                  .size()
                                  .sort_values(ascending=False)
                                  .head(10)
                                  .index)

# Finding the top 10 tech skills based on how often they appear across different occupations
top_tech_skills = (tech_skills_with_title.groupby('Example')
                   .size()
                   .sort_values(ascending=False)
                   .head(10)
                   .index)

# Filtering the dataset
filtered_tech_skills = tech_skills_with_title[tech_skills_with_title['Title'].isin(top_occupations_by_tech_skills) &
                                              tech_skills_with_title['Example'].isin(top_tech_skills)]

tech_skills_heatmap_data = (filtered_tech_skills.pivot_table(index='Title',
                                                             columns='Example',
                                                             aggfunc='size',
                                                             fill_value=0))

# Generating the heatmap for tech skills
plt.figure(figsize=(12, 8))
sns.heatmap(tech_skills_heatmap_data, cmap='viridis', annot=True, linewidths=.5)
plt.title('Heatmap of Top 10 Tech Skills Across Top 10 Occupations')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```

k. Machine Learning Models

The below code shown defines machine learning pipelines for four different algorithms: Logistic Regression, Random Forest, Support Vector Machine (SVM), and Gradient Boosting. Each pipeline includes data preprocessing steps and the respective classifier.

After training each model on the training data and making predictions on the test data, it evaluates the performance using accuracy metrics and classification reports. (Shown in ***Project Results*** section).

```
# Dropping the 'Hot_Tech' column since it was used to create the target variable
X = features_combined.drop(columns=['Hot_Tech', 'Target_Growth'])
y = features_combined['Target_Growth']
```

```
#defining training and testing variables
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# defining ML Model pipelines
pipelines = {
    'Logistic Regression': Pipeline([('scaler', StandardScaler()), ('classifier', LogisticRegression(random_state=42))]),
    'Random Forest': Pipeline([('classifier', RandomForestClassifier(random_state=42))]),
    'Support Vector Machine': Pipeline([('scaler', StandardScaler()), ('classifier', SVC(random_state=42))]),
    'Gradient Boosting': Pipeline([('classifier', GradientBoostingClassifier(random_state=42))])
}
```

Logistic Regression

```
# Training, predicting, and evaluating Logistic Regression
logistic_pipeline = pipelines['Logistic Regression']
logistic_pipeline.fit(X_train, y_train)
y_pred_logistic = logistic_pipeline.predict(X_test)
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
print("Logistic Regression Accuracy:", accuracy_logistic)
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_logistic))
```

Random Forest

```
# Training, predicting, and evaluating Random Forest
rf_pipeline = pipelines['Random Forest']
rf_pipeline.fit(X_train, y_train)
y_pred_rf = rf_pipeline.predict(X_test)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print("Random Forest Accuracy:", accuracy_rf)
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

Support Vector Machine

```
# Training, predicting, and evaluating Support Vector Machine
svm_pipeline = pipelines['Support Vector Machine']
svm_pipeline.fit(X_train, y_train)
y_pred_svm = svm_pipeline.predict(X_test)
accuracy_svm = accuracy_score(y_test, y_pred_svm)
print("Support Vector Machine Accuracy:", accuracy_svm)
print("Support Vector Machine Classification Report:")
print(classification_report(y_test, y_pred_svm))
```

Gradient Boosting

```
# Training, predicting, and evaluating Gradient Boosting
gb_pipeline = pipelines['Gradient Boosting']
gb_pipeline.fit(X_train, y_train)
y_pred_gb = gb_pipeline.predict(X_test)
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print("Gradient Boosting Accuracy:", accuracy_gb)
print("Gradient Boosting Classification Report:")
print(classification_report(y_test, y_pred_gb))
```

One of the challenges was deciding which machine learning models to include for comparison. We needed models suitable for classification tasks and capable of handling the dataset's complexity.

Another challenge was the pipeline construction, which involved preprocessing steps such as scaling and fitting classifiers. Ensuring compatibility between preprocessing steps and classifiers posed another challenge.

We opted for Logistic Regression, Random Forest, SVM, and Gradient Boosting as they offer a diverse range of approaches suitable for comparison. Logistic Regression provides simplicity, Random Forest offers robustness to noise, SVM is effective in high-dimensional spaces, and Gradient Boosting can handle complex relationships in the data.

And, regarding the pipeline construction, we addressed compatibility issues by designing pipelines that integrate preprocessing steps with classifiers. For instance, scaling was applied before classifiers in pipelines where it was necessary for algorithm performance.

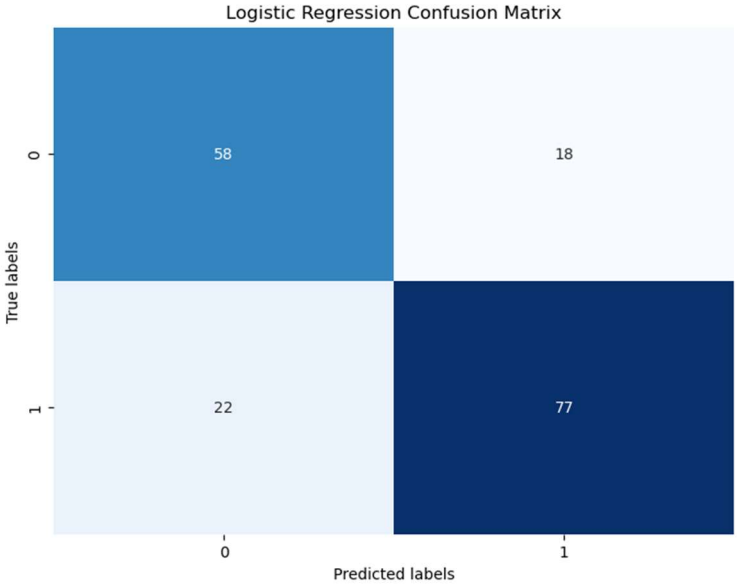
- Visualizations

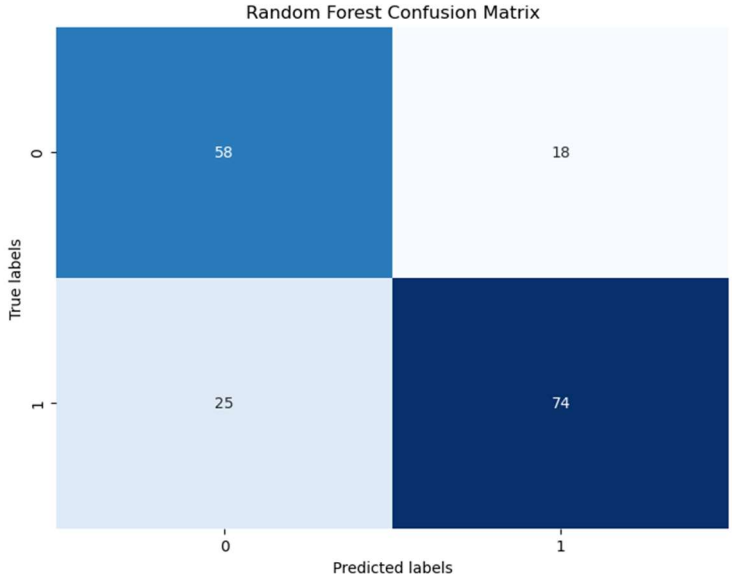
In visualization, we use confusion matrices for these models. For which, this function, plot_confusion_matrix, facilitates the calculation of a confusion matrix (using func. confusion matrix()), which then visualizes it as a heatmap using seaborn's (as sns) heatmap function.

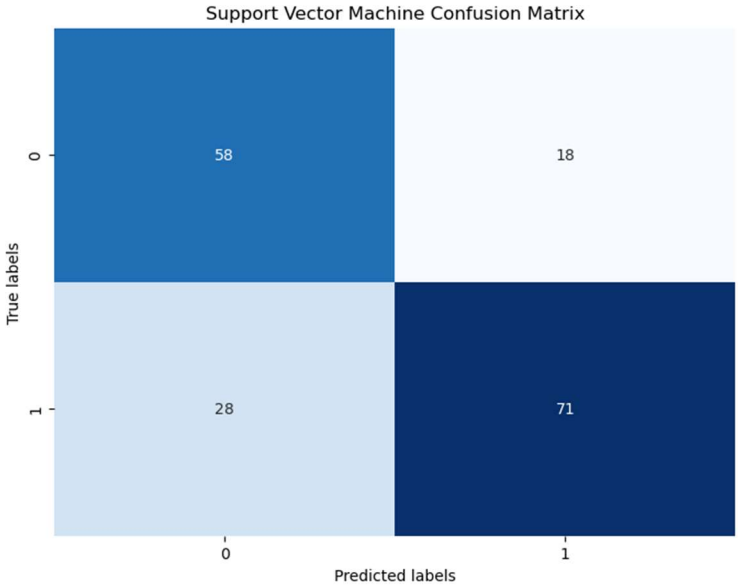
The heatmap's color intensity represents the number of samples in each cell of the confusion matrix, with annotations displaying the actual values. The plot includes axis labels indicating the true and predicted labels.

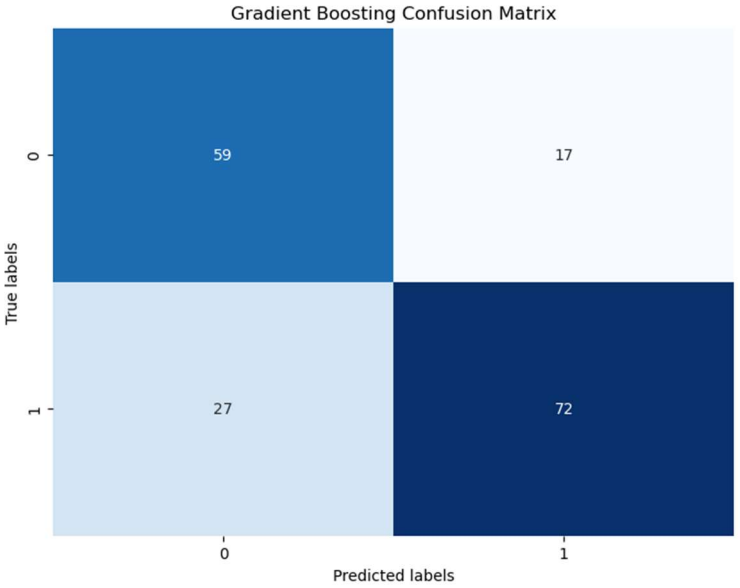
Confusion Matrices

