
Unsupervised Learning for User Clustering

For this project, each team member has been assigned a specific objective. I, Jie Luo (jiemluo), am focused on Unsupervised Learning, where my task is to identify patterns and clusters among users. These insights will help us recommend personalized learning paths tailored to the unique needs and behaviors of each user.

In this notebook, I will perform unsupervised learning to identify common patterns and clusters among users. This can help recommend personalized learning paths for new users.

Table of Contents

1. Load the Raw Dataset
2. Data Processing
 - Encode Categorical Variables
 - Feature Scaling
3. K-Means Clustering
 - Evaluating optimal number of clusters
 - Perform K-Means Clustering on Original Scaled Data
 - Applying Dimensionality Reduction with PCA
 - Evaluation by calculating Silhouette Scores
 - Applying t-SNE with Clustering Visualization
4. Other Clustering Algorithms
 - Gaussian Mixture Models (GMMs)
 - Hierarchical Agglomerative Clustering
 - DBSCAN
 - Applying t-SNE with Clustering Visualization
5. Evaluation Compare all the models implemented with Silhouette Score
6. Cluster Analysis
 - Analyzing clusters based on feature means

▼ Part I. Load the Raw Dataset

```
# Import necessary libraries
import pandas as pd
import numpy as np

# For data preprocessing
from sklearn.preprocessing import StandardScaler, LabelEncoder

# For clustering algorithms
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.mixture import GaussianMixture

# For dimensionality reduction
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE

# For evaluation metrics
from sklearn.metrics import silhouette_score

# For visualization
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
# Ignore warnings for cleaner output
import warnings
warnings.filterwarnings('ignore')
```

```
# Upload the final.csv file from local
from google.colab import files
uploaded = files.upload()
```



Choose Files df_final.csv

- **df_final.csv**(text/csv) - 1629450 bytes, last modified: 10/13/2024 - 100% done
Saving df_final.csv to df_final.csv

```
# Load the CSV into a pandas DataFrame
df = pd.read_csv('df_final.csv')
```

```
# Check the unique column names
unique_columns = set(df.columns)
unique_columns
```



```
{'Age',
 'Count_Learning_Methods',
 'Count_Online_Resources',
 'Employment_Status',
 'Field_Working_Education',
 'Field_Working_Others',
 'Field_Working_Self-employed',
 'Field_Working_Software development and IT',
 'Field_Working_unemployed',
 'High_Expectation',
 'Highest_Degree_Ordinal',
 'Hours_Learning_Weekly',
 'In-person Events',
 'Income',
 'Industry_Experience',
 'Intuition_Encoded',
 'Job_Status_Expectation(Objective2)',
 'Job_Status_Income(Objective1)',
 'Laid_Off_Potential',
 'Listen_Podcasts',
 'Money_Spent',
 'Months_Finding_New_Job',
 'Months_Programming',
 'Replacable_Job_Potential',
 'Study_Field_Computer-related',
 'Study_Field_Not applicable',
 'Study_Field_Other Science & Engineering',
 'Study_Field_Others',
 'Youtube_Channels'}
```

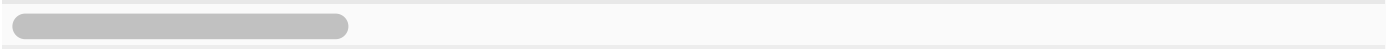
```
# Drop the target columns for supervised learning objectives from the DataFrame
df = df.drop(['Job_Status_Expectation(Objective2)', 'Job_Status_Income(Objective1)'], axis=1)
```

```
# Display the first three rows of the DataFrame
df.head(3)
```



	Intuition_Encoded	Age	Income	Employment_Status	High_Expectation	Industry_Experience	Highest_De
0	3.0	35.0	2.0	1.0	1.0	1.0	
1	3.0	27.0	2.0	1.0	1.0	1.0	
2	3.0	24.0	2.0	1.0	1.0	1.0	

3 rows × 27 columns



▼ Part II. Data Processing

- Encode Categorical Variables
- Feature Scaling

Encode Categorical Variables

Note: Normally, encoding categorical variables is necessary because clustering algorithms like K-Means and Gaussian Mixture Models rely on numerical data for distance calculations (e.g., Euclidean distance). Categorical variables, especially nominal categories like 'Employment_Status' with values such as 'Employed', 'Unemployed', etc., cannot be directly used in these computations. However, since you've already performed one-hot encoding or ordinal encoding on your categorical variables, and your DataFrame df now consists entirely of numerical columns, we can skip the 'Encode Categorical Variables' step in the data preprocessing.

