AT1 - AUSTRALIAN SKILLS CLASSIFICATION ANALYSIS

36118 APPLIED NATURAL LANGUAGE PROCESSING - AUTUMN 2025

Shashikanth Senthil Kumar

25218722

Table of Contents

1. Project Overview
2. Data Understanding
   - 2.1. Install and Import Packages
   - 2.2. Data Loading
   - 2.3. General EDA
3. Selected Datasets
4. Preprocessing Steps
5. Word Frequency Analysis
6. TF-IDF Analysis
7. N-gram Analysis
8. Topic Modeling
9. Clustering Analysis
10. Similarity Analysis
    - 10.1. ESCO Dataset
    - 10.2. Cosine Similarity using TF-IDF
    - 10.3. BERT Embeddings
11. Key Findings
12. Future Skills for Data Scientists
13. Conclusion and Future Works
14. References

1. Project Overview

↳ 1 cell hidden

2. Data Understanding

To perform the analysis, we first **installed and imported all necessary libraries** required for text processing and visualization, including **NLTK, Gensim, pyLDAvis, and umap**. And then, we downloaded **Punk_tab, Stopwords, and WordNet** from the **NLTK library** to assist in tokenization, stopword removal, and lemmatization.

Next, we loaded the given **Excel file** (Australian Skills Classification dataset) and automated the process of **assigning each sheet as a separate DataFrame** (df1, df2, df3, ...). This revealed that the dataset consists of **11 different sheets**, each containing different aspects of skill classification.

To gain an initial understanding of the dataset, we performed **general Exploratory Data Analysis (EDA)** on each sheet, including:

- Displaying the **first few rows** using `head()`.
- Generating **statistical summaries** for both numerical (`describe()`) and categorical (`describe(include='object')`) columns.
- Checking **data types** using `info()`.
- Checking **null values** using `isna()`.
- Identifying **duplicate records** and assessing **data quality**.

2.1. Install and Import Packages

```
# When you run for the first time make sure to remove the Quotes and run the below code to install the required packages to run this project
"""
!pip install pyLDAvis
!pip install umap-learn
!pip install sentence-transformers
!pip install gensim
!pip install spacy
!pip install wordcloud
!pip install nltk
"""
```

```
'\n!pip install pyLDAvis\n!pip install umap-learn\n!pip install sentence-transformers\n!pip install gensim\n!pip install spacy\n!pip install wordcloud\n!pip install nltk
\n'
```

```
# Import all the required packages
import numpy as np
import pandas as pd
import re
import nltk
import spacy
import torch
import gensim
```

```
import math
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import pyLDAvis # Module for interactive topic model visualization
from pyLDAvis.gensim import prepare # Function to prepare LDA model for visualiztion
from collections import Counter
from nltk import bigrams
import networkx as nx
from nltk.util import ngrams
from nltk.corpus import stopwords,wordnet
from nltk.tokenize import word_tokenize
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sentence_transformers import SentenceTransformer
from wordcloud import WordCloud
from gensim import corpora, models
from sklearn.cluster import KMeans
from sklearn.metrics import  silhouette_score
from umap import UMAP
from sklearn.metrics.pairwise import cosine_similarity


# Download necessary NLTK resources
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
```

```
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
True
```

```
import warnings
warnings.filterwarnings('ignore')


# Initialize lemmatizer and Stop words
lemmatizer = WordNetLemmatizer()
stop_words =stopwords.words('english')


# This code is to display the entire output of the code
pd.set_option("display.max_rows", None)  # Show all rows
pd.set_option("display.max_columns", None)  # Show all columns
pd.set_option("display.width", None)  # Adjust width to fit the screen
pd.set_option("display.max_colwidth", None)  # Show full column contents
```

2.2 Data Loading

```
# Load all sheets into a dictionary
file_path = "Australian Skills Classification - December 2023.xlsx"
dfs = pd.read_excel(file_path, sheet_name=None)  # Load all sheets
```

Assign each sheets in the file as seperate datasets to explore each

```
# Dynamically assign each sheet to df1, df2, df3, ...
for i, (sheet_name, df) in enumerate(dfs.items(), start=1):
    globals()[f"df{i}"] = df  # Creates df1, df2, df3, ...
print(i) # To find how many sheets we have in the given excel file
```

```
11
```

2.3 General EDA

[ ] ↳ 93 cells hidden

3. **Selected Datasets**

After completing a **general exploratory data analysis (EDA)** on the 11 sheets in the Australian Skills Classification (ASC) dataset, we identified **five key sheets** that are most relevant for text analysis:

1. **Glossary (gloss_df)** - Contains definitions and explanations of terms, which are useful for text processing and understanding domain-specific terminology.
2. **Occupation Details (occ_df)** - Provides job-related descriptions, allowing for analysis of trends, key skills, and competencies required for different occupations.
3. **Core Competency Description (comp_df)** - Lists core competencies and proficiency levels, essential for analyzing skill classifications.
4. **Specialist Task Hierarchy (skill_df)** - Details structured tasks performed within occupations, enabling better categorization and clustering of skills.
5. **Technology Tools (tools_df)** - Provides a list of relevant tools and technologies, which is useful for understanding AI and tech-related skills.

Datasets Used for Analysis

For this project, we selected the following two datasets for different NLP tasks:

- **Occupation Details (occ_df)**: Used for **word frequency analysis, n-gram analysis, topic modeling, and similarity analysis** to explore trends in job descriptions and skill requirements.
- **Specialist Task Hierarchy (skill_df)**: Used for **clustering analysis** to group skills based on their textual similarities and uncover underlying patterns.
- **ESCO Dataset (esco_df)**: A single csv file from the ESCO Dataset named "occupations_en.csv" was used as esco_df to perform similarity analysis with occ_df to find the similar jobs in both taxonomies.

4. **Preprocessing Steps**

To perform analysis we need to clean the dataset first for that we will define a function called occ_preprocess_text() which contains a series of preprocessing steps to clean and standardize the text data.

- **Lowercasing** - converts all the text to lowercase using text.lower() function.
- **Removing Numbers** - eliminates numeric values using regular expressions (re.sub(r'\d+', '', text)), which are irrelevantfor text based analysis.
- **Removing Punctuation** - removes special characters and punctuations using regular expressions (re.sub(r'[^\w\s]', '', text)), helps to focus only on meaningful words.
- **Tokenization** - splits the text into individual words using a NLTK library function word_tokenize().
- **Stopword Removal** - removes the most commonly used words and the words that doesn't contribute significant meaning to text (eg:'the',is','and'...).
- **Lemmatization** - converts words to their base form using WordNetLemmatizer(), which helps to reduce the vocablary size while retaining the same meaning (eg:'running'='run').
- **Returning cleaned data** - finally the processed texts are rejoined using " ".join(tokens) and returned to do further analysis.

```
# Function for text preprocessing
def occ_preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'\d+', '', text)  # Remove numbers
    text = re.sub(r'[^\w\s]', '', text)  # Remove punctuation
    tokens = word_tokenize(text)  # Tokenization
    tokens = [word for word in tokens if word not in stop_words]  # Remove stopwords
    tokens = [lemmatizer.lemmatize(word) for word in tokens]  # Lemmatization
    return " ".join(tokens)
```

5. **Word Frequency Analysis**

In this analysis, we aim to identify the most common words in occupation descriptions to understand the most in-demand skills in Australia.

Step 1: Counting Words in Raw Descriptions

- We first count the number of words in the description column using the word_count function.
- The word count is stored in the description_length column.
- Then, we analyze the most frequent words in the raw text (without preprocessing) to observe common patterns.

Step 2: Preprocessing Descriptions

- We apply the occ_preprocess_text function to clean the text by:
    - Converting to lowercase.
    - Removing numbers and punctuation.
    - Tokenizing and removing stopwords.
    - Applying lemmatization.
- The cleaned output is stored in the cleaned_description column.

Step 3: Refining Stopwords

- The initial word frequency analysis of cleaned_description revealed many generic job-related terms unrelated to skills.
- We created a custom stopword list and extended the stop_words set.
- The text was reprocessed using the updated stopword list.
- The final cleaned text was stored in cleaned_description_length for comparison.

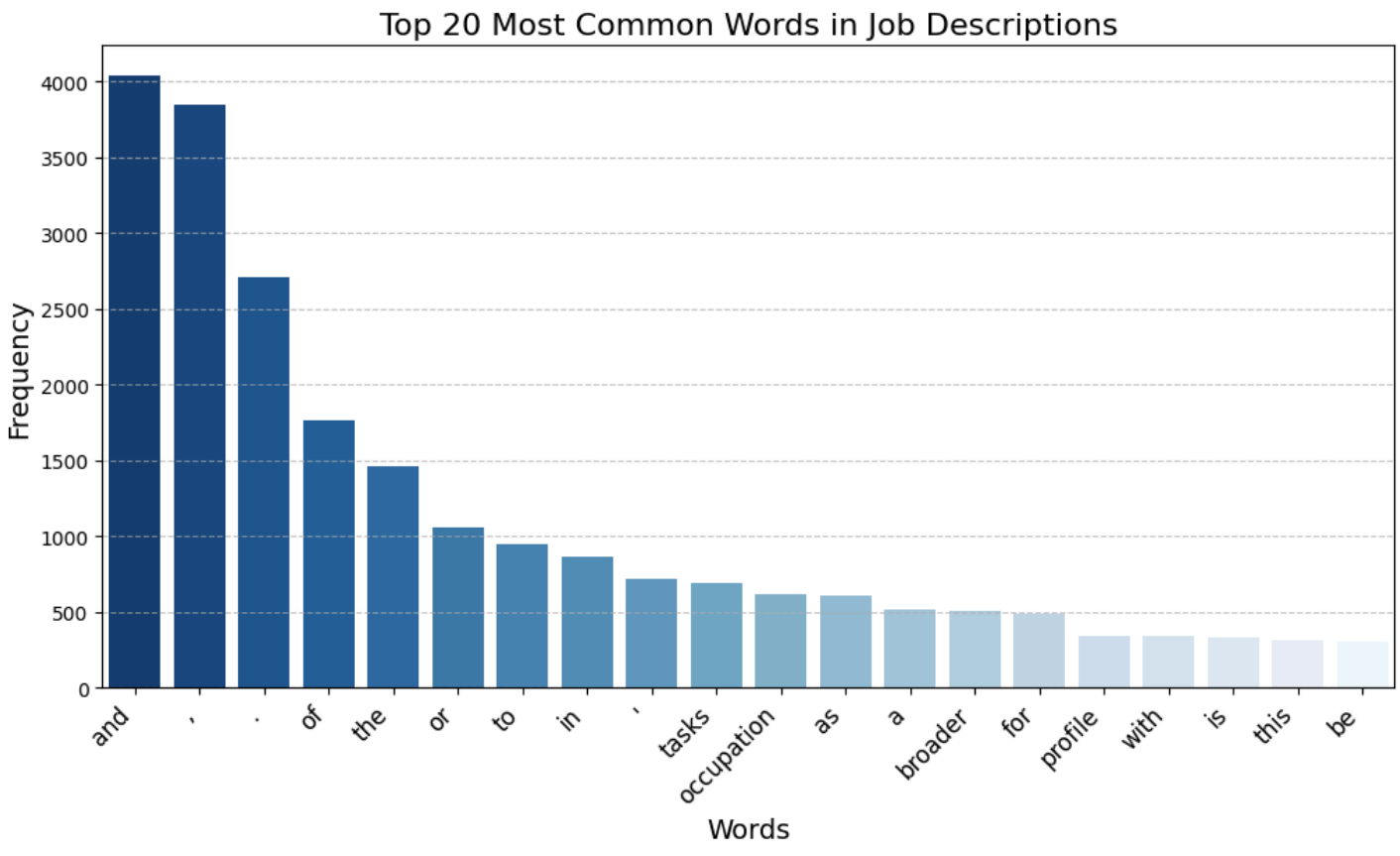
Step 4: Final Word Frequency Analysis

- After preprocessing, we extracted the most common words from cleaned_description.
- These frequent terms provide key insights into the most in-demand skills in Australia.