```
# Function to plot confusion matrix
def plot_confusion_matrix(y_true, y_pred, title):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", cbar=False)
    plt.title(title)
    plt.xlabel("Predicted labels")
    plt.ylabel("True labels")
    plt.show()
```









l. Data Processing & Aggregate for Skills & Abilities for Visualizing – HeatMap4

The below code shown, where we define a function preprocess_and_aggregate() to preprocess and aggregate data while retaining occupation titles. It filters data based on an importance scale, aggregates important values per occupation, and pivots the data for analysis.

This function is applied separately to skills and abilities data. Additionally, tech skills data is processed, treating 'Hot Technology' as a binary feature and aggregated at the occupation level.

The resulting feature sets for skills and abilities are combined, and missing values are filled with zeros. This comprehensive feature set, final_features_skills_abilities, is printed for examination, along with a separate feature set for technology skills, final_features_tech.

These feature sets provide valuable insights for subsequent analysis.

```
# Function to preprocess and aggregate data while retaining occupation titles
def preprocess_and_aggregate(df, value_col, importance_filter):
    # Filtering based on the importance scale
    filtered_df = df[df['Scale ID'] == importance_filter]

    # Aggregating the important values by mean per occupation, keep Title for reference
    aggregated_df = filtered_df.groupby(['O*NET-SOC Code', 'Title', 'Element Name'])[value_col].mean().reset_index()

    # Pivoting data to wide format
    pivot_df = aggregated_df.pivot_table(index=['O*NET-SOC Code', 'Title'], columns='Element Name', values=value_col).fillna(0)

    return pivot_df
```

```
# Applying preprocessing to skills and abilities
skills_features = preprocess_and_aggregate(skills_df, 'Data Value', 'IM')
abilities_features = preprocess_and_aggregate(abilities_df, 'Data Value', 'IM')

# Combining skills and abilities
combined_features = skills_features.join(abilities_features, how='outer', rsuffix='_abil').fillna(0)

# Preprocessing tech skills - assume 'Hot Technology' as a binary feature (1 if 'Y', else 0)
tech_skills_df['Hot Technology'] = tech_skills_df['Hot Technology'].map({'Y': 1, 'N': 0})
tech_aggregated = tech_skills_df.groupby(['O*NET-SOC Code', 'Title'])['Hot Technology'].sum()

# Final feature set by combining all features
final_features = combined_features.join(tech_aggregated, how='outer').fillna(0)