Feature Scaling

Clustering algorithms like K-Means use distance metrics to assign data points to clusters. If one feature has a much larger scale than others, it can disproportionately influence the distance calculations. In this case, we will need to adjust the data to a common scale. The way we are using is by scaling transforms features to have a mean of zero and a standard deviation of one.

```
# Feature Scaling
scaler = StandardScaler()
scaled_features = scaler.fit_transform(df)

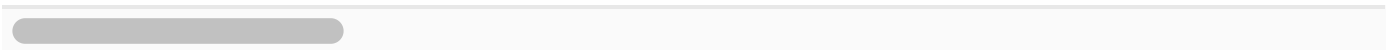
# Create a DataFrame with scaled features (optional)
scaled_df = pd.DataFrame(scaled_features, columns=df.columns)

# Display the first three rows of the DataFrame
scaled_df.head(3)
```



	Intuition_Encoded	Age	Income	Employment_Status	High_Expectation	Industry_Experience	Highest_De
0	0.434753	0.681805	1.183217	0.902157	0.728434	2.807225	
1	0.434753	-0.024449	1.183217	0.902157	0.728434	2.807225	
2	0.434753	-0.289295	1.183217	0.902157	0.728434	2.807225	

3 rows × 27 columns



✓ Part III. K-Means Clustering

- Evaluating optimal number of clusters
- Perform K-Means Clustering on Original Scaled Data
- Applying Dimensionality Reduction with PCA
- Applying Dimensionality Reduction with t-SNE
- Evaluation by calculating Silhouette Scores

Evaluating optimal number of clusters

```
# Function to plot the Elbow Method and Silhouette Scores on the same plot with two y-axes
def evaluate_clustering(scaled_data, max_k=10):
    inertia = []
    silhouette_scores = []
    K = range(2, max_k + 1)

    for k in K:
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(scaled_data)
        inertia.append(kmeans.inertia_)
        labels = kmeans.labels_
        silhouette = silhouette_score(scaled_data, labels)
        silhouette_scores.append(silhouette)

    # Create a figure with a shared x-axis
    fig, ax1 = plt.subplots(figsize=(6, 3))

    # Plot Inertia (Elbow Method) on the left y-axis
    ax1.plot(K, inertia, 'bx-', label='Inertia (Elbow Method)', lw=1, markersize=3)
    ax1.set_xlabel('Number of clusters (k)', fontsize=9)
    ax1.set_ylabel('Inertia', color='darkblue', fontsize=9)
    ax1.tick_params(axis='y', labelcolor='darkblue')

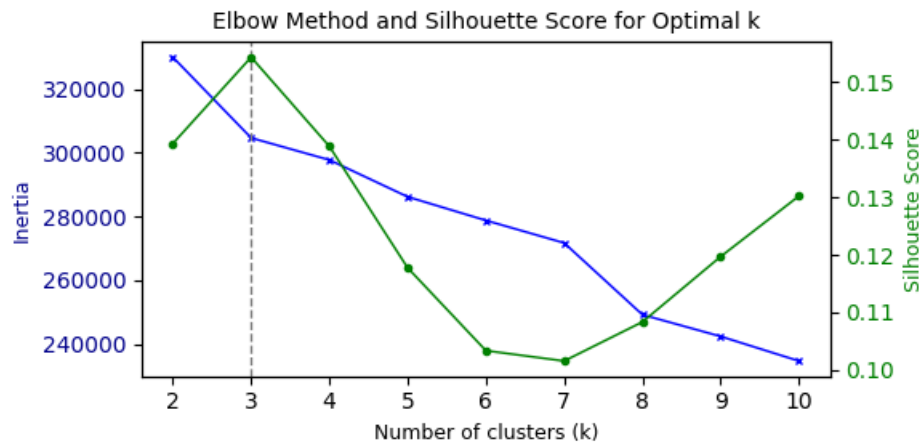
    # Create a second y-axis to plot Silhouette Score
    ax2 = ax1.twinx()
    ax2.plot(K, silhouette_scores, 'go-', label='Silhouette Score', lw=1, markersize=3)
    ax2.set_ylabel('Silhouette Score', color='green', fontsize=9)
    ax2.tick_params(axis='y', labelcolor='green')

    # Add a vertical dashed line at cluster = 3
    ax1.axvline(x=3, color='grey', linestyle='--', lw=1)

    # Add a title to the plot
    plt.title('Elbow Method and Silhouette Score for Optimal k', fontsize=10)

    # Display the plot
    plt.tight_layout()
    plt.show()

# Evaluate clustering to find optimal k
evaluate_clustering(scaled_features, max_k=10)
```