Step 1: Counting Words in Raw Descriptions

```
occ_df.rename(columns={'ANZSCO Description': 'description'}, inplace=True)
```

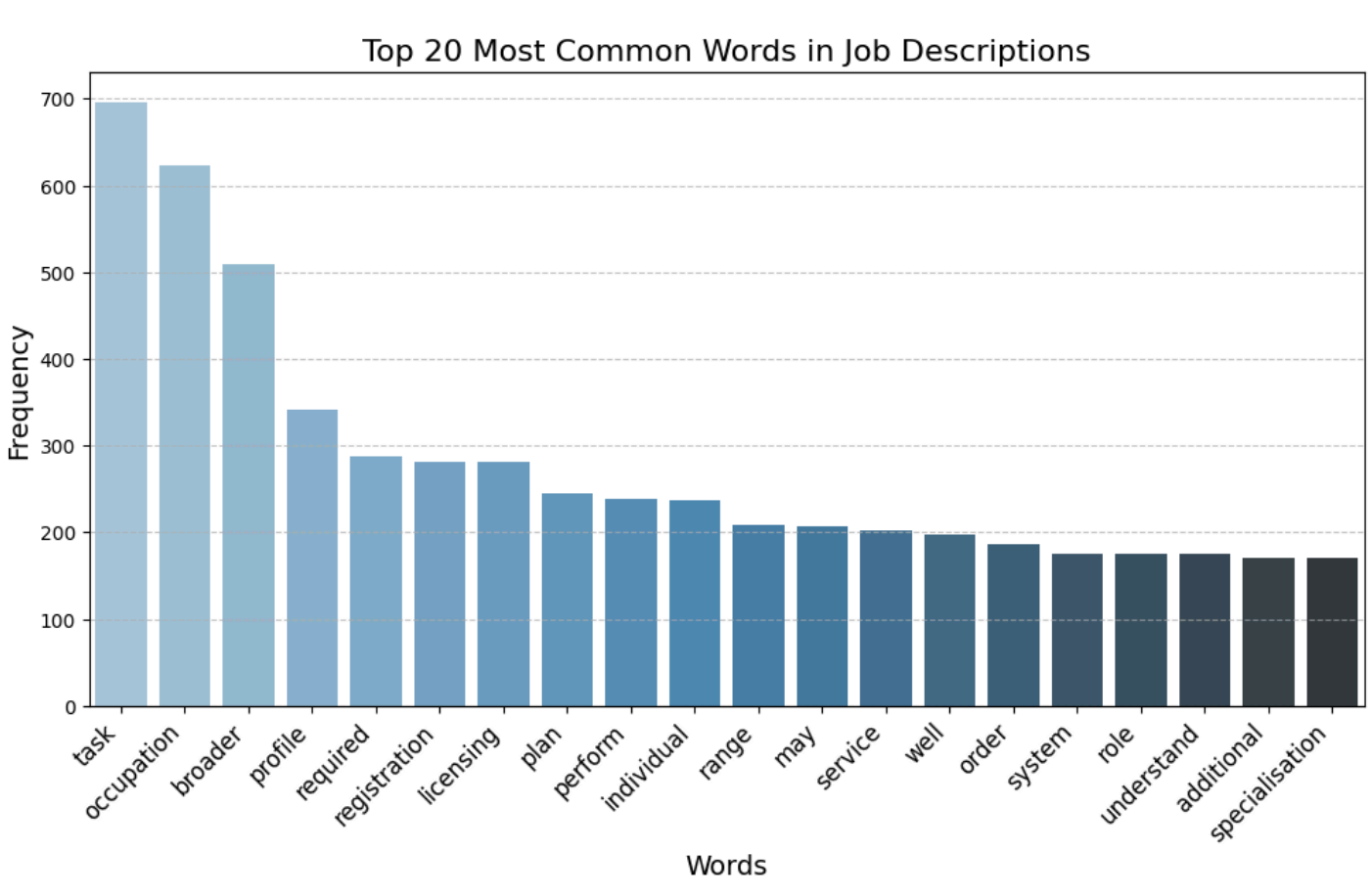
```
def word_count(text):
    wc = len(text.split())
    return wc


# create a new column consists the length of the description
occ_df['description_length'] = occ_df['description'].astype(str).apply(word_count)


# Combine all job descriptions
all_words = " ".join(occ_df["description"]).lower()
tokens = word_tokenize(all_words)

# Count word frequencies
word_freq = Counter(tokens)
common_words = word_freq.most_common(50)


for i,j in common_words:
  print(i)


# Get the top 20 most common words
top_words = word_freq.most_common(20)

# Extract words and their frequencies
words, frequencies = zip(*top_words)

# Plot the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(x=list(words), y=list(frequencies), palette="Blues_r")
plt.xticks(rotation=45, ha="right", fontsize=12)
plt.ylabel("Frequency", fontsize=14)
plt.xlabel("Words", fontsize=14)
plt.title("Top 20 Most Common Words in Job Descriptions", fontsize=16)
plt.grid(axis="y", linestyle="--", alpha=0.7)
```




```
# Generate Word Cloud
wordcloud = WordCloud(max_words=50,width=800, height=400, background_color="white", colormap="viridis").generate_from_frequencies(word_freq)

# Plot the Word Cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")  # Hide axes
plt.title("Word Cloud of Job Descriptions")
```

Step 2: Preprocessing Descriptions

```
# Apply preprocessing to job descriptions
occ_df["cleaned_description"] = occ_df["description"].astype(str).apply(occ_preprocess_text)


# ---- WORD FREQUENCY ANALYSIS ----
all_words = " ".join(occ_df["cleaned_description"]).split()
word_freq = Counter(all_words)
common_words = word_freq.most_common(50)  # Top 50 words


for i,j in common_words:
  print(i)


# Get the top 20 most common words
top_words = word_freq.most_common(20)

# Extract words and their frequencies
words, frequencies = zip(*top_words)

# Plot the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(x=list(words), y=list(frequencies), palette="Blues_d")
plt.xticks(rotation=45, ha="right", fontsize=12)
plt.ylabel("Frequency", fontsize=14)
plt.xlabel("Words", fontsize=14)
plt.title("Top 20 Most Common Words in Job Descriptions", fontsize=16)
plt.grid(axis="y", linestyle="--", alpha=0.7)
```




```
# Generate Word Cloud
wordcloud = WordCloud(max_words=50,width=800, height=400, background_color="white", colormap="viridis").generate_from_frequencies(word_freq)
```

```
# Plot the Word Cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")  # Hide axes
plt.title("Cleaned Word Cloud of Job Descriptions")
```

Step 3: Refining Stopwords

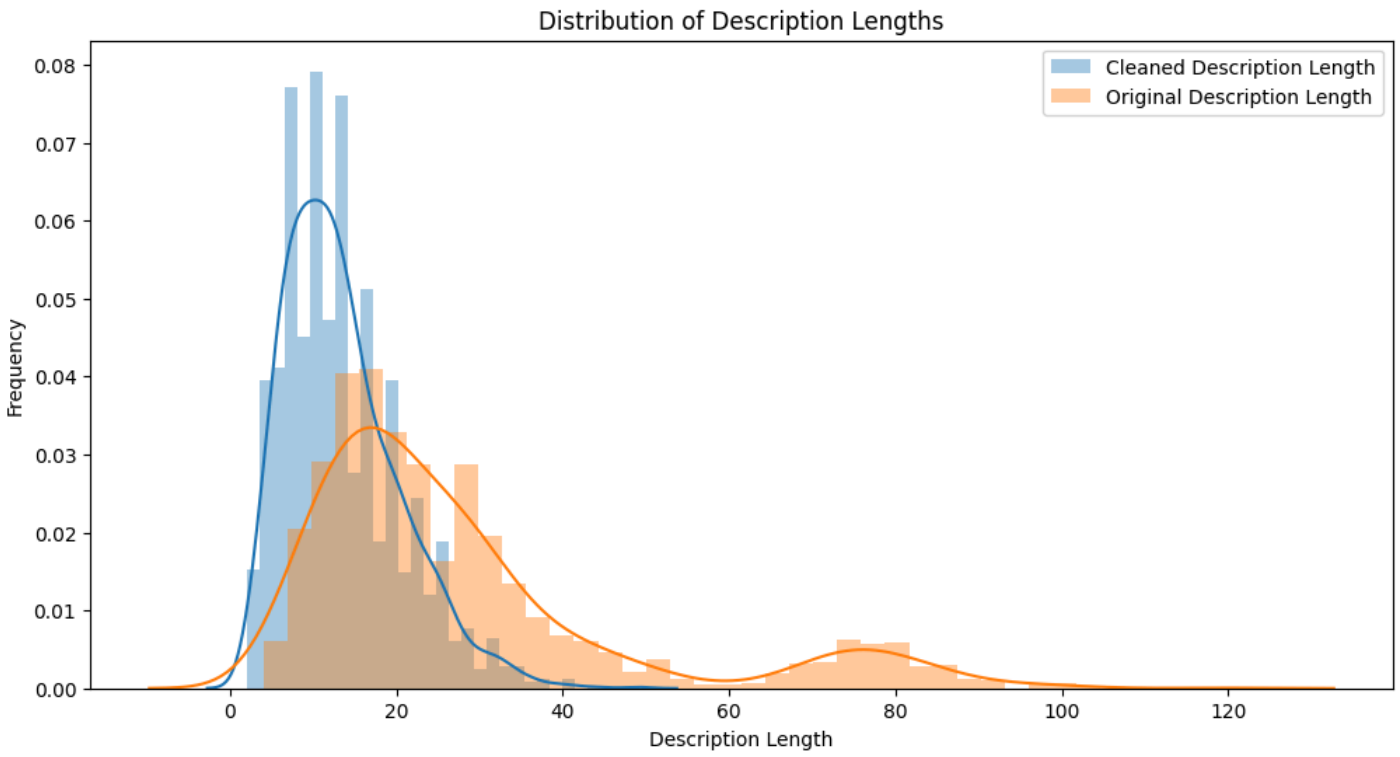
Many frequent words in job descriptions were generic, structural, or non-informative key skills. Removing them helped highlight technical and domain-specific competencies.

- Generic job terms (eg: role, task, position) lacked skill relevance.
- Structural words (eg: require, provide, ensure) described expectations, not skills.
- Industry-wide terms (eg: coordinate, organization) appeared across all fields, adding little insight.
- Common verbs (eg: plan,perform, design) described actions, not specific expertise.

```
# ---- DEFINE CUSTOM STOPWORDS ----
custom_stopwords =[
    "may", "well", "full", "additional", "understand", "order", "range",
    "occupation", "role", "worker", "group", "individual", "task", "performs", "provides", "organises",
    "broader", "profile", "required", "registration", "licensing", "subset", "combined", "undertaken",
    "plan", "perform", "coordinate", "control", "assist", "support", "design",
    "service", "system", "operation", "activity", "organisation", "cover", "data", "nec"
]


# Combine with NLTK stopwords
stop_words.extend(custom_stopwords)


#now run the preprocess again to job descriptions
occ_df["cleaned_description"] = occ_df["cleaned_description"].astype(str).apply(occ_preprocess_text)
```

```
# create a new column consists the length of the cleaned_deascription
occ_df['cleaned_description_length'] = occ_df['cleaned_description'].astype(str).apply(word_count)


occ_df.sort_values(by ='cleaned_description_length',ascending=True).head()


occ_df.sort_values(by ='cleaned_description_length',ascending=False).head()


# Create the distplot
plt.figure(figsize=(12, 6))
sns.distplot(occ_df['cleaned_description_length'], label="Cleaned Description Length")
sns.distplot(occ_df['description_length'], label="Original Description Length")
plt.legend()
plt.title('Distribution of Description Lengths')
plt.xlabel('Description Length')
plt.ylabel('Frequency')
```