print(final_features.head())
```

The above code then selects a subset of skills and abilities, including 'Reading Comprehension', 'Active Listening', 'Writing', and 'Speaking', from the final feature set final_features.

This subset is then visualized as a heatmap using seaborn heatmap function. The heatmap displays the importance of each selected skill and ability across the top 20 occupations, with annotations representing the “importance” values.

The y-axis indicates the occupation code and title, while the x-axis represents the selected skills and abilities.

This visualization (shown in the ***Project Results*** section) provides insights into the relative importance of these skills and abilities across different occupations, aiding in understanding occupational requirements and potential skill gaps.

Output Snippet: final_features.head()

O*NET-SOC Code	Title	Active Learning	Active Listening	Complex Problem Solving	Coordination	Critical Thinking	Written Comprehension	Written Expression	Hot Technology
11-1011.00	Chief Executives	3.75	4.00	4.38	4.25	4.38	4.25	4.12	17
11-1011.03	Chief Sustainability Officers	3.75	4.00	4.00	3.75	4.12	4.00	4.12	14
11-1021.00	General and Operations Managers	3.62	4.00	3.62	3.88	3.88	4.00	4.00	50
11-1031.00	Legislators	0.00	0.00	0.00	0.00	0.00	0.00	0.00	11
11-2011.00	Advertising and Promotions Managers	3.25	4.12	3.50	3.50	4.00	4.00	3.88	30

[5 rows x 88 columns]

4. Project Results

Observations from data visualization and network analysis provide valuable insights into our networks:

Network Structure: The visualizations effectively illustrated the complexity and density of the occupational networks. This structure helped identify which occupations share similar skill sets, abilities, or technology uses, indicating potential pathways for career transitions or developments.

Node Importance: Using degree centrality measures, the project identified key occupations and skills that act as hubs within their respective networks. For instance, certain skills like management of material resources and programming showed high centrality in the skills network.

Network Connectivity: The connectivity within the networks varied, with some occupations showing strong interconnections through common skills or technologies_skills, suggesting these occupations are versatile in terms of skill application.

Community Structure: The detection of communities within the graphs revealed clusters of occupations that share a tight-knit set of skills or abilities.

Node Attributes: Attributes associated with nodes, such as the type of skills or technological_skills, were crucial in understanding the function and role of each occupation within the industry.

Community Detection: Applied community detection algorithms highlighted distinct groups within the networks, which could represent specialized fields or sectors within the broader industry.

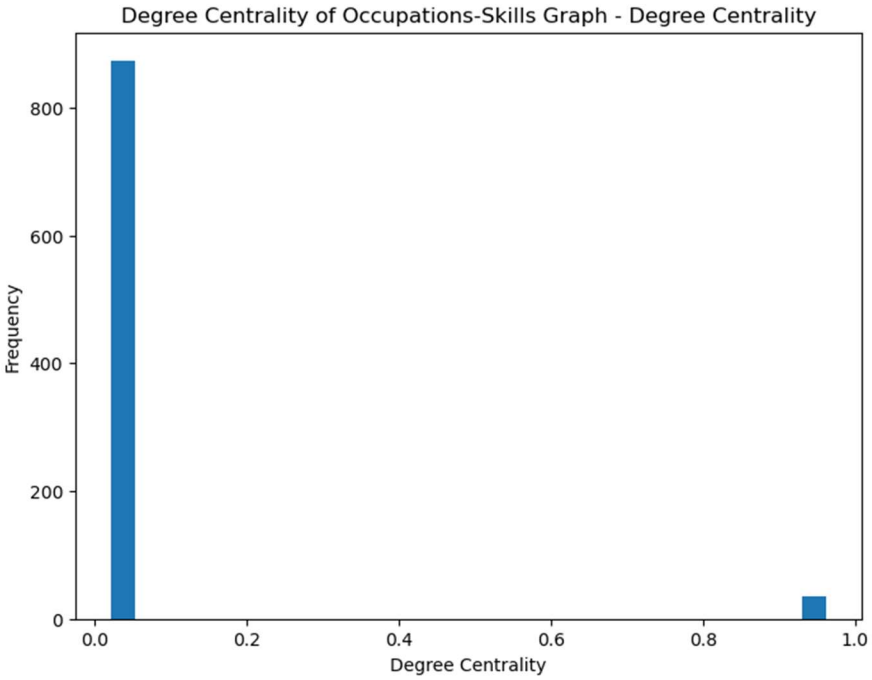
Influencers: Identification of influencer nodes within each graph was done using a projection technique on non-occupation nodes, revealing the most influential skills, abilities, and technologies.

For example, in the technology_skills graph, the ability to use specific software like Microsoft Excel and Microsoft Word demonstrated high centrality, indicating their critical role across many occupations.

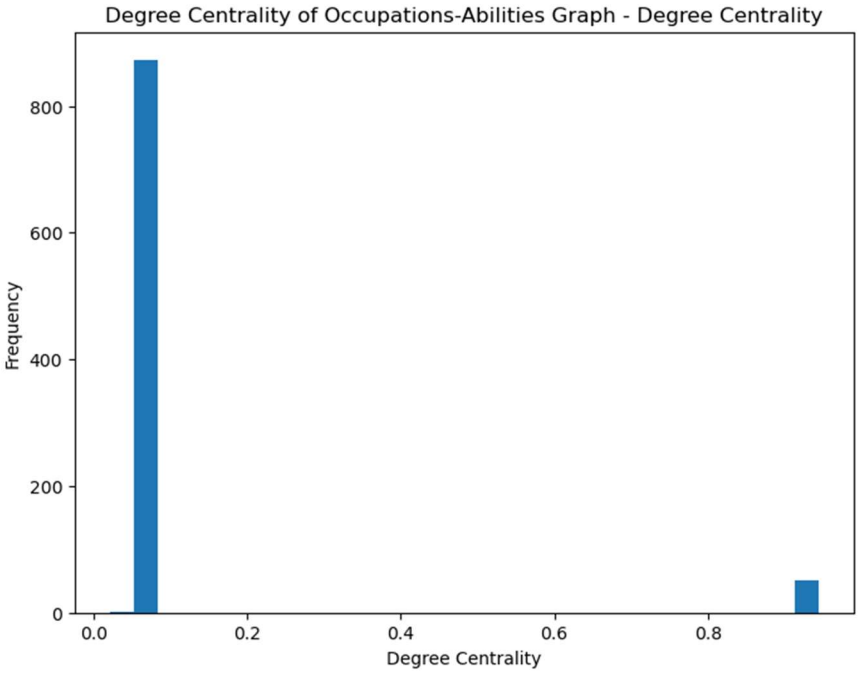
Network Measures :-

Degree Centrality

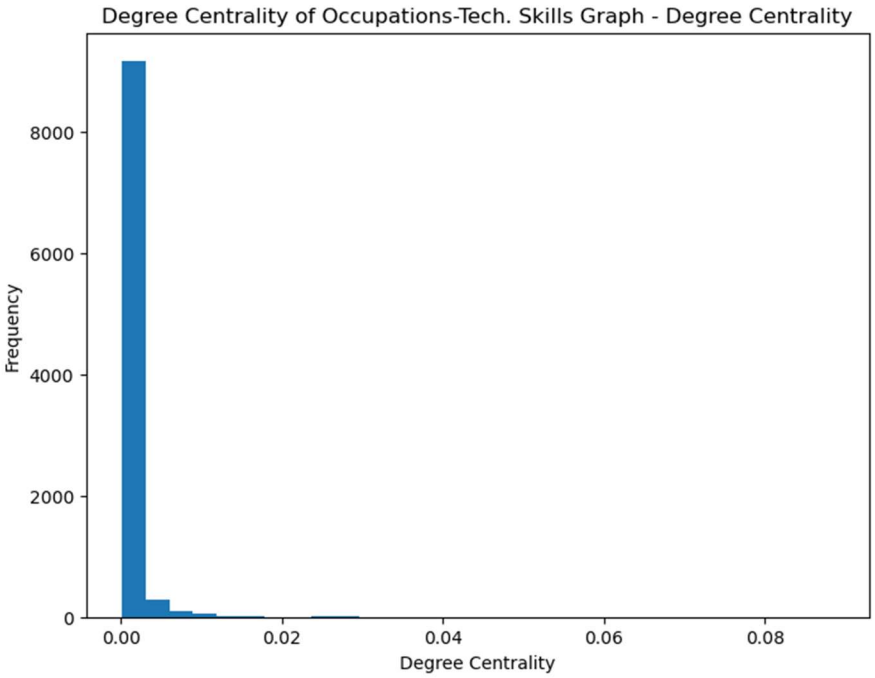
Skills Graph (Avg = 0.0742): Indicates moderate connectivity among skills, suggesting diverse linkages across occupations.



Abilities Graph (Avg = 0.1062): Higher connectivity among abilities shows they are more universally applicable across various occupations.

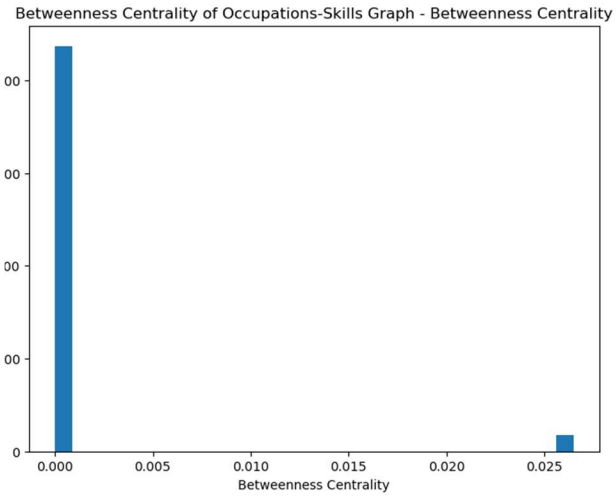


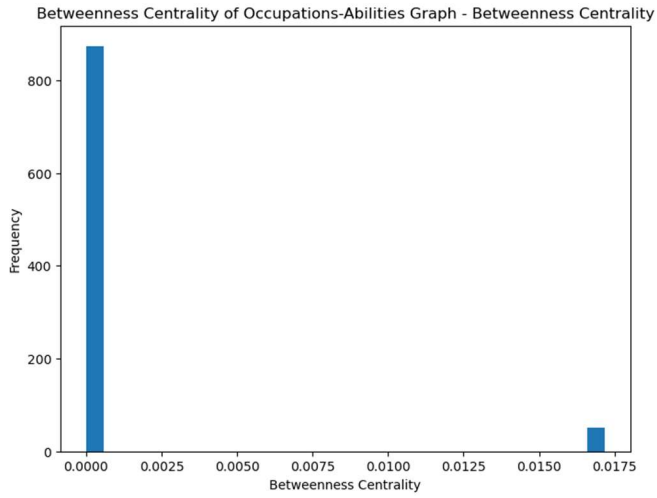
Technology Skills Graph (Avg = 0.000694): Very low connectivity, reflecting the specialized nature of technology skills unique to specific jobs.



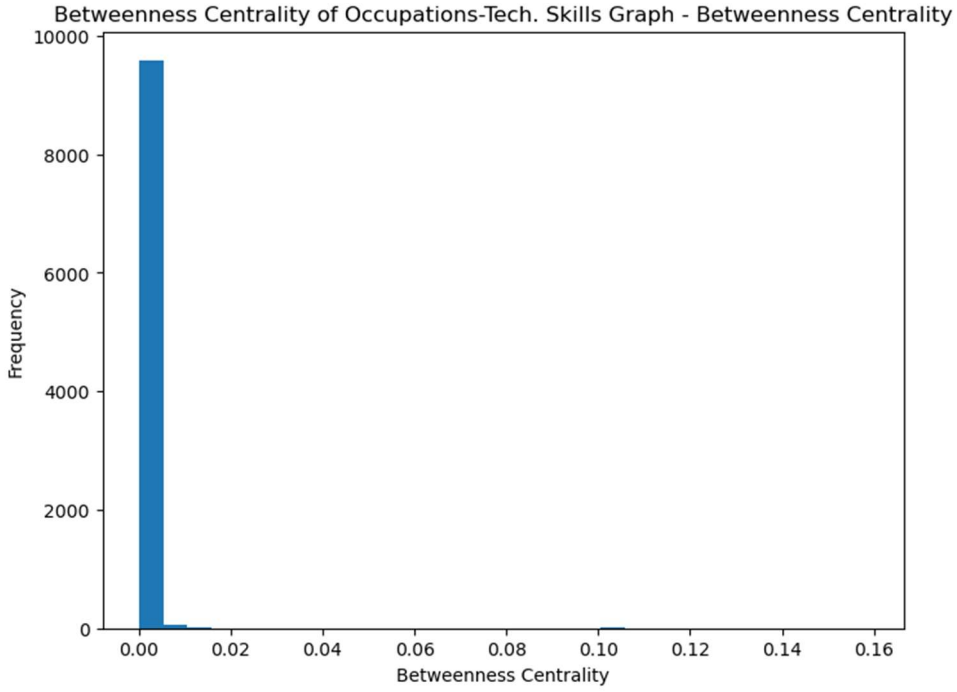
Betweenness Centrality

Skills Graph (Avg = 0.00102) and Abilities Graph (Avg = 0.000968): Both show low betweenness, indicating no single skill or ability dominates as a central or controlling bridge within their networks.



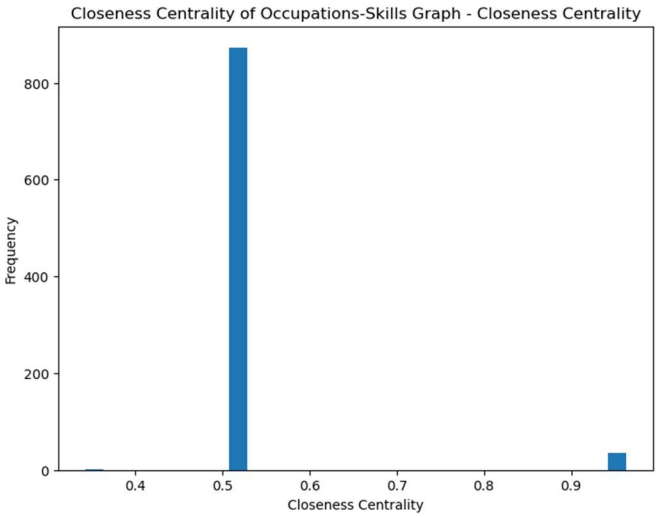


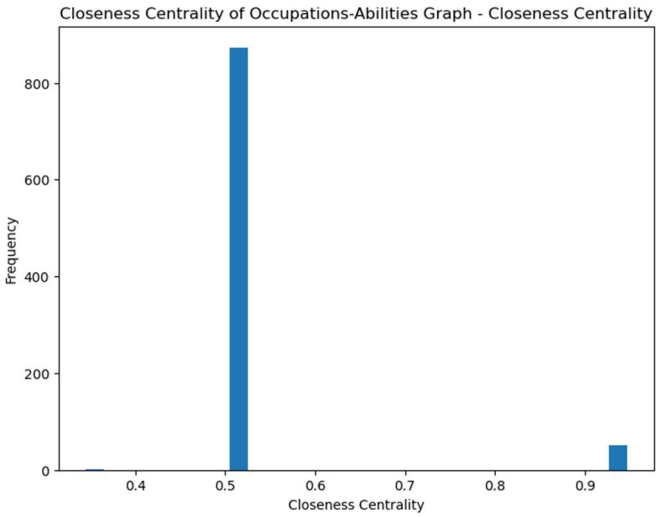
Technology Skills Graph (Avg = 0.000283): Extremely low betweenness suggests technology skills rarely connect disparate parts of the network, indicating their specialized application.



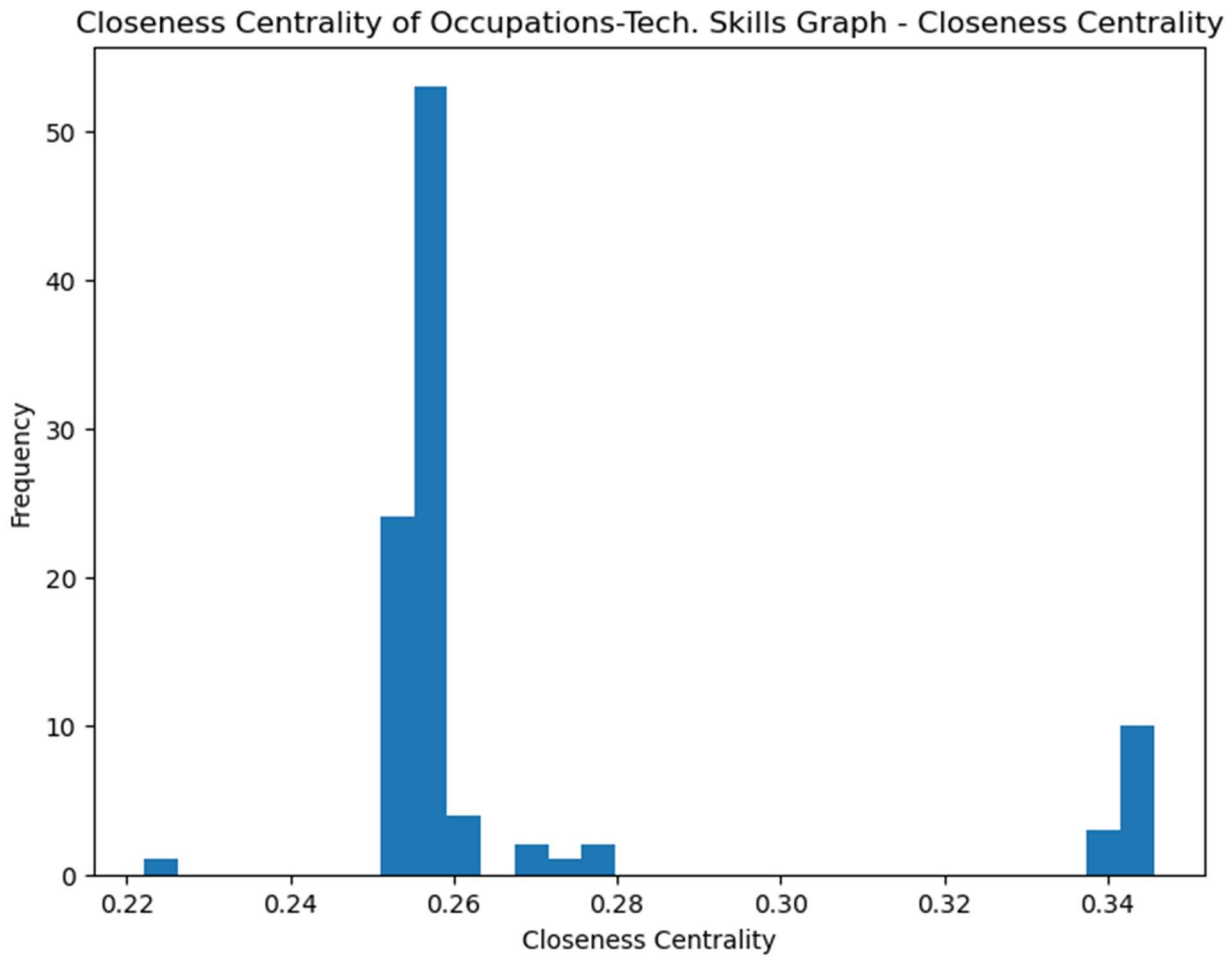
Closeness Centrality

Skills Graph (Avg = 0.5273) and Abilities Graph (Avg = 0.5388): Moderate to high values suggest skills and abilities can quickly influence or be influenced by changes across their networks.



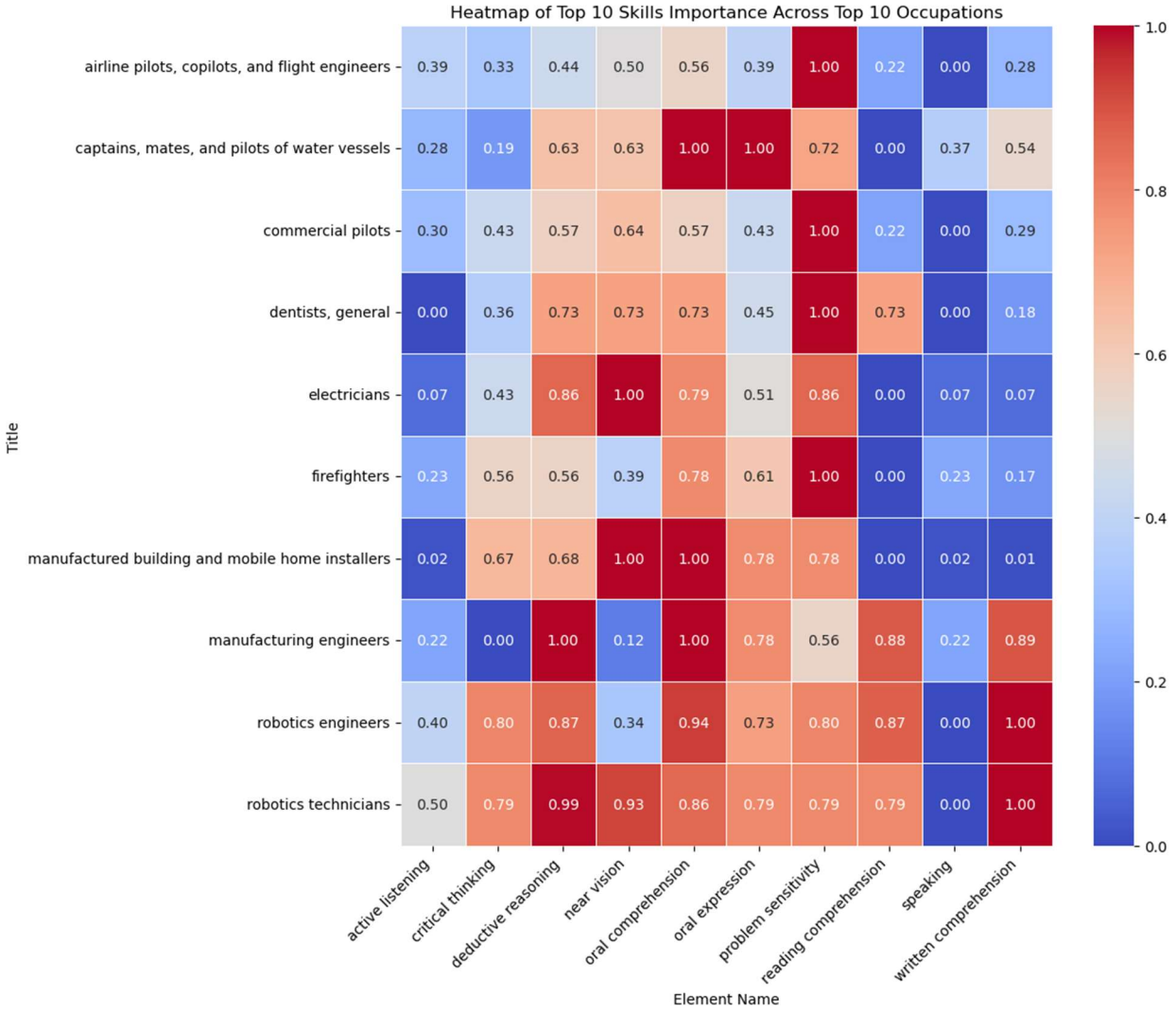


Technology Skills Graph (Avg = 0.2678): Low closeness centrality indicates technology skills are isolated and interact less frequently with other skills in the network.



Overall, Abilities are generally more interconnected and versatile across occupations compared to skills. Technology_skills, in contrast, are highly specialized with limited cross-occupational applicability.

Heatmap1 - Importance of the Top 10 Skills across the Top 10 occupations.



The following heatmap helped us in identifying the skills that are crucial within specific occupations and reveals patterns that might be pivotal. For example,

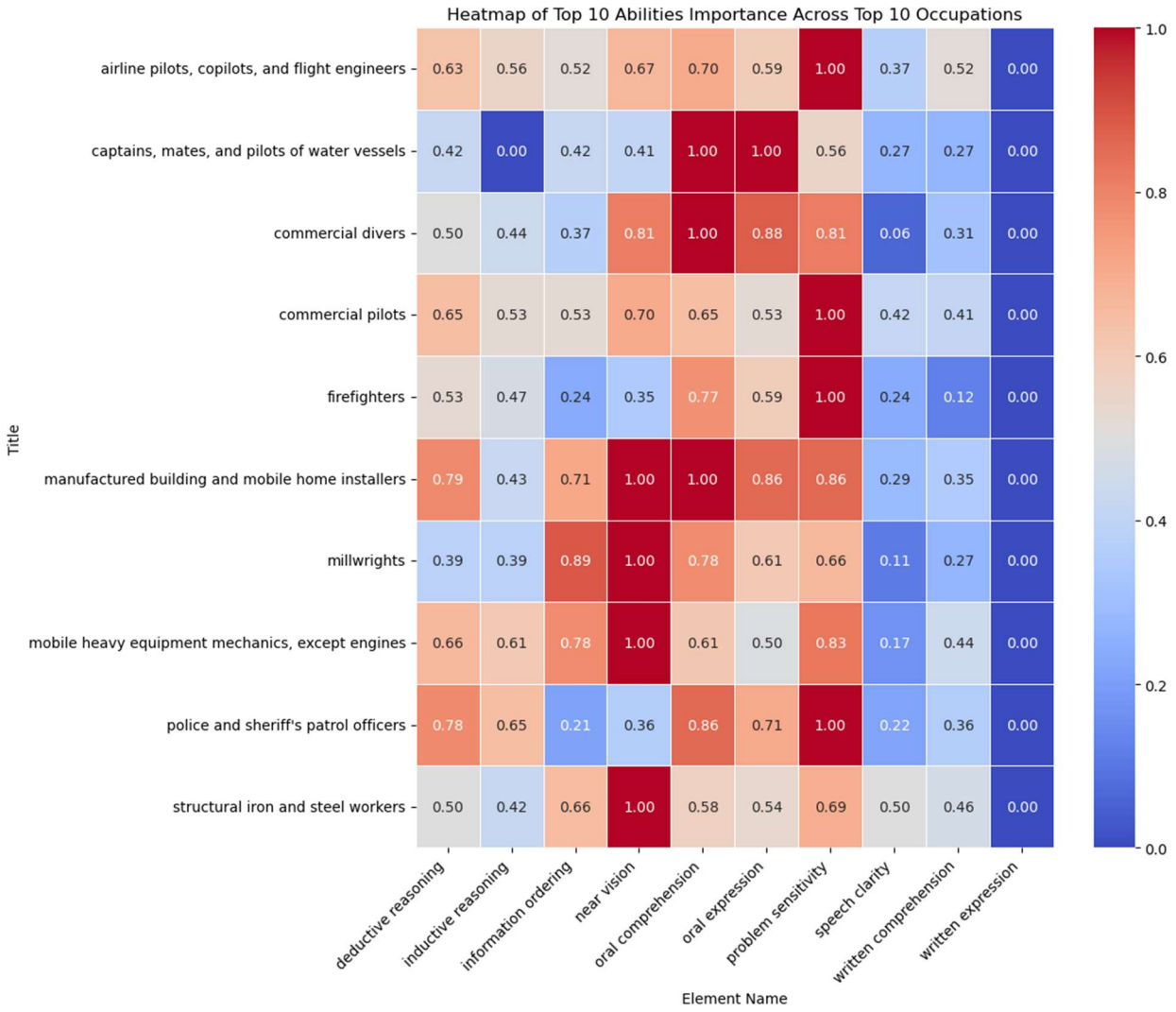