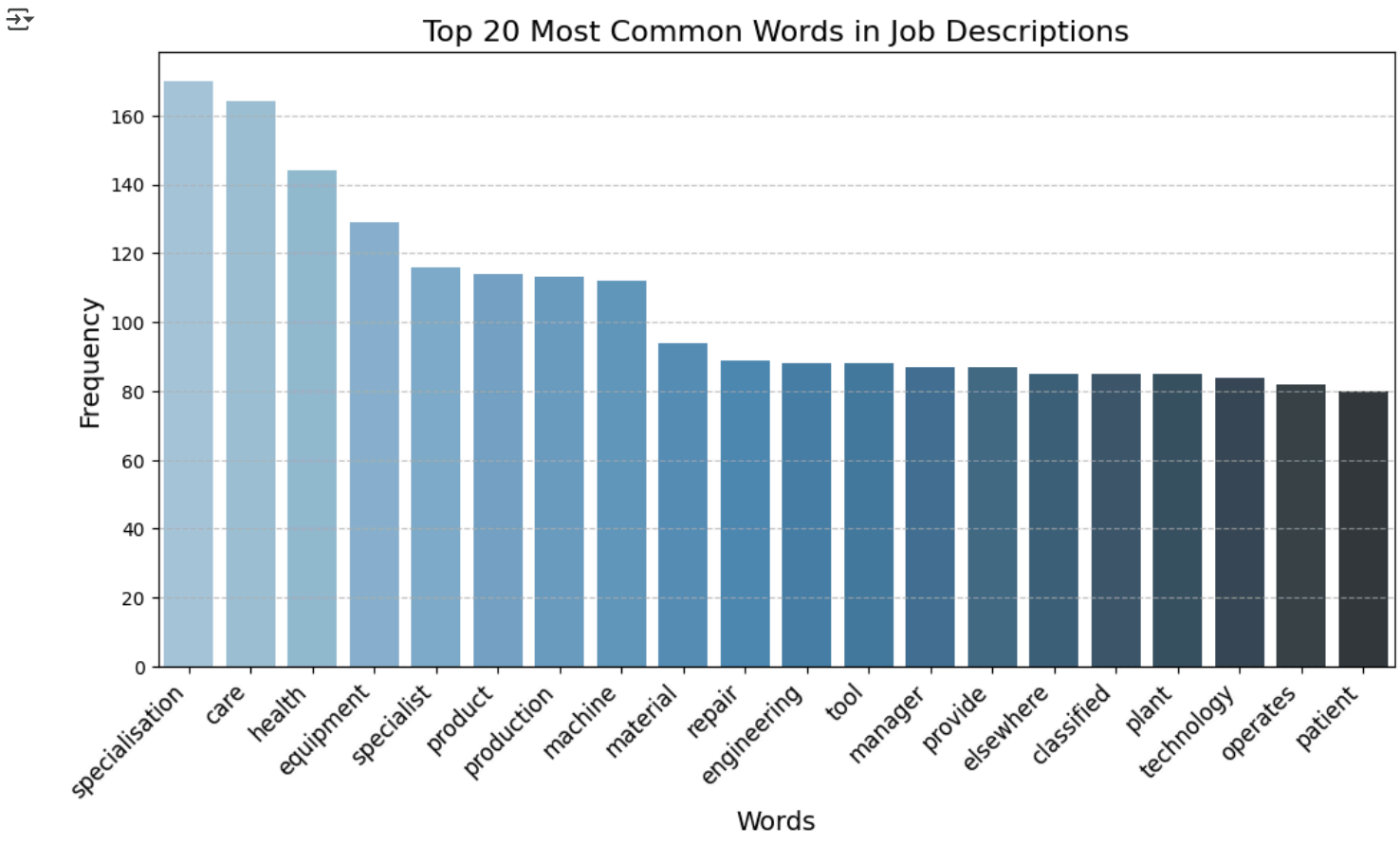
```
Text(0, 0.5, 'Frequency')
```



Step 4: Final Word Frequency Analysis

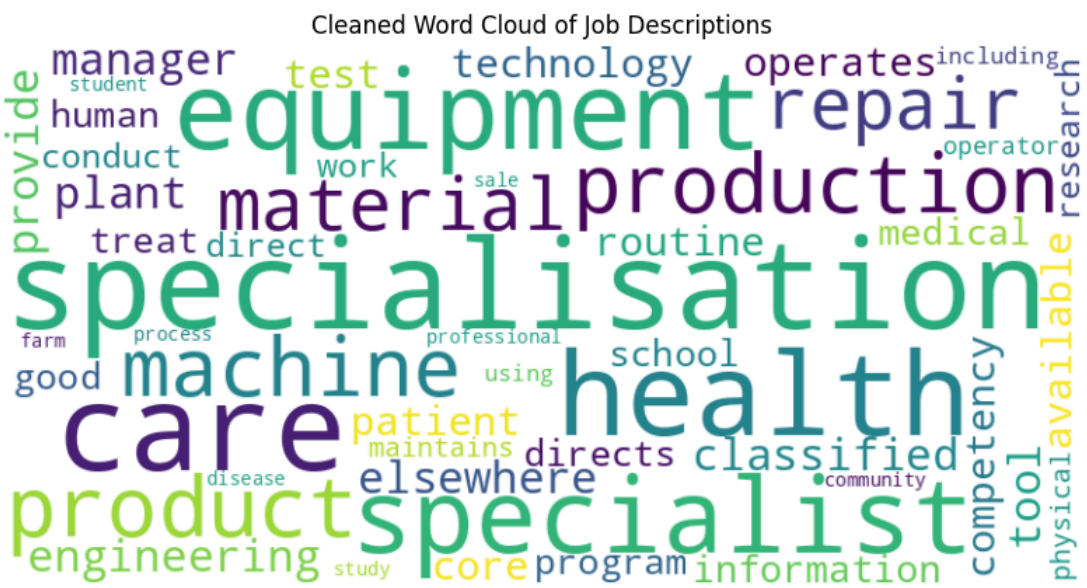
```
# ---- WORD FREQUENCY ANALYSIS ----
all_words = " ".join(occ_df["cleaned_description"]).split()
word_freq = Counter(all_words)
common_words = word_freq.most_common(50)


for i,j in common_words:
  print(i)


# Get the top 20 most common words
top_words = word_freq.most_common(20)

# Extract words and their frequencies
words, frequencies = zip(*top_words)

# Plot the bar chart
plt.figure(figsize=(12, 6))
sns.barplot(x=list(words), y=list(frequencies), palette="Blues_d")

# Formatting
plt.xticks(rotation=45, ha="right", fontsize=12)
```

```
plt.ylabel("Frequency", fontsize=14)
plt.xlabel("Words", fontsize=14)
plt.title("Top 20 Most Common Words in Job Descriptions", fontsize=16)
plt.grid(axis="y", linestyle="--", alpha=0.7)
```




```
# Generate Word Cloud
wordcloud = WordCloud(max_words=50,width=800, height=400, background_color="white", colormap="viridis").generate_from_frequencies(word_freq)

# Plot the Word Cloud
plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")  # Hide axes
plt.title("Cleaned Word Cloud of Job Descriptions")
```

```
Text(0.5, 1.0, 'Cleaned Word Cloud of Job Descriptions')
```



INSIGHTS:

From the most frequent words we find some key themes in Australian occupations:

- Healthcare & Medical Roles – Words like health, patient, medical, disease, treat suggest a strong presence of healthcare-related professions, aligning with Australia's demand for medical workers.
- Engineering & Technical Fields – Terms such as engineering, material, tool, machine, repair, technology, equipment highlight the significance of technical and manufacturing roles, emphasizing Australia's industrial workforce.
- Education & Research – The presence of student, school, study, research, program suggests that education and research-based roles are prominent, indicating a focus on knowledge-based industries.
- Management & Operations – Words like manager, professional, directs, maintains, process, competency indicates that leadership and operational skills are crucial across sectors.
- Agriculture & Trade – Terms like farm, sale, plant, production point to the relevance of agriculture, retail, and supply chain industries in Australia.

6. **TF-IDF Analysis**

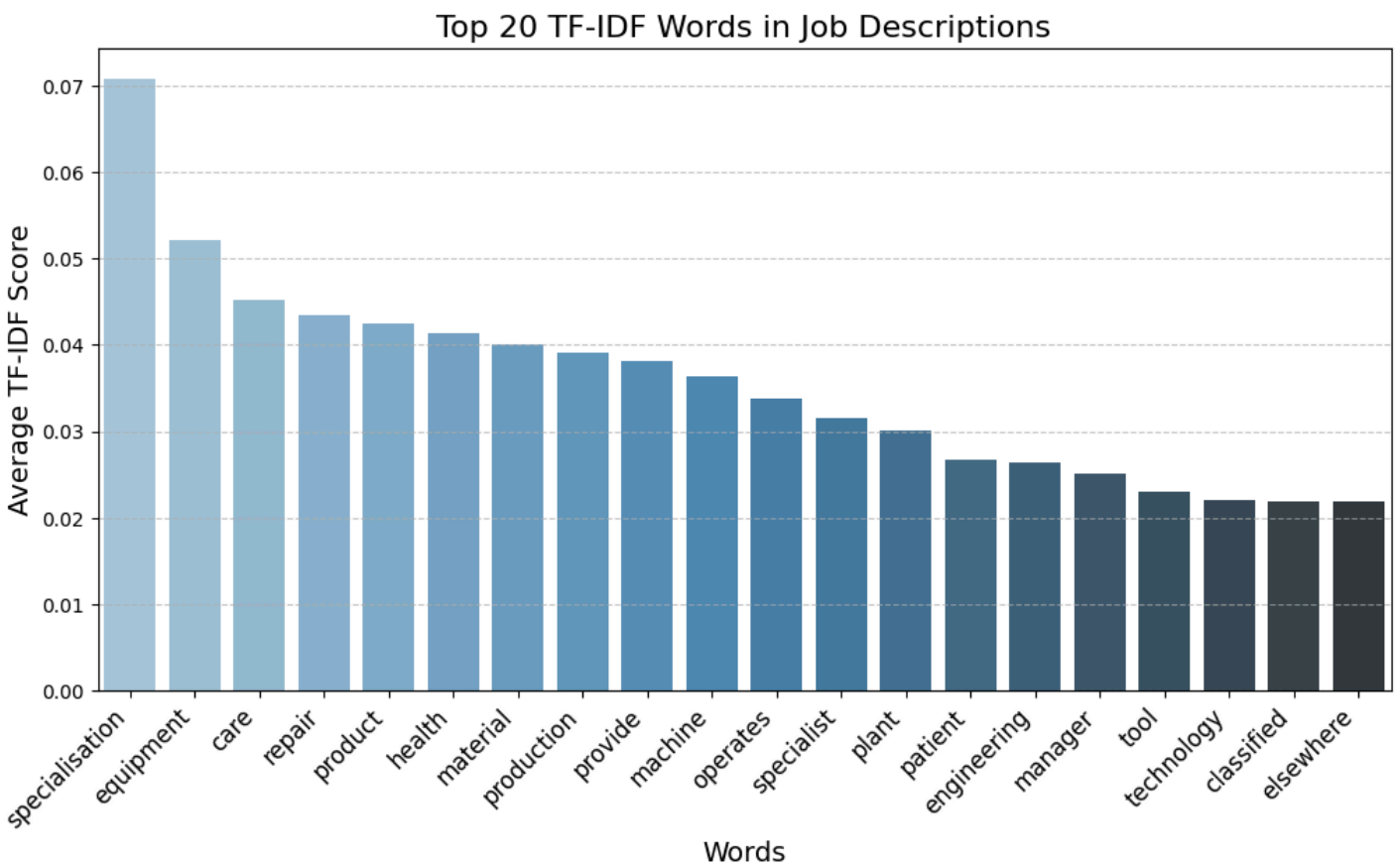
After conducting word frequency analysis, TF-IDF analysis is applied to highlight **unique and contextually important words** in the cleaned_description column. While word frequency analysis identifies the most common words, TF-IDF refines this by giving higher weight to words that are frequent in specific descriptions but rare across the entire dataset. This reduces the influence of common words and emphasizes **distinctive terms** that are more relevant to the specific roles. Thus TF-IDF helps to identify key skills and tasks, making the analysis more precise and insightful for understanding critical competencies in data science and AI.

```
# Sample job descriptions dataset
job_descriptions = occ_df["cleaned_description"].astype(str).tolist()
stop_words.extend(custom_stopwords)
# Initialize TF-IDF vectorizer
vectorizer = TfidfVectorizer(max_features=20, stop_words=stop_words)
```

```
# Fit and transform the data
tfidf_matrix = vectorizer.fit_transform(job_descriptions)

# Convert to DataFrame
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())

# Display top words with highest TF-IDF scores
print("\nTop TF-IDF Words:\n", tfidf_df.mean().sort_values(ascending=False))
```

```
Top TF-IDF Words:
 specialisation    0.070838
equipment         0.052074
care              0.045219
repair            0.043408
product           0.042469
health            0.041377
material          0.040146
production        0.039178
provide           0.038118
machine           0.036324
operates          0.033874
specialist        0.031590
plant             0.030168
patient           0.026782
engineering       0.026328
manager           0.025053
tool              0.023067
technology        0.022106
classified        0.021926
elsewhere         0.021926
dtype: float64
```

```
# Compute mean TF-IDF scores for each word
tfidf_means = tfidf_df.mean().sort_values(ascending=False)

#BAR CHART: Top 20 TF-IDF Words
plt.figure(figsize=(12, 6))
sns.barplot(x=tfidf_means.index, y=tfidf_means.values, palette="Blues_d")
plt.xticks(rotation=45, ha="right", fontsize=12)
plt.ylabel("Average TF-IDF Score", fontsize=14)
plt.xlabel("Words", fontsize=14)
plt.title("Top 20 TF-IDF Words in Job Descriptions", fontsize=16)
plt.grid(axis="y", linestyle="--", alpha=0.7)
```




```
#WORD CLOUD: TF-IDF Weighted Words
word_scores = dict(zip(tfidf_means.index, tfidf_means.values))

plt.figure(figsize=(12, 6))
wordcloud = WordCloud(width=800, height=400, background_color="white").generate_from_frequencies(word_scores)

plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("TF-IDF Word Cloud", fontsize=16)
```

INSIGHTS:

1. **Specialization**: Words like **specialisation**, **specialist**, and **engineering** highlight the importance of specialized knowledge and expertise in roles.
2. **Industry-Specific**: Terms such as **equipment**, **repair**, **material**, and **machine** point to technical or industrial roles involving hands-on work.
3. **Healthcare Focus**: Words like **care**, **health**, **patient**, and **product** suggest a strong presence of healthcare-related roles.
4. **Management & Operations**: Terms like **manager**, **plant**, **production**, and **operates** indicate roles with managerial and operational responsibilities.
5. **Technology**: **Technology**, **machine**, and **tool** shows the importance of advanced technology and machinery in certain roles.