We observed High Importance Skills across Multiple Occupations, like Critical Thinking and Problem Sensitivity, these skills show a uniformly high importance across several occupations like electricians, manufacturing engineers, and robotics engineers and technicians. This suggests that analytical skills are highly valued across technical and engineering fields, indicating a common requirement for problem-solving abilities in these areas.

Similarly, for Near Vision and Oral Comprehension which is particularly crucial for occupations such as dentists and robotics technicians, reflecting the need for precision and effective communication in roles that involve intricate tasks or complex operational settings.

Additionally, for Manufactured Building and Mobile Home Installers these occupations show a significant reliance on near vision and problem sensitivity, which are critical in occupations involving physical construction and detailed assembly work.

Moreover, for Firefighters they display a broader skillset necessity, including active listening, critical thinking, and oral comprehension, showcasing the diverse skills required in emergency response and safety professions.

Heatmap2 - Importance of the Top 10 Abilities across the Top 10 Occupations.



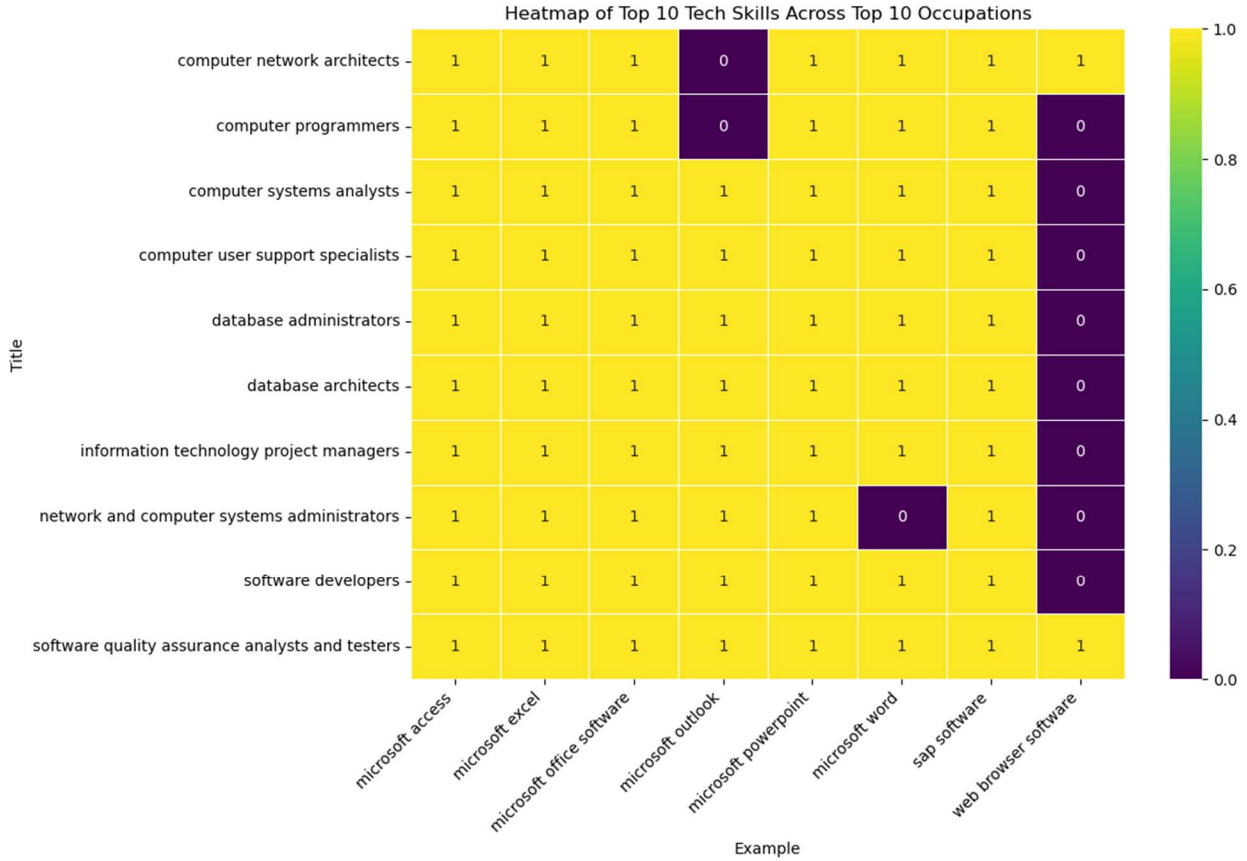
This heatmap offered us a detailed look at which abilities are critical in various professional roles. By examining how these abilities are rated across different jobs, we can gain insights into the core competencies that are most valued in specific fields.

For example, we see Key Abilities Across Multiple Occupations, like Near Vision and Problem Sensitivity, where these abilities score highly across several occupations, including manufactured building and mobile home installers and mobile heavy equipment mechanics, which suggest us that precise visual skills and the ability to anticipate and solve problems on the spot are crucial in roles that require hands-on, technical work.

Moreover the other relation we noticed was of Occupation-Specific Abilities, where Deductive and Inductive Reasoning were highly valued in structured and detail-oriented roles such as structural iron and steel workers, and millwrights, where logical reasoning is necessary to solve complex problems.

Along with, Information Ordering which is particularly important for commercial pilots and commercial divers, reflecting the need to sequence tasks and information logically during operations.

Heatmap3 – Technology_Skills across 10 Key IT-related Occupations



This heatmap was helpful in identifying which technology skills are most universally required across these roles and which skills are unique to certain positions. Such as,