7. **N-Gram Analysis**

N-Gram analysis helps uncover meaningful word combinations and contextual relationships in job descriptions. It helps to find how specific skills and tasks are described together, identifying key skill sets like "data analysis," "machine learning" etc... It also helps capture industry-

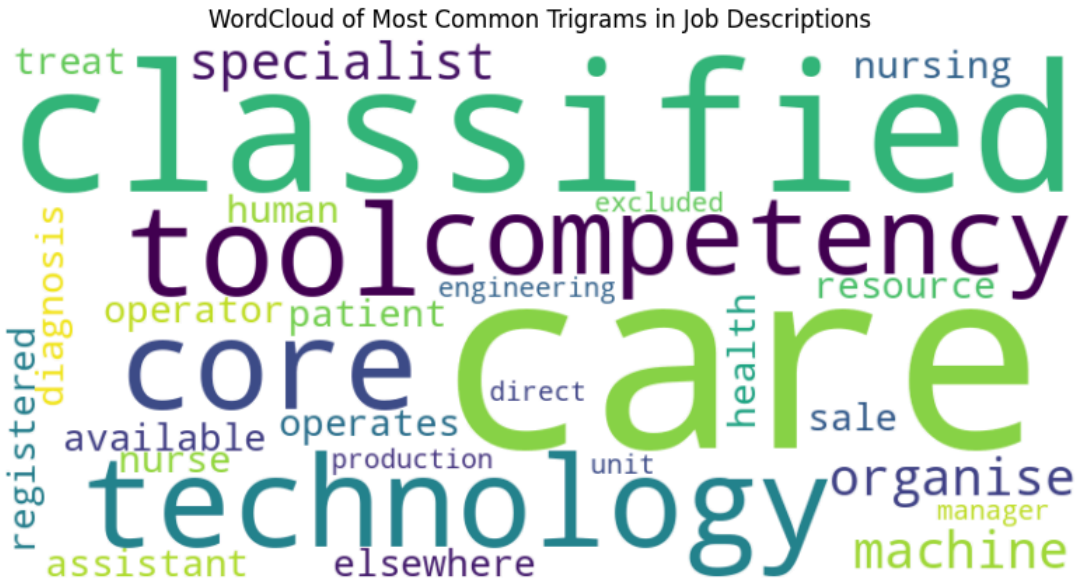
specific terminology, improve text classification, and spot emerging trends in job roles. By analyzing word pairings or sequences, N-Gram analysis provides a deeper understanding of job requirements and enhances workforce planning and skill identification.

```
text_data = " ".join(occ_df["cleaned_description"]).lower()
tokens = word_tokenize(text_data)

# Generate Bigrams and Trigrams
bigrams = list(ngrams(tokens, 2))
trigrams = list(ngrams(tokens, 3))

# Count frequency
bigram_freq = Counter(bigrams)
trigram_freq = Counter(trigrams)


# Top 50 bigrams & trigrams
top_bigrams = bigram_freq.most_common(50)
top_trigrams = trigram_freq.most_common(50)


print("Top 50 Bigrams:")
for i,j in top_bigrams:
  print(" ".join(i))


print("Top 50 Trigrams:")
for i,j in top_trigrams:
  print(" ".join(i))


# Visualization: WordCloud for Bigrams
bigram_phrases = [' '.join(bg) for bg, _ in bigram_freq.most_common(20)]
bigram_text = ' '.join(bigram_phrases)

wordcloud = WordCloud(width=800, height=400, background_color='white').generate(bigram_text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("WordCloud of Most Common Bigrams in Job Descriptions")
```

```
Text(0.5, 1.0, 'WordCloud of Most Common Bigrams in Job Descriptions')
```

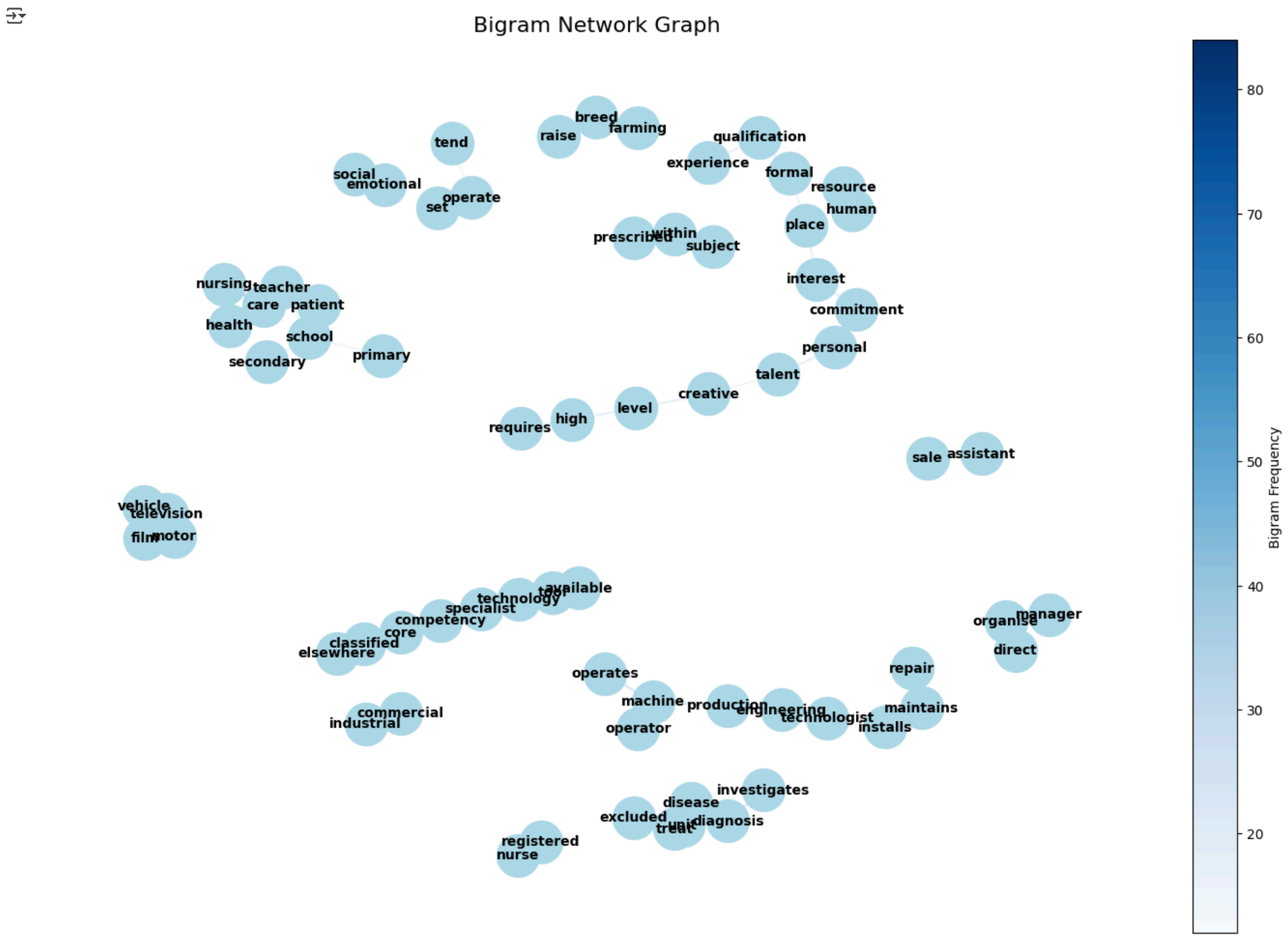

```
# Visualization: WordCloud for Bigrams
trigram_phrases = [' '.join(tri) for tri, _ in trigram_freq.most_common(20)]
trigram_text = ' '.join(trigram_phrases)

wordcloud = WordCloud(width=800, height=400, background_color='white').generate(bigram_text)

plt.figure(figsize=(10, 5))
plt.imshow(wordcloud, interpolation="bilinear")
plt.axis("off")
plt.title("WordCloud of Most Common Trigrams in Job Descriptions")
```

```
Text(0.5, 1.0, 'WordCloud of Most Common Trigrams in Job Descriptions')
```




```
from nltk import bigrams

def plot_bigram_network(tokens, n=50):
    # Generate bigrams
    bi_grams = list(bigrams(tokens))

    # Count bigram frequencies
    bigram_freq = Counter(bi_grams)

    # Create network graph
    G = nx.Graph()
```

```
# Lists for edge properties
edge_width = []
edge_color = []

# Add edges based on frequency
for (word1, word2), count in bigram_freq.most_common(n):
    G.add_edge(word1, word2, weight=count)
    edge_width.append(count)  # Store frequency for width
    edge_color.append(count)  # Store frequency for color

# Normalize edge width for better visualization
edge_width = np.array(edge_width)
edge_width = 1 + (edge_width - edge_width.min()) * 5 / (edge_width.max() - edge_width.min())  # Scale between 1 and 6

# Normalize edge colors for colormap
edge_norm = plt.Normalize(vmin=min(edge_color), vmax=max(edge_color))
edge_colors_mapped = plt.cm.Blues(edge_norm(edge_color))  # Convert values to RGBA

# Create figure and axis
fig, ax = plt.subplots(figsize=(16, 12))  # Explicitly create figure and axis

# Compute node positions
pos = nx.spring_layout(G, k=0.5, iterations=50)

# Draw nodes
nx.draw_networkx_nodes(G, pos, node_size=1000, node_color='lightblue', ax=ax)

# Draw edges with normalized colors
nx.draw_networkx_edges(G, pos, width=edge_width, edge_color=edge_colors_mapped, ax=ax)

# Draw labels
nx.draw_networkx_labels(G, pos, font_size=10, font_weight="bold", ax=ax)

# Set title and remove axis
ax.set_title('Bigram Network Graph', fontsize=16)
ax.axis('off')

    # Add a colorbar with explicit axis
    sm = plt.cm.ScalarMappable(cmap=plt.cm.Blues, norm=edge_norm)
    sm.set_array([])
    cbar = fig.colorbar(sm, ax=ax, fraction=0.046, pad=0.04)
    cbar.set_label('Bigram Frequency', fontsize=10)

# Run the function with cleaned tokenized text
plot_bigram_network(tokens)
```



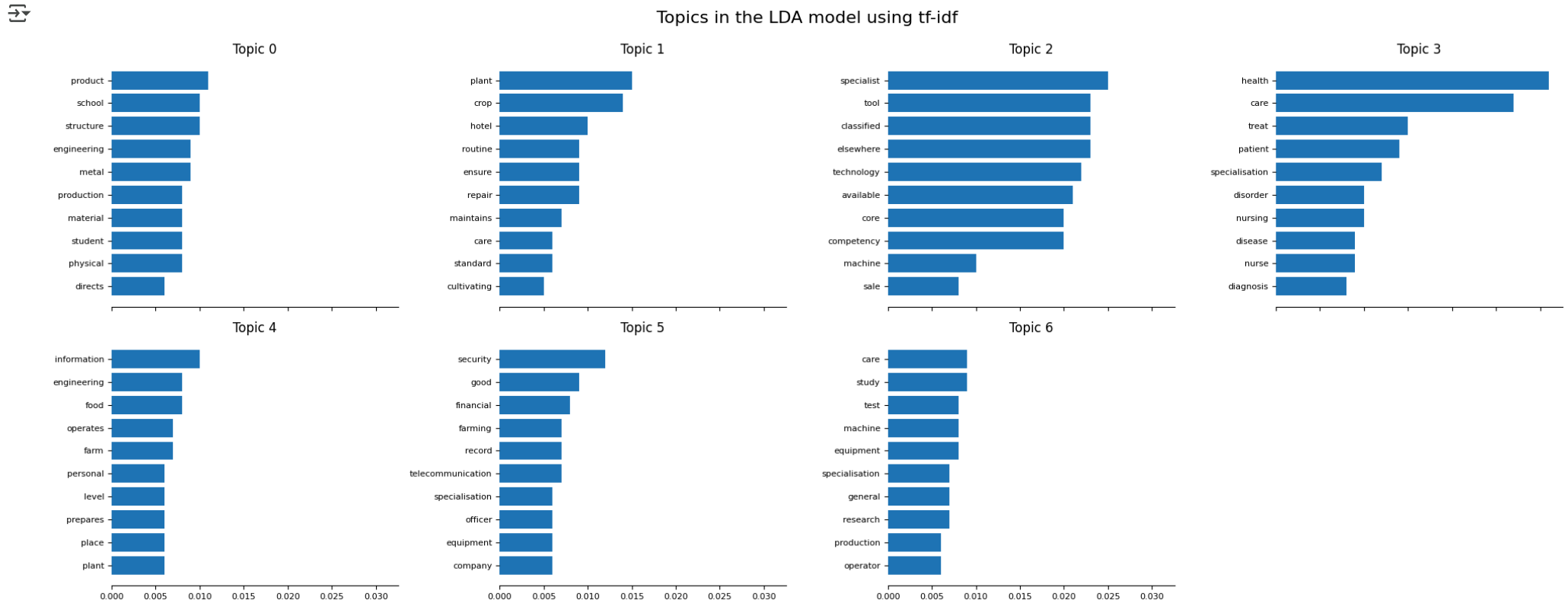
INSIGHTS:

- **Technical and Healthcare Roles**: A significant number of bigrams and trigrams are related to healthcare, technical, and engineering roles, such as "nursing care," "specialist technology," and "machine operator".
- **Educational and Supervisory Focus**: Many trigrams emphasize training, supervision, and qualification, indicating the importance of leadership and education in various industries.
- **Personal Attributes and Commitment**: Phrases like "personal commitment interest" highlight the growing importance of personal passion and commitment to the job, especially in roles like healthcare, teaching, and management.
- **Emerging Skills**: The presence of terms like "creative talent" and "high level creative" signals a growing demand for creativity and high-level cognitive skills across industries.