In Universal Tech Skills, Skills such as Microsoft Office software (including Excel, PowerPoint, and Word) are shown to be universally required across almost all the analyzed IT roles. This tells us the fundamental nature of these tools in today's IT jobs, underlining their importance in everyday computational tasks and data management.

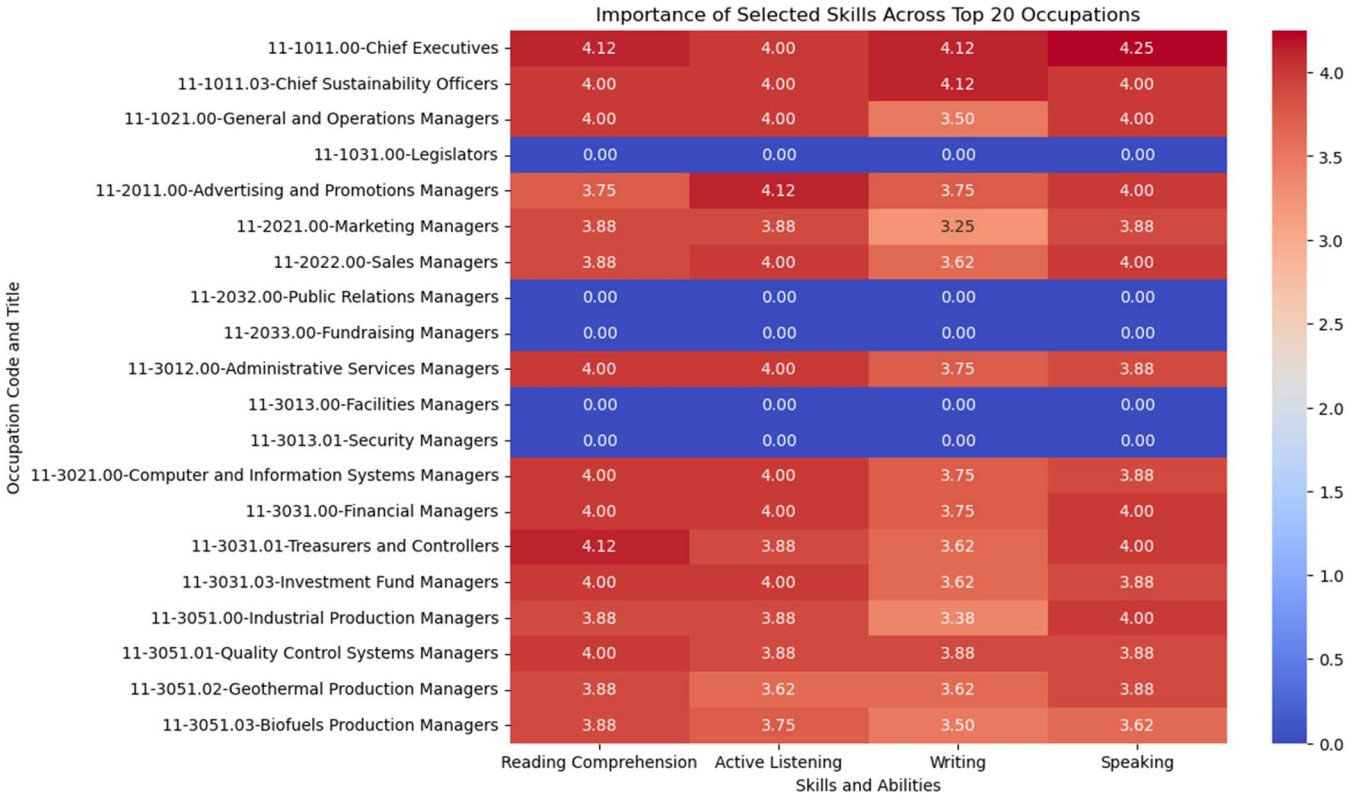
Similarly, Microsoft Access and Outlook also appear frequently but with slightly less universality, suggesting that while important, they may be more critical in specific contexts or roles that require more in-depth data manipulation or communication management.

Moving on to Specialized Tech Skills, we see that SAP software and Web browser software show selective importance, being crucial in certain roles but absent in others.

For example, roles like software developers and software quality assurance analysts do not rely on SAP, which is typically more relevant in roles focused on enterprise resource planning or data systems management.

The absence of some skills in certain roles (marked as 0 in the heatmap) like network and computer systems administrators not needing SAP software highlights the specialized needs and focus of certain IT positions.

Heatmap4 – Selected Skills & Abilities across Range of Occupations



This heatmap showcases the importance of selected skills—specifically reading comprehension, active listening, writing, and speaking—across a range of managerial and executive occupations. Where the intensity of the colors indicates the relative importance of these skills, ranging from 0 (least important) to 4 (most important).

We see that the roles of Chief Executives and Financial Managers show high ratings across all 4 skills, particularly in speaking and active listening which indicates the critical need for excellent communication skills in leadership and financial management, where decision-making and stakeholder interaction are frequent.

Even the Marketing and Sales Managers both emphasize high competency in speaking and active listening, which are essential for effective persuasion, negotiation, and customer engagement.

We can observe that the Administrative and Facilities Managers display a high importance for active listening and speaking but less so for reading comprehension and writing, which tells us the operational nature of their roles which requires more direct communication and less focus on written content.

Machine Learning Models - Accuracy

We implemented 4 different models for project. Namely, Logistic Regression, Random Forest, Support Vector Machine (SVM), and Gradient Boosting. The results we obtained we as follows:

In logistic regression standard scaling was applied to features where it achieved the highest accuracy among the models at approximately 77.14%.