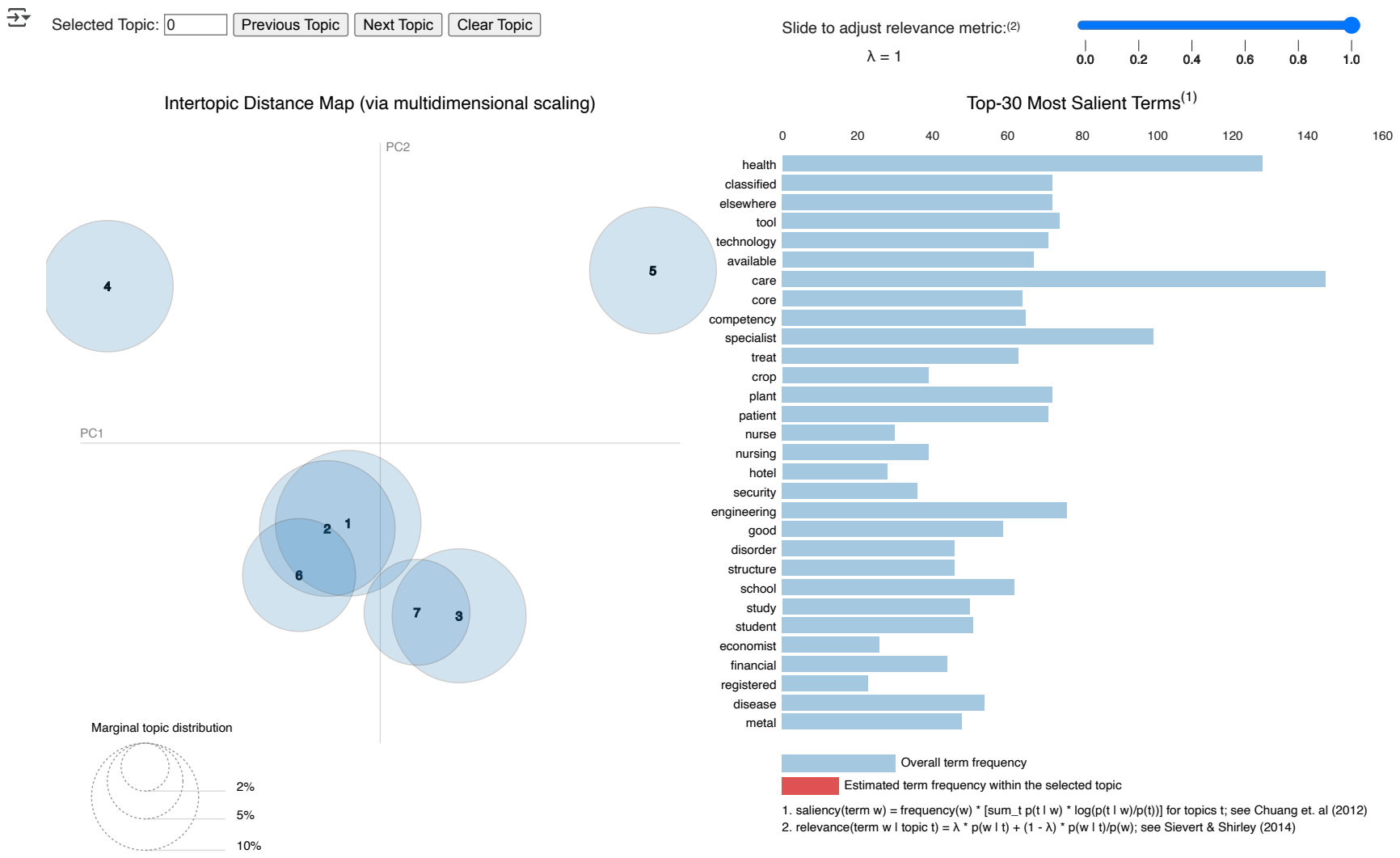
8. **Topic Modeling**

Topic Modeling is essential in analysing job trends because it helps to uncover hidden themes, complex patterns and relationships within large datasets. It performs grouping on related terms as topics which helps to interpret and classify the data easily.By identifying emerging trends and skill demands, topic modeling aids in workforce planning and recruitment, highlighting competencies required for different roles. It improves text classification, automates job matching, and allows for efficient analysis of large datasets, providing actionable insights without manual review. Ultimately, it helps organizations stay ahead of industry shifts and better align job descriptions with the skills in demand. Also topic model gives different outputs on each time they run,so to overcome this problem we have fixed the random.seed and random_state as 42. This ensures that we get the same result on every run.

```
occ_df["cleaned_desc"] = occ_df["cleaned_description"].apply(lambda x: word_tokenize(x))
# Create dictionary and corpus for LDA
dictionary = corpora.Dictionary(occ_df["cleaned_desc"])
corpus = [dictionary.doc2bow(text) for text in occ_df["cleaned_desc"]]

# Set a fixed random seed for reproducibility
np.random.seed(42)
gensim.utils.random.seed(42)

# Apply LDA with 5 topics
lda_model = models.LdaModel(corpus, num_topics=7, id2word=dictionary, passes=10, random_state=42)

# Print topics
for idx, topic in lda_model.print_topics(-1):
    print(f"Topic {idx}: {topic}")


# Visualize the topics in bar plots
def plot_top_words(model, feature_names, n_top_words, title):
    # Get the number of topics
    num_topics = len(model.print_topics(-1))

    # Calculate the number of rows and columns for subplots
    num_cols = min(4, num_topics)  # Maximum 4 columns
    num_rows = math.ceil(num_topics / num_cols)

    # Create subplots
    fig, axes = plt.subplots(num_rows, num_cols, figsize=(5*num_cols, 4*num_rows), sharex=True)
    axes = axes.flatten() if num_topics > 1 else [axes]

    for topic_idx, topic in model.print_topics(-1):
        # Use regex to extract words and weights
        words_weights = re.findall(r'(0\.\d+)\*"(.+?)"', topic)

        if words_weights:
            weights, words = zip(*words_weights)
            weights = [float(w) for w in weights]
        else:
            print(f"No matches found for topic {topic_idx}")
            continue

        ax = axes[topic_idx]
        ax.barh(words[:n_top_words], weights[:n_top_words])
        ax.set_title(f'Topic {topic_idx}')
        ax.invert_yaxis()
        ax.tick_params(axis='both', which='major', labelsize=8)
        for i in 'top right left'.split():
            ax.spines[i].set_visible(False)

    # Hide any unused subplots
    for idx in range(num_topics, len(axes)):
        axes[idx].set_visible(False)

    fig.suptitle(title, fontsize=16)
    plt.tight_layout()
    plt.subplots_adjust(top=0.9)  # Adjust to prevent title overlap

plot_top_words(lda_model, dictionary.values(), n_top_words=10, title='Topics in the LDA model using tf-idf')
```




```
# Enable the display of visualizations in IPython Notebooks
pyLDAvis.enable_notebook()

# Prepare and transform and LDA model
pyLDAvis_data = prepare(lda_model, corpus, dictionary)


pyLDAvis.display(pyLDAvis_data)
```



INSIGHTS:

- Topic 0: Engineering and Education
  - This topic is centered around engineering and education-related terms. Words like "product," "school," "structure," and "student" suggest roles involving the production or design of educational or structural materials.
  - The focus is on technical skills (e.g., "engineering," "metal," "production," and "material"), potentially indicating roles in educational institutions or engineering-based industries.
- Topic 1: Agricultural and Routine Management
  - This topic is related to roles in agriculture and facility management, with terms like "plant," "crop," "hotel," "routine," and "repair."
  - It suggests a mix of agricultural responsibilities ("crop," "plant," "cultivating") and operational or maintenance tasks in environments like hotels or similar settings ("routine," "repair," "standard").
- Topic 2: Specialized Competency and Technology
  - This topic emphasizes specialized roles requiring specific competencies and knowledge in technology.
  - Words like "specialist," "tool," "technology," "competency," and "machine" point to highly skilled positions, possibly in technical fields or industries where precise tools and technologies are crucial. It reflects roles that require expertise and the ability to work with advanced technologies or specialized tools.
- Topic 3: Healthcare and Nursing
  - Focused on healthcare and medical fields, this topic highlights roles in nursing and patient care.
  - Terms like "health," "care," "patient," "nurse," and "diagnosis" point to occupations in healthcare, where treatment, diagnosis, and care of patients are central responsibilities. It indicates the importance of medical specialization and care for various disorders.
- Topic 4: Engineering, Information, and Agriculture
  - This topic combines roles involving engineering, information, and agriculture.
  - Terms like "information," "engineering," "food," and "farm" suggest positions that require both technical and agricultural knowledge, possibly related to food production, agricultural engineering, or systems involved in farming operations.
- Topic 5: Security, Finance, and Telecommunication
  - Roles in security, finance, and telecommunication dominate this topic.
  - The words "security," "financial," "company," "telecommunication," and "farming" indicate jobs related to managing security measures, financial records, or telecommunications infrastructure. This may also include roles in the agricultural sector ("farming") with an overlap in technology or infrastructure management.
- Topic 6: Research, Equipment, and Specialized Care
  - This topic highlights roles involving research, specialization, and the use of equipment.
  - Keywords like "care," "study," "test," "machine," and "research" suggest positions related to scientific or medical research, possibly in laboratory settings or technical fields requiring specialized knowledge in testing and equipment management.

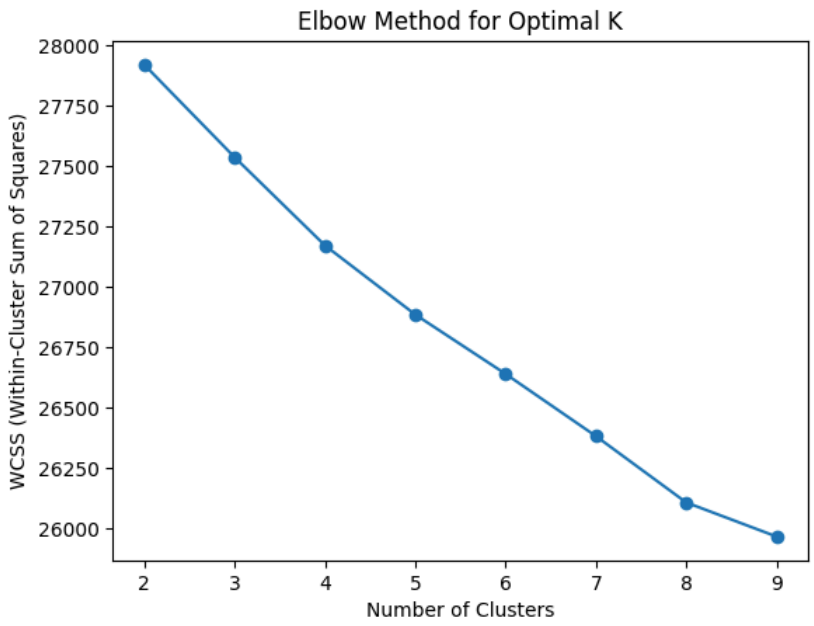
9. **Clustering**

Clustering is important for the job analysis because it helps to group the similar job roles based on their shared attributes which makes it easier to identify trends and patterns. It aids in efficient data segmentation, enhances job matching and gives dynamic insights into the job market.

```
skill_df.head()
```

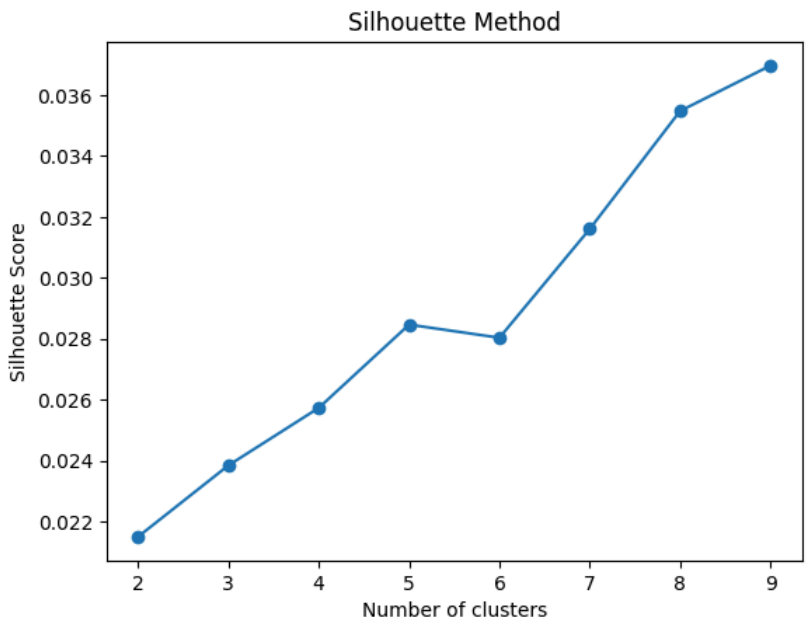
```
# Convert text data (e.g., job descriptions, skills) into TF-IDF features
vectorizer = TfidfVectorizer(stop_words='english', max_features=500)
X = vectorizer.fit_transform(skill_df['Skills Statement'])


# Find the optimal K using Elbow Method
wcss = []
for k in range(2, 10):  # Testing different K values
    kmeans = KMeans(n_clusters=k, random_state=42, n_init=10)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)

# Plot the Elbow Method
plt.plot(range(2, 10), wcss, marker='o')
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS (Within-Cluster Sum of Squares)")
plt.title("Elbow Method for Optimal K")
```

```
Text(0.5, 1.0, 'Elbow Method for Optimal K')
```




```
# Function to plot the Silhouette Method
silhouette_avg_scores = []
k_range = range(2,10)
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42,n_init=10)
    kmeans.fit(X)
    labels = kmeans.labels_
    silhouette_avg = silhouette_score(X, labels)
    silhouette_avg_scores.append(silhouette_avg)
plt.figure()
plt.plot(k_range, silhouette_avg_scores, marker='o')
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Method')
```

```
Text(0.5, 1.0, 'Silhouette Method')
```