```
Logistic Regression Accuracy: 0.7714285714285715
Logistic Regression Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.76      0.74        76
           1       0.81      0.78      0.79        99

    accuracy                           0.77       175
   macro avg       0.77      0.77      0.77       175
weighted avg       0.77      0.77      0.77       175
```

Where as in Random Forest it was implemented without scaling, given its invariance to the scale of features, with an accuracy of about 75.43%.

```
Random Forest Accuracy: 0.7542857142857143
Random Forest Classification Report:
              precision    recall  f1-score   support

           0       0.70      0.76      0.73        76
           1       0.80      0.75      0.77        99

    accuracy                           0.75       175
   macro avg       0.75      0.76      0.75       175
weighted avg       0.76      0.75      0.76       175
```

Next, Was the Support Vector Machine (SVM) which was similar to logistic regression, SVM used standard scaling and achieved an accuracy of approximately 73.71%.

```
Support Vector Machine Accuracy: 0.7371428571428571
Support Vector Machine Classification Report:
              precision    recall  f1-score   support

           0       0.67      0.76      0.72        76
           1       0.80      0.72      0.76        99

    accuracy                           0.74       175
   macro avg       0.74      0.74      0.74       175
weighted avg       0.74      0.74      0.74       175
```

Finally, it was Gradient Boosting yet another tree-based model like Random Forest, not requiring feature scaling, with an accuracy of about 74.86%.

```
Gradient Boosting Accuracy: 0.7485714285714286
Gradient Boosting Classification Report:
              precision    recall  f1-score   support

           0       0.69      0.78      0.73        76
           1       0.81      0.73      0.77        99