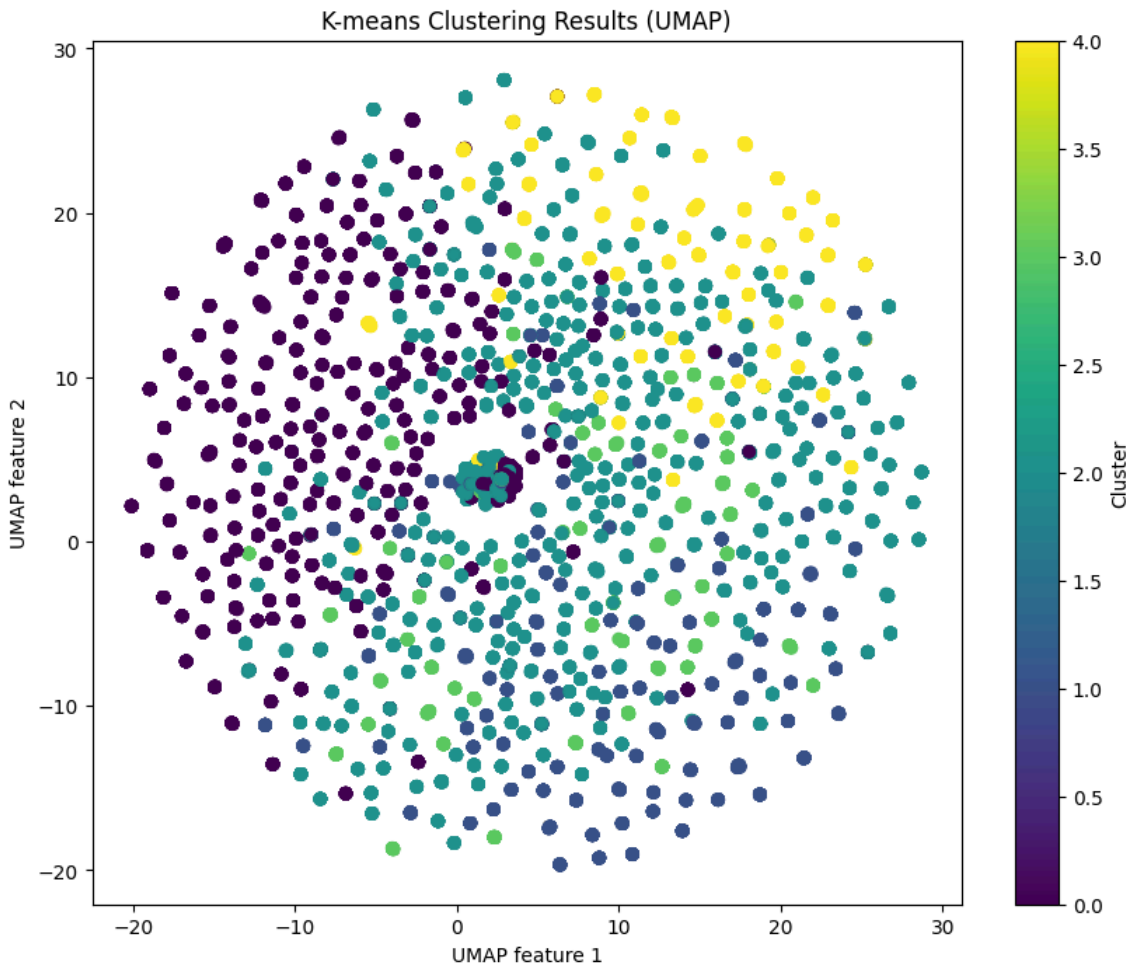

```
# Fit K-Means with the optimal K
optimal_k = 5  # Choose based on elbow method result
kmeans = KMeans(n_clusters=optimal_k, random_state=42, n_init=10)
skill_df["cluster"] = kmeans.fit_predict(X)

# Print sample clusters
for i in range(optimal_k):
    print(f"\nCluster {i}:")
    print(skill_df[skill_df["cluster"] == i]["Skills Statement"].head(10))  # Display top 10


# Calculate Silhouette Score
silhouette_avg = silhouette_score(X, skill_df["cluster"])
print(f"Silhouette Score: {silhouette_avg}")
```

```
Silhouette Score: 0.028463649004172822
```

```
# Loop through each cluster and display word cloud separately
for i in range(optimal_k):
    # Extract text from the cluster
    cluster_text = " ".join(skill_df[skill_df["cluster"] == i]["Skills Statement"].dropna())
    # Generate word cloud
    wordcloud = WordCloud(width=800, height=500, background_color="white",max_words=50).generate(cluster_text)
    # Display the word cloud
    plt.figure(figsize=(10, 6))
    plt.imshow(wordcloud, interpolation="bilinear")
```

```
    plt.axis("off")
    plt.title(f"Word Cloud for Cluster {i}", fontsize=14)


# Reduce dimensionality using UMAP
umap = UMAP(n_components=2, random_state=42)
X_umap = umap.fit_transform(X)


#Plot the clusters
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_umap[:, 0], X_umap[:, 1], c=kmeans.labels_, cmap='viridis')
plt.title('K-means Clustering Results (UMAP)')
plt.xlabel('UMAP feature 1')
plt.ylabel('UMAP feature 2')
plt.colorbar(scatter, label='Cluster')
```

```
<matplotlib.colorbar.Colorbar at 0x7bd027840d10>
```



INSGHTS:

- **Cluster 0**
  - It includes job roles focused on equipment maintenance, inspections, and safety in sectors like agriculture and forestry.
  - The cluster highlights skills in equipment handling, risk management, and performing agricultural tasks like grading and sowing.
- **Cluster 1**
  - It focuses on analytical, financial, and communication-related tasks.
  - It includes skills in data analysis, risk assessment, financial reporting, and compliance monitoring.
- **Cluster 2**
  - It focuses on strategic planning, financial management, stakeholder engagement, and regulatory compliance.
  - It includes skills in data analysis for performance evaluation, business and financial planning, budgeting, contract negotiation, and policy implementation.
- **Cluster 3**
  - It focuses on inventory management, emergency response, and professional skill maintenance.
  - It includes skills related to tracking and maintaining stock levels, conducting audits, and ensuring the usability of resources and equipment.
- **Cluster 4**
  - It focuses on leadership, management, and oversight.
  - It includes directing financial, operational, sales, HR, and research activities, ensuring compliance with regulations, and managing budgets, resources, and timelines.

  From this analysis we have got a silhouette score of 0.0285 which indicates the overlap in the clusters we can see this in the Scatterplot.

10. **Similarity Analysis**

The aim of this analysis is to find is there any similar job roles between European dataset(esco_df) and Australian Dataset(occ_df). We will be using two text similarity techniques Tf-IDF with Cosine Similrity and Bidirectional Encoder Representations from Transformers(BERT) Embeddings. By computing similarity scores we try to identify closely related job roles between these taxonomies and understand how skills ans job roles align globally. First we explore the European dataset(esco_df) and then proceed with the similarity analysis.

10.1. **ESCO Dataset**

[ ] ↳ 8 cells hidden

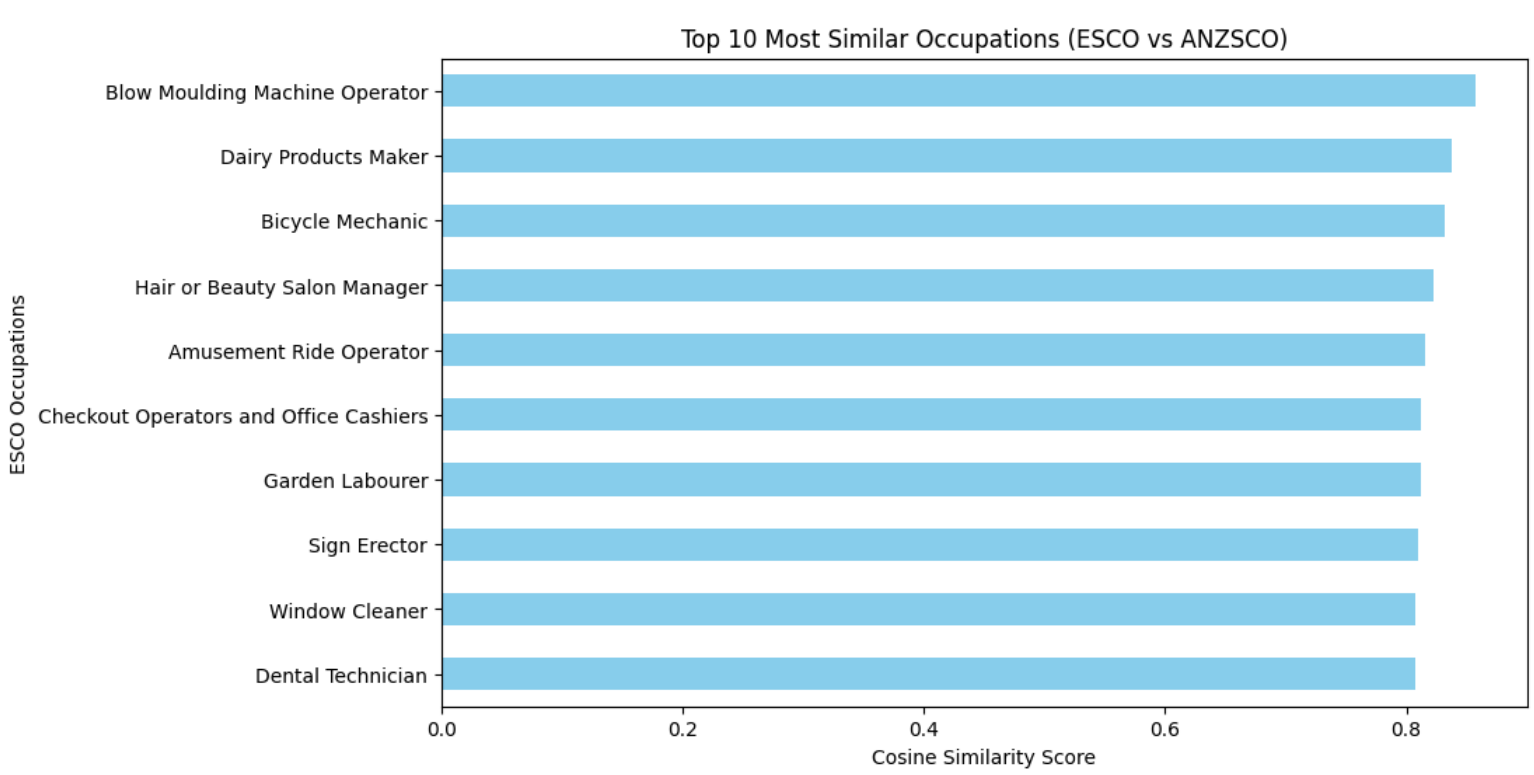
10.2. **TF_IDF Analysis with Cosine Similarity**

This analysis converts job titles and descriptions into term frequency-inverse document frequency(TF-IDF) vectors and measures the cosine similarity to find the matching occupations and the least similar occupations.

```
# Extract text fields for comparison
esco_texts = esco_df["preferredLabel"].astype(str) + " " + esco_df["cleaned_description"].astype(str)
occ_texts = occ_df["ANZSCO Title"].astype(str) + " " + occ_df["cleaned_description"].astype(str)

# TF-IDF Vectorization
vectorizer = TfidfVectorizer(stop_words="english", max_features=5000,ngram_range=(1,2))
esco_tfidf = vectorizer.fit_transform(esco_texts)
occ_tfidf = vectorizer.transform(occ_texts)

# Compute Cosine Similarity
similarity_matrix = cosine_similarity(occ_tfidf,esco_tfidf)

# Convert to DataFrame for readability
similarity_df = pd.DataFrame(similarity_matrix, index=occ_df["ANZSCO Title"], columns=esco_df["preferredLabel"])


# Find top 10 ANZSCO occupations with the highest similarity scores
top_anzsco_jobs = similarity_df.max(axis=0).nlargest(10).index  # Get top 10 ANZSCO jobs
# Find the most similar ESCO job for each ANZSCO job
top_esco_jobs = similarity_df.loc[:, top_anzsco_jobs].idxmax(axis=0)  # Get ESCO jobs with highest similarity
# Subset the similarity matrix for these pairs
top_similarity = similarity_df.loc[top_esco_jobs, top_anzsco_jobs]
# Display Top Similar Occupations
top_matches_tfidf = top_similarity.max(axis=1).sort_values(ascending=False).head(10)
print("\nTop Matching Occupations (TF_IDF):\n", top_matches_tfidf)

# Find top 10 ANZSCO occupations with the least similarity scores
last_anzsco_jobs = similarity_df.min(axis=0).nsmallest(10).index  # Get last 10 ANZSCO jobs
# Find the least similar ESCO job for each ANZSCO job
last_esco_jobs = similarity_df.loc[:, last_anzsco_jobs].idxmax(axis=0)  # Get ESCO jobs with least similarity
# Subset the similarity matrix for these pairs
least_similarity = similarity_df.loc[last_esco_jobs, top_anzsco_jobs]
# Display Top Similar Occupations
least_matches_tfidf = similarity_df.min(axis=1).sort_values(ascending=True).head(10)
print("\nLeast Similar Occupations (TF_IDF):\n", least_matches_tfidf)
```

```
Top Matching Occupations (TF_IDF):
 ANZSCO Title
Blow Moulding Machine Operator            0.857456
Dairy Products Maker                      0.837233
Bicycle Mechanic                          0.832136
Hair or Beauty Salon Manager              0.822085
Amusement Ride Operator                   0.815831
Checkout Operators and Office Cashiers    0.811905
Garden Labourer                           0.811786
Sign Erector                              0.810170
Window Cleaner                            0.807634
Dental Technician                         0.807371
dtype: float64

Least Similar Occupations (TF_IDF):
 ANZSCO Title
Chief Executives and Managing Directors              0.0
Broadcast Transmitter Operator                       0.0
Jeweller                                             0.0
Library Technician                                   0.0
Gallery or Museum Technician                         0.0
Power Generation Plant Operator                      0.0
Wind Turbine Technician                              0.0
Gas or Petroleum Operator                            0.0
Chemical Plant Operator                              0.0
Wood Machinists and Other Wood Trades Workers nec    0.0
dtype: float64
```

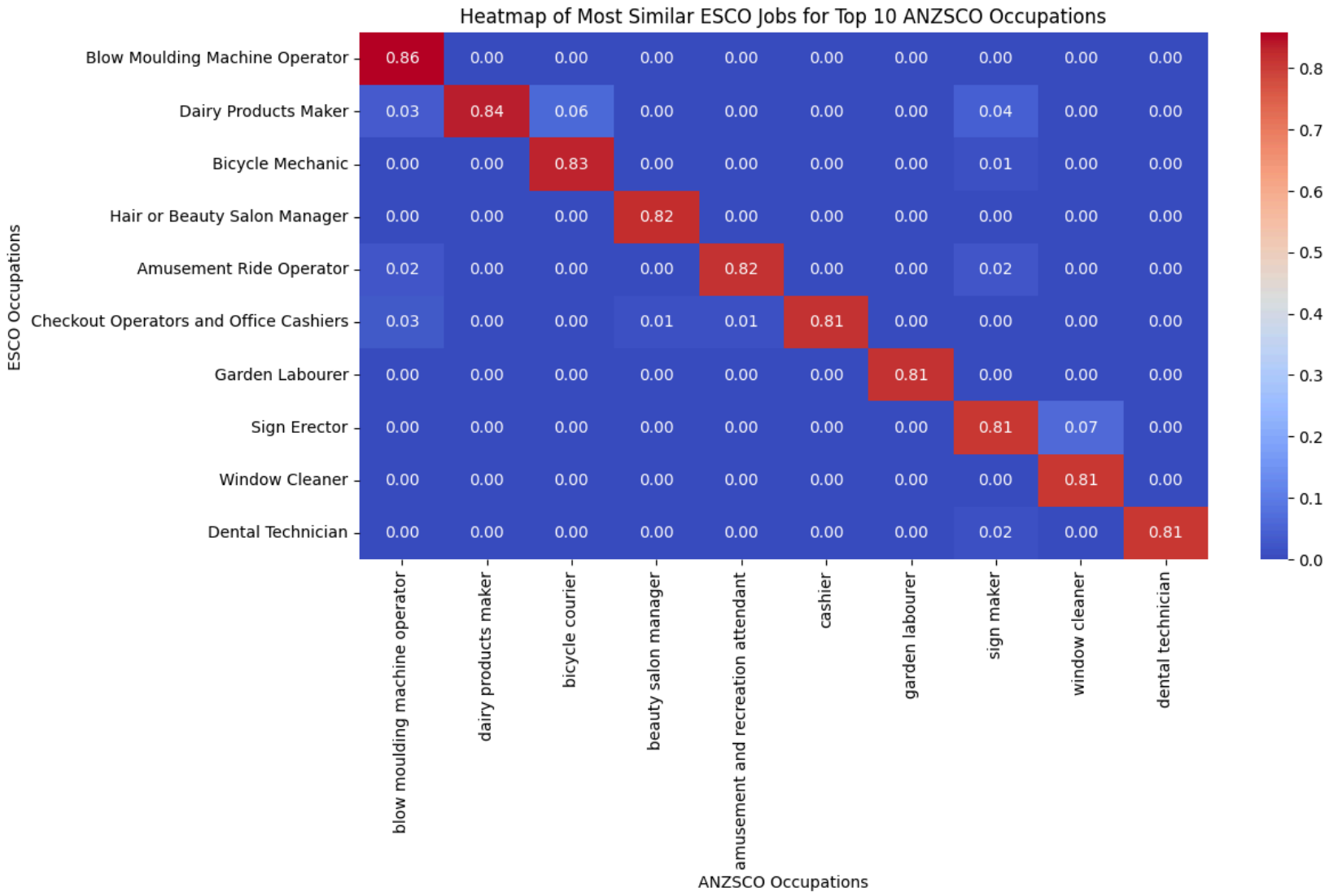
```
# Bar plot for Top Matches
top_matches_tfidf.plot(kind="barh", figsize=(10, 6), color="skyblue")
plt.xlabel("Cosine Similarity Score")
plt.ylabel("ESCO Occupations")
plt.title("Top 10 Most Similar Occupations (ESCO vs ANZSCO)")
plt.gca().invert_yaxis()  # Invert y-axis for better readability
```




```
# Find top 10 ANZSCO occupations with the highest similarity scores
top_anzsco_jobs = similarity_df.max(axis=0).nlargest(10).index  # Get top 10 ANZSCO jobs

# Find the most similar ESCO job for each ANZSCO job
top_esco_jobs = similarity_df.loc[:, top_anzsco_jobs].idxmax(axis=0)  # Get ESCO jobs with highest similarity

# Subset the similarity matrix for these pairs
top_similarity_df = similarity_df.loc[top_esco_jobs, top_anzsco_jobs]
```

```
# Plot the heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(top_similarity_df, annot=True, fmt=".2f", cmap="coolwarm",
            xticklabels=top_anzsco_jobs, yticklabels=top_esco_jobs)
plt.title("Heatmap of Most Similar ESCO Jobs for Top 10 ANZSCO Occupations")
plt.xlabel("ANZSCO Occupations")
plt.ylabel("ESCO Occupations")
plt.xticks(rotation=90)
```

```
(array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]),
 [Text(0.5, 0, 'blow moulding machine operator'),
  Text(1.5, 0, 'dairy products maker'),
  Text(2.5, 0, 'bicycle courier'),
  Text(3.5, 0, 'beauty salon manager'),
  Text(4.5, 0, 'amusement and recreation attendant'),
  Text(5.5, 0, 'cashier'),
  Text(6.5, 0, 'garden labourer'),
  Text(7.5, 0, 'sign maker'),
  Text(8.5, 0, 'window cleaner'),
  Text(9.5, 0, 'dental technician')])
```



OBSERVATION:

- Top Matching Occupations:
  - Occupations with the highest similarity (e.g., Blow Moulding Machine Operator, Amusement Ride Operator, Bicycle Mechanic) share common technical or domain-specific language.
  - Roles from similar sectors have overlapping vocabularies, indicating shared skill sets and job requirements.
- Least Similar Occupations:
  - Occupations with a similarity score of 0 (e.g., Chief Executives and Managing Directors, Jeweller, Power Generation Plant Operator) have distinct job descriptions and specialized terminology.
  - These roles belong to highly specialized sectors (e.g., leadership, technical engineering, crafts), highlighting a lack of overlap in language with more general or operational roles.
- Top matches indicate sectoral similarity, while least matches highlight clear distinctions between specialized job categories, suggesting the need for more refined taxonomies.

10.3. **BERT Analysis**

In this analysis we are using a pre-trained BERT language model(all-MiniLM-L6-v2) which transforms the texts into dense vector representations and capture the sematic meaning of the text, making it easier to compare the meaning with different texts. Then we are using Cosine similarity to determine the similarity of the texts.

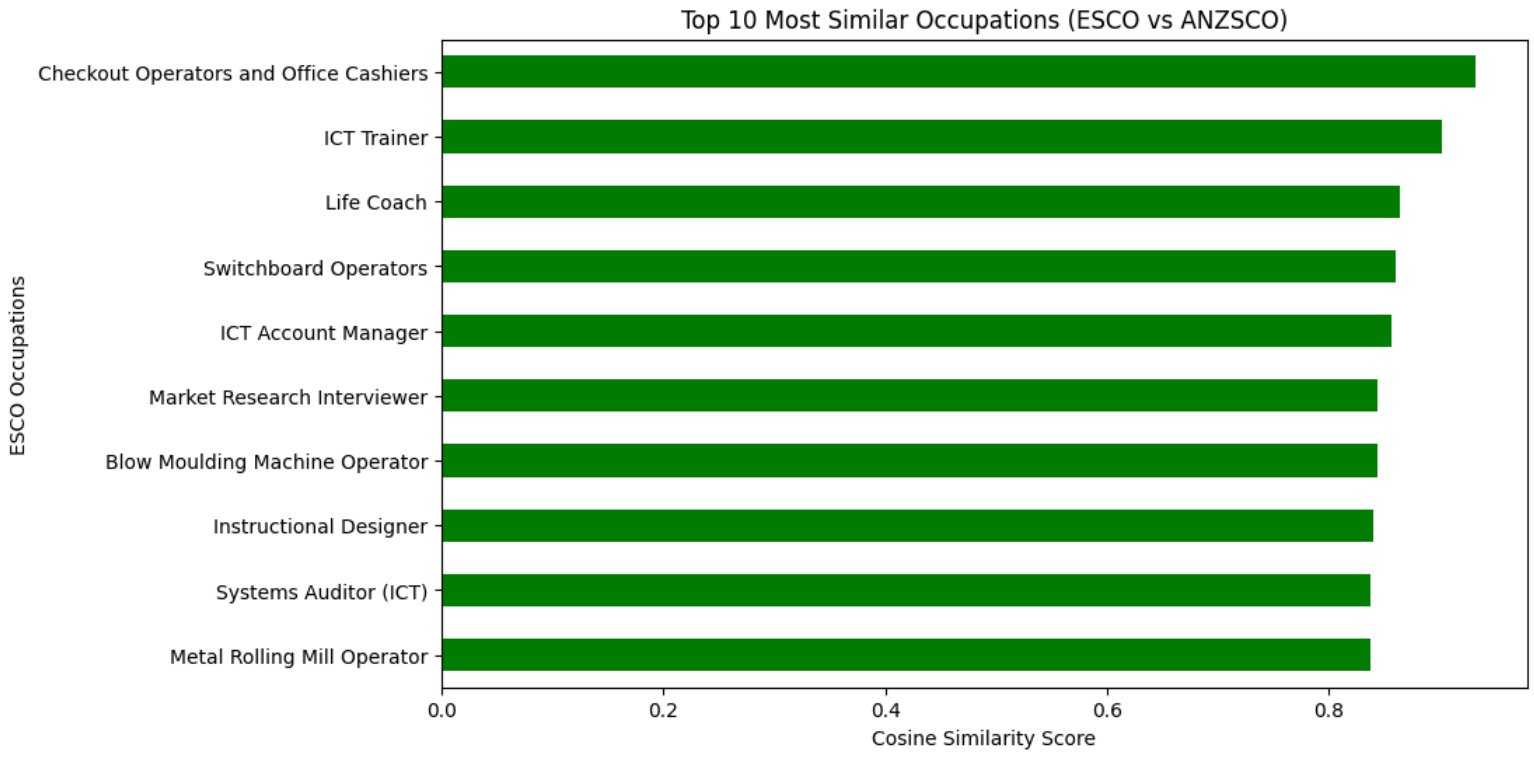
```
model = SentenceTransformer("all-MiniLM-L6-v2")
esco_embeddings = model.encode(esco_texts)
occ_embeddings = model.encode(occ_texts)
similarity_matrix_bert = cosine_similarity( occ_embeddings,esco_embeddings)
```

```
similarity_df_bert = pd.DataFrame(similarity_matrix_bert, columns=esco_df["preferredLabel"], index=occ_df["ANZSCO Title"])
# Find top 10 ANZSCO occupations with the highest similarity scores
top_anzsco_jobs = similarity_df_bert.max(axis=0).nlargest(10).index  # Get top 10 ANZSCO jobs
# Find the most similar ESCO job for each ANZSCO job
top_esco_jobs = similarity_df_bert.loc[:, top_anzsco_jobs].idxmax(axis=0)  # Get ESCO jobs with highest similarity
# Subset the similarity matrix for these pairs
top_similarity_bert = similarity_df_bert.loc[top_esco_jobs, top_anzsco_jobs]
# Display Top Similar Occupations
top_matches_bert = top_similarity_bert.max(axis=1).sort_values(ascending=False).head(10)
print("\nTop Matching Occupations (BERT):\n", top_matches_bert)

# Find top 10 ANZSCO occupations with the least similarity scores
last_anzsco_jobs = similarity_df_bert.min(axis=0).nsmallest(10).index  # Get last 10 ANZSCO jobs
# Find the least similar ESCO job for each ANZSCO job
```

```
last_esco_jobs = similarity_df_bert.loc[:, last_anzsco_jobs].idxmax(axis=0)  # Get ESCO jobs with least similarity
# Subset the similarity matrix for these pairs
least_similarity_bert = similarity_df_bert.loc[last_esco_jobs, top_anzsco_jobs]
# Display Top Similar Occupations
least_matches_bert = similarity_df_bert.min(axis=1).sort_values(ascending=True).head(10)
print("\nLeast Similar Occupations (BERT):\n", least_matches_bert)
```

```
Top Matching Occupations (BERT):
 ANZSCO Title
Checkout Operators and Office Cashiers    0.932330
ICT Trainer                               0.902493
Life Coach                                0.863886
Switchboard Operators                     0.860497
ICT Account Manager                       0.856525
Market Research Interviewer               0.844147
Blow Moulding Machine Operator            0.843342
Instructional Designer                    0.839717
Systems Auditor (ICT)                     0.837789
Metal Rolling Mill Operator               0.837192
dtype: float32

Least Similar Occupations (BERT):
 ANZSCO Title
Sugar Cane Grower                                 -0.252778
Osteopath                                         -0.226633
Podiatrist                                        -0.226263
Glass Blower                                      -0.223286
Library Assistant                                 -0.214297
Stagehand                                         -0.211254
Logging Assistant                                 -0.203914
Podiatrists                                       -0.203764
Aircraft Baggage Handler and Airline Ground Crew  -0.202565
Music Researcher                                  -0.199997
dtype: float32
```