    accuracy                           0.75       175
   macro avg       0.75      0.75      0.75       175
weighted avg       0.76      0.75      0.75       175
```

From these results we are able to say that logistic regression offers a good balance of accuracy and model interpretability, which is useful for understanding feature importance.

The varying performances show differences in how each model handles the features, particularly the binary 'Hot_Tech' values and the combined values from skills and abilities.

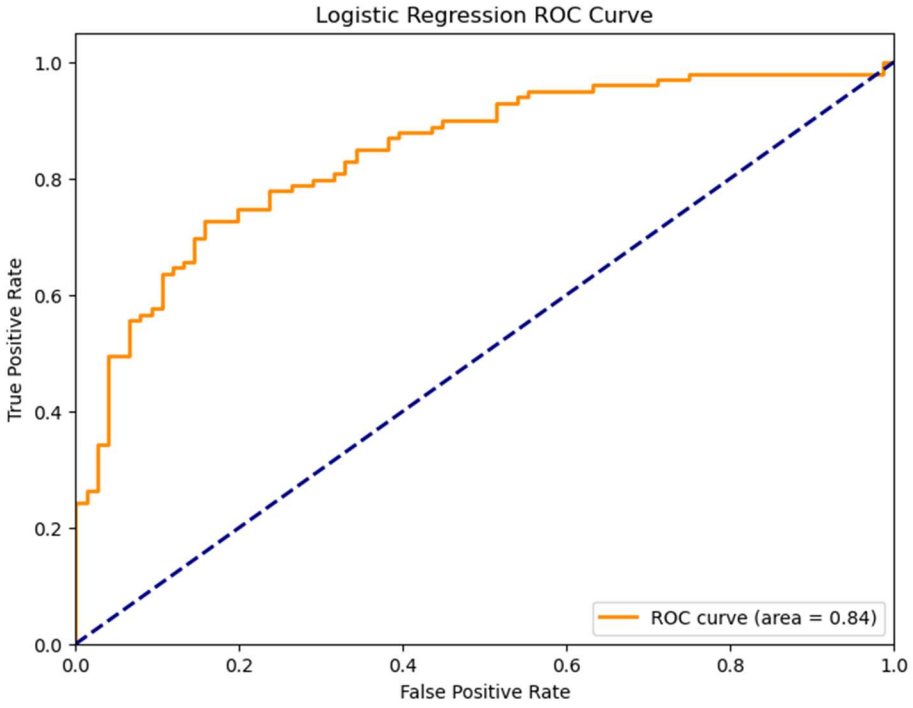
However, from this we know that these models could be used to identify occupations that are likely to require increasing levels of technology-related skill.

Machine Learning Models – ROC Curves

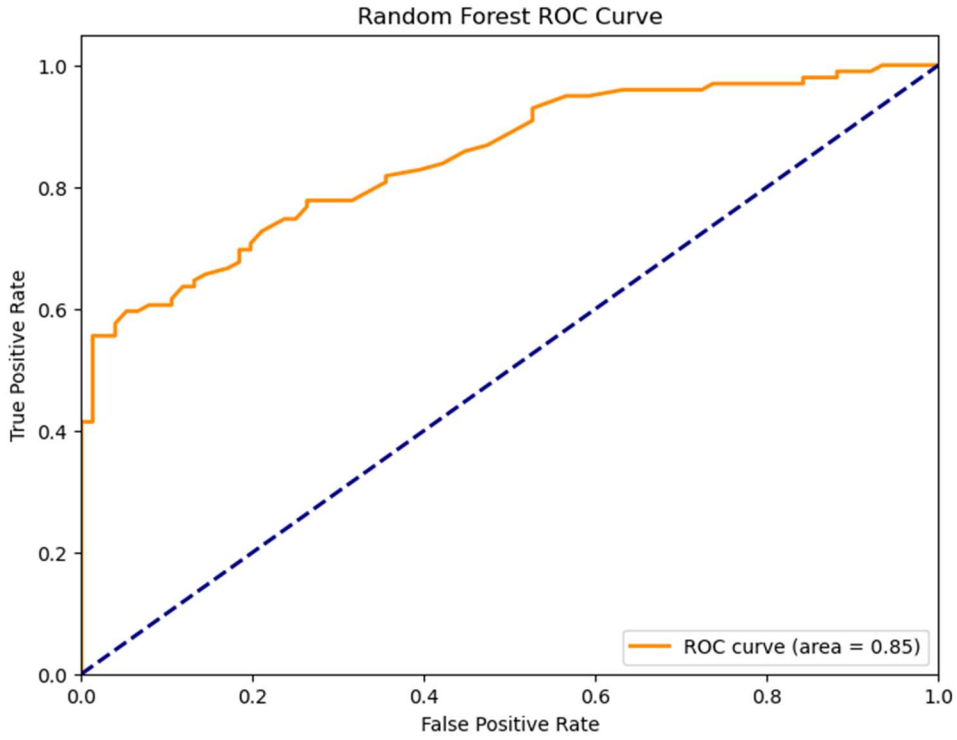
The ROC curves figure from these models, Logistic Regression, Random Forest, Support Vector Machine (SVM), and Gradient Boosting give us a look at their performance. These curves are useful for evaluating the diagnostic ability of each model by plotting the True Positive(TP) rate (sensitivity) against the False Positive(FP) rate (1-specificity) at various threshold settings.

Here's what we observed from these curves, are as follows:

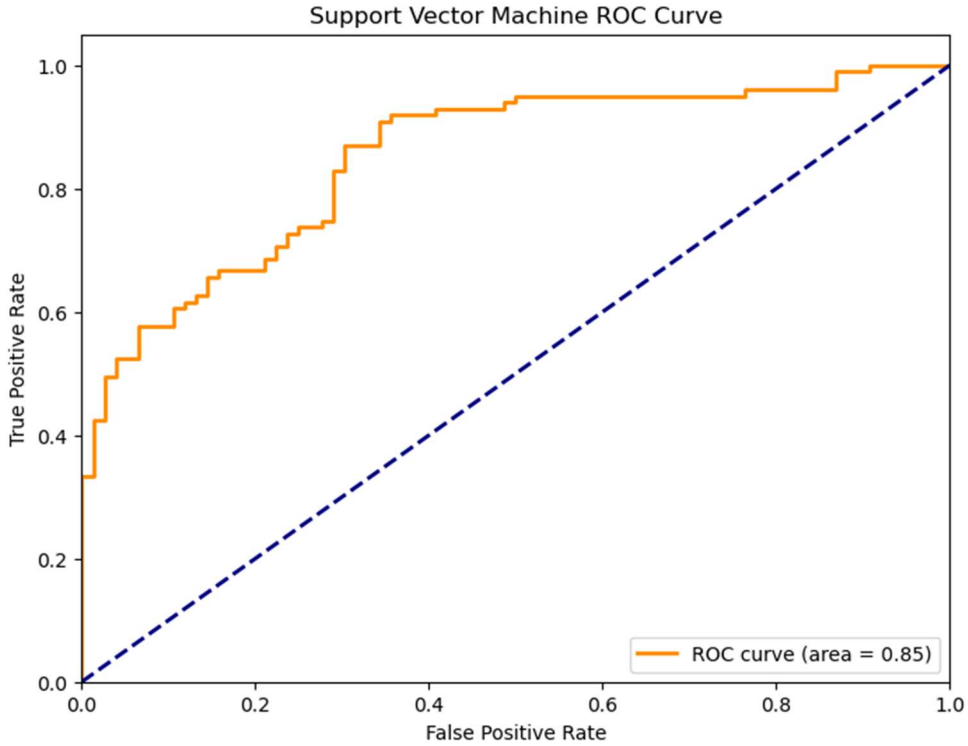
In Logistic Regression, the curve is fairly smooth and close to the top-left corner of the plot, indicating good model performance. It has an AUC (Area Under the Curve) of 0.84. Which means that the LR model is quite good at distinguishing between the classes but of course has some room for improvement in reducing false positives or increasing true positives.



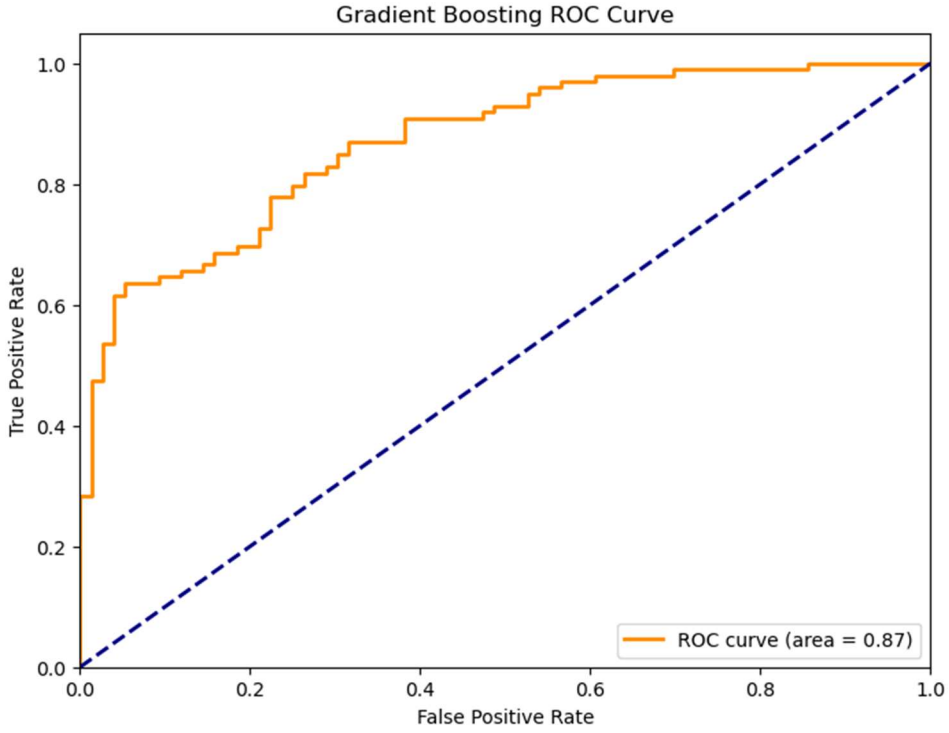
In Random Forest, the curve rises quickly to a high true positive rate and the RF model shows slightly better performance than LR as it has an AUC of 0.85. Which means that the model effectively captures a high proportion of True positives (TP) while keeping the false positives (FP) comparatively low.



In SVM, the curve indicates that the model maintains a consistent balance between sensitivity and specificity across different thresholds. Similar to RF, SVM shows strong performance with an identical AUC value of 0.85.



In Gradient Boosting, the curve closely approaches the top left corner, suggesting that the model has the highest TP rate for the lowest FP rate, making it the most effective at classifying the positive cases correctly. GB also exhibits the best performance among the models with an AUC of 0.87.



5. Discussion of Project

Reflecting on the completion of the project, we believe it went quite well in terms of achieving the goals we set out to understand the complex interrelationships between occupations through skills, abilities, and technology. The use of bipartite graphs, community detection and centrality measures to elucidate the complex relationships within occupational networks.

Community detection algorithms allowed us to uncover clusters of occupations that shared similar skills, abilities, or technological skills, highlighting potential pathways for career transitions or developments. This approach was crucial in understanding how different job roles interconnect and which skill sets are transferable across sectors. The identification of key influencers within these communities, through centrality analysis, further enhanced our insights by pinpointing which skills or abilities act as pivotal nodes, influencing multiple occupations.

For future projects, we would like to streamline the data preprocessing and integration process to enhance efficiency and accuracy, as the overall code execution took reasonable amount of time to execute, an attempt at minimizing that time would be an important factor. Additionally, we would like to explore various community detection algorithms that could offer more nuanced insights into occupational clustering.

The project taught us the importance of effective graphical representations and data analysis in understanding complex networks and information. Through network metrics such as centrality and community detection, significant learnings were gained about the structure and influential factors within the occupational networks.

6. Future Work

Given more time, expanding the scope could include the incorporating real-time labor market data to capture emerging trends, along with unsupervised machine learning modeling like CNNs, RCNN, MLP, etc.… on the future requirements of emerging skills and roles could be immensely beneficial.

Additionally, Extending the analysis to include demographic and geographical data to understand regional labor market dynamics better. Moreover, implementing sentiment analysis on employer reviews and feedback from platforms like LinkedIn and Glassdoor to gauge employer satisfaction with the current workforce's skills and adaptability.

Another added approach could include linking labor market data with broader economic indicators such as GDP growth rates, unemployment rates, and industry growth statistics to analyze the correlation between economic performance and labor market trends. Each of these areas could be explored and understood which could significantly enhance the depth and utility of the project in the future with such enhancements.

REFERENCES

1. **Pandas**: Package URL: [Pandas Documentation]
    - McKinney, W. (2010). Data Structures for Statistical Computing in Python. In Proceedings of the 9th Python in Science Conference (pp. 56–61).
2. **NetworkX**: Package URL: [NetworkX Documentation]
   Stable Documentation URL: (https://networkx.org/documentation/stable/)
3. **Matplotlib**: Package URL: [Matplotlib Documentation]
   Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95.
4. **NumPy**: Package URL: [NumPy Documentation]
5. **Python Standard Library**: Python Software Foundation. (n.d.).
    - Python Standard Library Documentation
6. Hagberg, A. A., Schult, D. A., & Swart, P. J. (2008). Exploring network structure, dynamics, and function using NetworkX. In Proceedings of the 7th Python in Science Conference (pp. 11–15).
7. O*NET Resource Center, National Center for O*NET Development, www.onetcenter.org/. From "O*NET Resource Center" by the U.S. Department of Labor, Employment and Training Administration (USDOL/ETA). Used under the CC BY 4.0 license.
8. "**O*NET 28.2 Database**." O*NET Resource Center, National Center for O*NET Development, www.onetcenter.org/database.html.
9. "**Abilities** - O*NET 28.2 Data Dictionary." O*NET Resource Center, National Center for O*NET Development, www.onetcenter.org/dictionary/28.2/excel/abilities.html.
10. "**Skills** - O*NET 28.2 Data Dictionary." O*NET Resource Center, National Center for O*NET Development, www.onetcenter.org/dictionary/28.2/excel/skills.html.
11. "**Technology Skills** - O*NET 28.2 Data." O*NET Resource Center, National Center for O*NET Development, www.onetcenter.org/dictionary/28.2/excel/technology_skills.html.
12. Zafarani, R., Abbasi, M. A., & Liu, H. (2014). Social Media Mining: An Introduction.
13. Borgatti, S. P., & Halgin, D. S. (2011). Analyzing affiliation networks. Sociological Compass, 5(8), 674-691.

14. Newman, M. E. J. (2006). Modularity and community structure in networks. Proceedings of the National Academy of Sciences, 103(23), 8577-8582.
15. Newman, M. E. J. (2004). Fast algorithm for detecting community structure in networks. Physical Review E, 69(6), 066133.
16. Clauset, A., Newman, M. E. J., & Moore, C. (2004). Finding community structure in very large networks. Physical Review E, 70(6), 066111.
17. Kempe, D., Kleinberg, J., & Tardos, É. (2003). Maximizing the spread of influence through a social network. Proceedings of 9th ACM SIGKDD international conference on Knowledge discovery and data mining.
18. Erdos_renyi_graph — NetworkX 3.2.1 documentation
19. Random Model Graph: Social Media Mining - (Ch.4, Pg 110 - 118) - Textbook
20. Watts_strogatz_graph — NetworkX 3.2.1 documentation
21. Small World Graph: Social Media Mining - (Ch.4, Pg 119 - 123) - Textbook
22. Barabasi_albert_graph — NetworkX 3.2.1 documentation
23. Preferential Attachment Model: Social Media Mining - (Ch. 4, Pg 125 - 128) - Textbook
24. https://networkx.org/documentation/stable/auto_examples/index.html