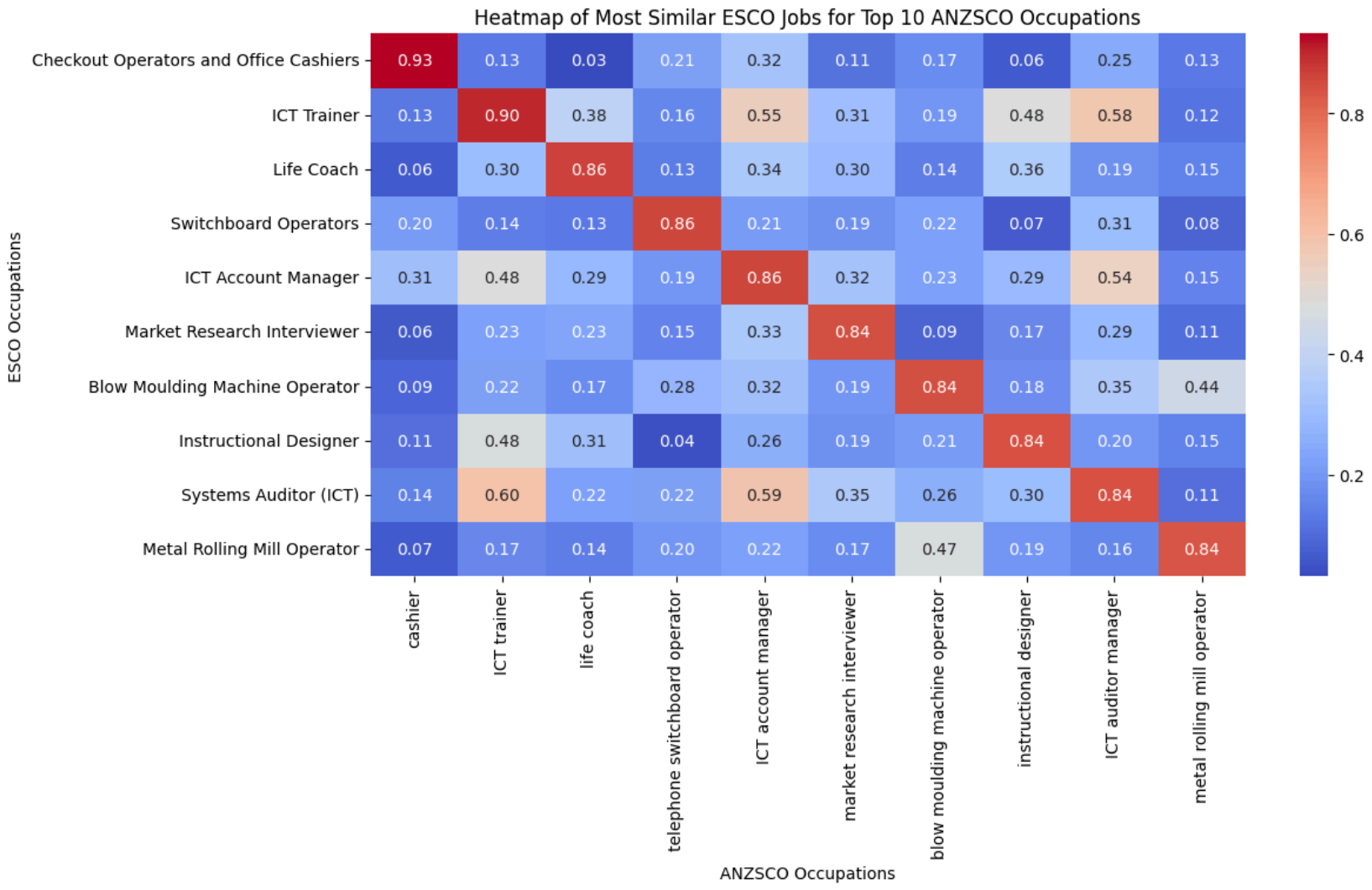
```
# Bar plot for Top Matches
top_matches_bert.plot(kind="barh", figsize=(10, 6), color="green")
plt.xlabel("Cosine Similarity Score")
plt.ylabel("ESCO Occupations")
plt.title("Top 10 Most Similar Occupations (ESCO vs ANZSCO)")
plt.gca().invert_yaxis()  # Invert y-axis for better readability
```




```
# Find top 10 ANZSCO occupations with the highest similarity scores
top_anzsco_jobs = similarity_df_bert.max(axis=0).nlargest(10).index  # Get top 10 ANZSCO jobs

# Find the most similar ESCO job for each ANZSCO job
top_esco_jobs = similarity_df_bert.loc[:, top_anzsco_jobs].idxmax(axis=0)  # Get ESCO jobs with highest similarity
```

```
# Subset the similarity matrix for these pairs
top_similarity_bert = similarity_df_bert.loc[top_esco_jobs, top_anzsco_jobs]

# Plot the heatmap
plt.figure(figsize=(12, 6))
sns.heatmap(top_similarity_bert, annot=True, fmt=".2f", cmap="coolwarm",
            xticklabels=top_anzsco_jobs, yticklabels=top_esco_jobs)
plt.title("Heatmap of Most Similar ESCO Jobs for Top 10 ANZSCO Occupations")
plt.xlabel("ANZSCO Occupations")
plt.ylabel("ESCO Occupations")
plt.xticks(rotation=90)
```

```
(array([0.5, 1.5, 2.5, 3.5, 4.5, 5.5, 6.5, 7.5, 8.5, 9.5]),
 [Text(0.5, 0, 'cashier'),
  Text(1.5, 0, 'ICT trainer'),
  Text(2.5, 0, 'life coach'),
  Text(3.5, 0, 'telephone switchboard operator'),
  Text(4.5, 0, 'ICT account manager'),
  Text(5.5, 0, 'market research interviewer'),
  Text(6.5, 0, 'blow moulding machine operator'),
  Text(7.5, 0, 'instructional designer'),
  Text(8.5, 0, 'ICT auditor manager'),
  Text(9.5, 0, 'metal rolling mill operator')])
```



OBSERVATION:

- Top Matching Occupations (BERT):
  - Occupations like Checkout Operators and Office Cashiers, ICT Trainer, and Life Coach show high similarity, indicating that these roles share common skill sets or job descriptions, particularly in customer service, training, or communication-focused sectors.
  - Jobs in similar domains such as ICT, market research, and instructional design have overlapping professional terminology and requirements, reflecting a trend in service-oriented and tech-related occupations.
- Least Similar Occupations (BERT):
  - Occupations like Sugar Cane Grower, Osteopath, and Podiatrist have low similarity scores, reflecting a clear distinction between these roles and others in the dataset, likely due to their specialized and niche nature.
  - The significant differences in job functions and technical language in occupations such as Glass Blower and Logging Assistant show minimal overlap with other categories, indicating they require highly specialized knowledge and skills not shared with more general roles.
- Top matches demonstrate roles within service and tech domains, sharing broad skill sets and common terminologies, while least matches highlight specialized professions with distinct job characteristics, suggesting less cross-over between them in terms of skills and language.

Final Insights:

- High similarity across top occupations suggests a tendency for roles within the same industry or service area to share common skills, terminology, and job responsibilities, particularly in manual labor and service-oriented sectors.
- Low similarity highlights that specialized roles, whether in leadership, healthcare, or technical domains, operate with unique skill sets and vocabulary that are not broadly applicable to other job categories, indicating that these professions require specialized knowledge that is not shared with generalist roles.

11. **Key Findings:**

1. Healthcare & Medical Dominance:

   Both TF-IDF and BERT analyses highlighted the prominence of healthcare-related roles, with terms like "health," "patient," "care," and "treat" frequently appearing in occupations. This underscores the growing demand for medical professionals in Australia.

2. Strong Engineering & Technical Sector:

   Engineering, technology, and manufacturing roles are well-represented, with keywords like "machine," "repair," and "equipment" indicating a focus on technical and industrial expertise. The Topic Modeling and n-gram analysis also identified roles requiring advanced tools and machinery skills.

3. Educational and Research Roles:

   The n-gram and Topic Modeling analyses point to the significance of educational and research-based occupations, emphasizing the role of teaching and training in various industries, particularly in fields like engineering and technology.

4. Specialization and Sector-Specific Skills:

   TF-IDF analysis revealed a clear emphasis on specialized knowledge, especially in technical, healthcare, and managerial roles. These positions often require distinct skill sets and terminologies, reflecting Australia's need for expertise in niche areas like healthcare,

engineering, and operations.

5. Diverse Clusters:

   The clustering analysis revealed a clear separation between roles in terms of responsibilities and required expertise. While Cluster 0 and Cluster 1 showed strong overlaps in technical, operational, and financial roles, clusters like Cluster 3 and Cluster 4 demonstrated leadership and strategic roles, highlighting the diversity within the dataset. The Silhouette Score of 0.0285 suggests significant overlap across clusters, possibly due to a variety of similar job characteristics.

6. Similarity Patterns:

   High similarity was found among roles in similar sectors (e.g., manual labor and customer service), while low similarity highlighted distinct and specialized roles (e.g., Osteopath, Podiatrist) that require unique knowledge and skills.

7. Emerging Trends:

   The appearance of terms like "creative talent" and "high-level creative" in the n-gram and topic modeling analyses points to the increasing importance of creative skills, especially in sectors such as education, technology, and research.

12. **Future Skills for Data Scientists**

```
data_df=occ_df[occ_df['ANZSCO Title'].str.lower() == 'data scientist']

data_df = pd.concat([data_df, esco_df[esco_df['preferredLabel'].str.lower() == 'data scientist']])

tokens=[word_tokenize(desc) for desc in data_df["cleaned_description"]]

common_words = Counter([word for sublist in tokens for word in sublist])

print("Most Common Words:")
for i,j in common_words.most_common(20):
    print(i)
```

```
# Function to generate bigrams and their frequencies
def generate_bigrams(tokens):
    bigram_counts = Counter()
    for token_list in tokens:
        for bigram in bigrams(token_list):
          bigram_counts[bigram] +=1
    return bigram_counts

bigram_counts = generate_bigrams(tokens)

# Print the most common bigrams
print("\nMost Common Bigrams:")
for bigram, count in bigram_counts.most_common(20):  # Display the top 20 bigrams
    print(" ".join(bigram))
```

```
Most Common Bigrams:
data source
applies analytical
analytical technique
technique scientific
scientific procedure
procedure large
large datasets
datasets creating
creating advanced
advanced algorithm
algorithm model
model build
build deploys
deploys machine
machine learning
learning framework
framework obtain
obtain information
information strategic
strategic planning
```

Findings:

- Advanced Analytical Skills: Proficiency in advanced statistical methods and machine learning techniques to extract insights from data.
- Machine Learning & AI: Expertise in frameworks like TensorFlow and PyTorch for building and deploying machine learning models.
- Big Data Management: Ability to handle and analyze large datasets using tools like Hadoop and Spark.
- Strategic Thinking: Understanding how to use data for informed decision-making and business strategy.
- Model Deployment: Skills in deploying and automating models using CI/CD pipelines.

13. **Conclusion:**

This project provides a comprehensive analysis of the Australian occupation landscape using techniques such as TF-IDF, BERT, word frequency analysis, topic modeling, and clustering. The findings highlight several key trends and insights:

1. **Dominance of Healthcare and Technical Sectors:** There is a strong demand for specialized roles, particularly in healthcare, engineering, and technology.
2. **Overlap in Similar Sectors:** Roles within similar sectors, such as manual labor and service-oriented jobs, exhibit common terminology and skills, while highly specialized professions (e.g., Osteopath, Podiatrist) are distinct due to their unique terminology and skill sets.
3. **Emerging Need for Creative, Educational, and Technological Skills:** Increasingly, professions are placing emphasis on creativity and advanced cognitive skills, as indicated by terms like "creative talent" and "high-level creative."
4. **Training and Skill Gaps:** The analysis underscores the need for tailored training programs to address skill gaps in leadership, technical expertise, and creative competencies, ensuring professionals are prepared for future demands.
5. **Future Skills for Data Scientists:** Future data scientists will need advanced analytical skills, proficiency in machine learning and AI frameworks, big data management expertise, strategic thinking, and model deployment capabilities to meet the evolving demands of the field.

Future Works:

1. **Further Dataset Analysis**: We plan to analyze additional datasets identified during the exploratory data analysis (EDA) to gain deeper insights into the occupation landscape and skill requirements.
2. **Global Skill Requirements**: We will conduct similarity analysis across regions such as Europe, America, the UK, and others to better understand the global skill demands, particularly focusing on data science roles and the global evolution of skills.
3. **Clustering Model Improvement**: We aim to refine our clustering models by exploring techniques such as hierarchical clustering and DBSCAN to enhance the grouping and improve the Silhouette Score for better cluster evaluation.