Insights:

- Elbow Method Plot: The elbow point (the point where the rate of decrease in inertia slows down significantly) is around $k = 3$ or $k = 4$. This suggests that after 3 clusters, the additional gain in clustering quality decreases.
- Silhouette Score Plot: The silhouette score peaks at $k = 3$, indicating that this is the most optimal number of clusters in terms of how well-separated the clusters are. A higher silhouette score represents better-defined clusters.
- Based on both the silhouette score and elbow method, **the optimal number of clusters appears to be $k = 3$** . This provides the best trade-off between well-defined clusters (from the silhouette score) and reduced inertia (from the elbow method).

```
# Define the optimal_k as 3
optimal_k = 3
```

Perform K-Means Clustering on Original Scaled Data

```
# Clustering on original scaled data
kmeans_original = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_original_labels = kmeans_original.fit_predict(scaled_features)

# Calculate Silhouette Score
silhouette_original = silhouette_score(scaled_features, kmeans_original_labels)
print(f'Silhouette Score for K-Means on original data: {silhouette_original}')
```



Silhouette Score for K-Means on original data: 0.15437308497509747

Applying PCA

We will start with reduce dimensions to 2 by applying PCA.

```
# Apply PCA to reduce dimensions to 2 for visualization
pca = PCA(n_components=2, random_state=42)
pca_components = pca.fit_transform(scaled_features)

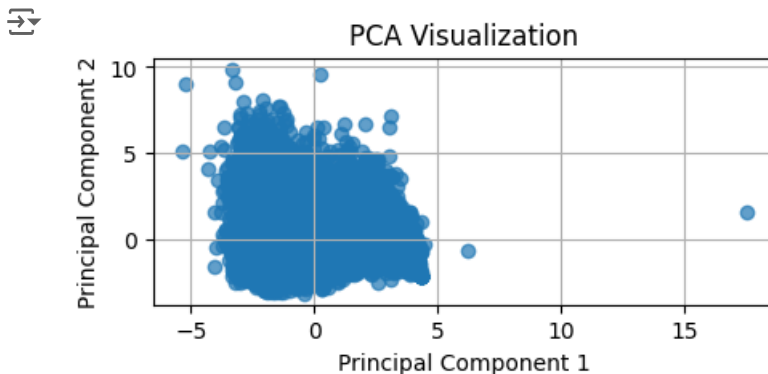
# Create a DataFrame with the principal components
pca_df = pd.DataFrame(data=pca_components, columns=['PC1', 'PC2'])

# Check the amount of variance explained by each principal component
explained_variance = pca.explained_variance_ratio_
```

```
print(f"Variance explained by PC1: {explained_variance[0]:.2%}")
print(f"Variance explained by PC2: {explained_variance[1]:.2%}")
print(f"Total variance explained by the first two PCs: {explained_variance.sum():.2%}")
```

```
➡ Variance explained by PC1: 14.51%
   Variance explained by PC2: 10.28%
   Total variance explained by the first two PCs: 24.79%
```

```
# Plot the PCA results
plt.figure(figsize=(5, 2))
plt.scatter(pca_df['PC1'], pca_df['PC2'], alpha=0.7)
plt.title('PCA Visualization')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.grid(True)
plt.show()
```

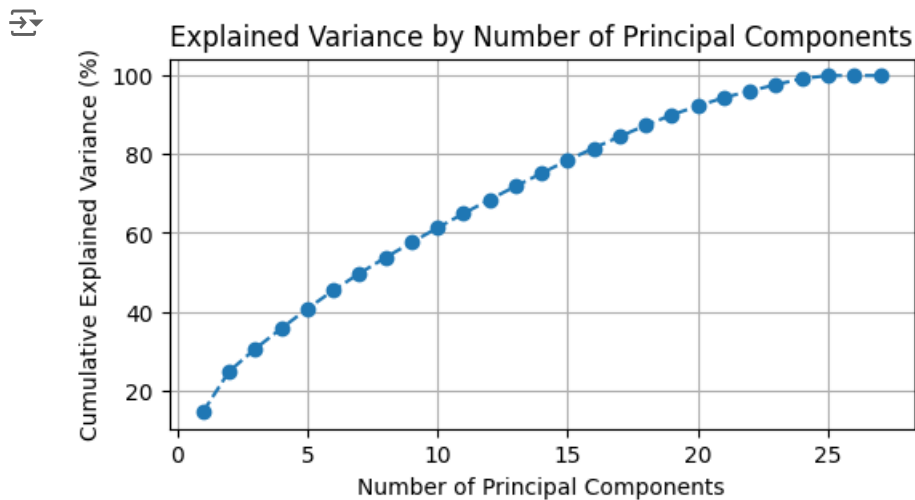


Insights:

- **Information Loss:** Retaining only ~25% of the variance suggests that a significant amount of information is lost when reducing the data to two dimensions. This may limit the effectiveness of visualizations and could obscure underlying patterns or clusters in the data.
- **Data Complexity:** The relatively low variance captured by the first two components indicates that the variance is spread out across many features. Important information is distributed among several principal components rather than being concentrated in the first few.

To get a better usage of PCA, I will increase the number of principal components and try to capture a sufficient amount of the total variance.

```
# Generate cumulative explained variance plot
pca_full = PCA().fit(scaled_features)
cumulative_variance = np.cumsum(pca_full.explained_variance_ratio_) * 100
plt.figure(figsize=(6, 3))
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o', linestyle='--')
plt.xlabel('Number of Principal Components')
plt.ylabel('Cumulative Explained Variance (%)')
plt.title('Explained Variance by Number of Principal Components')
plt.grid(True)
plt.show()
```



Insights:

After around 15 components, the curve flattens, indicating diminishing returns as additional components contribute less new information. By selecting around 15 components, we can capture approximately 80 of the variance, striking a balance between dimensionality reduction and retaining sufficient variance for meaningful analysis, while avoiding unnecessary complexity from including too many components.

```
# Reduce dimensions to 15 components
pca_n = PCA(n_components=15, random_state=42)
pca_components_n = pca_n.fit_transform(scaled_features)
```

```
# Access the explained variance ratio
explained_variance_ratio_15 = pca_n.explained_variance_ratio_
```

```
# Calculate cumulative explained variance
cumulative_variance_15 = np.cumsum(explained_variance_ratio_15)
```

```
# Get the cumulative variance explained by the first 15 components
total_cumulative_variance = cumulative_variance_15[-1] * 100 # Multiply by 100 to get percentage
```

```
# Print the result
print(f"The cumulative variance for n_components=15 is {total_cumulative_variance:.2f}%")
```

➡ The cumulative variance for n_components=15 is 78.48%

Insights:

78.48% variance explained with 15 components is generally acceptable.

```
# Perform K-Means clustering on the PCA-reduced data
kmeans_pca_15 = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_pca_labels_15 = kmeans_pca_15.fit_predict(pca_components_n)
```

```
# Calculate Silhouette Score
silhouette_pca_15 = silhouette_score(pca_components_n, kmeans_pca_labels_15)
print(f'Silhouette Score for K-Means on PCA-reduced data (15 components): {silhouette_pca_15}')
```

➡ Silhouette Score for K-Means on PCA-reduced data (15 components): 0.20081165896748351

Insights:

The **silhouette score** measures how well clusters are separated and how similar the points within a cluster are. It ranges from -1 to 1. A score of **0.2008** is relatively low, indicating that the clusters are not well-separated. Compared to **the Silhouette Score for K-Means on the original data (0.1544)**, PCA has provided some improvement, but the silhouette score is still low, meaning the dimensionality reduction hasn't significantly enhanced the clustering. To gain further insights and visualize the local relationships between data points, I want to implement t-SNE, as it is more effective at preserving local structures in high-dimensional data, which may reveal patterns that PCA and K-Means miss.

Sensitivity analysis

We also *would* like to implement sensitivity analysis because it is important to understand how robust the K-Means clustering results are to variations in key parameters, such as the number of PCA components or the choice of features. By analyzing how changes in these factors affect the clustering performance, we can ensure that our model is stable and reliable, avoiding overfitting to a specific configuration. This analysis also helps identify the most critical factors influencing the results, allowing us to make more informed decisions for optimizing the model.

```
from sklearn.decomposition import PCA

def sensitivity_analysis_pca(scaled_data, max_pca=20, max_k=5):
    silhouette_scores = {}

    # Loop over different numbers of PCA components
    for n_components in range(5, max_pca + 1, 5): # Testing PCA components in steps of 5
        pca = PCA(n_components=n_components)
        pca_data = pca.fit_transform(scaled_data)

        inertia = []
        silhouette_scores_pca = []
        K = range(2, max_k + 1)

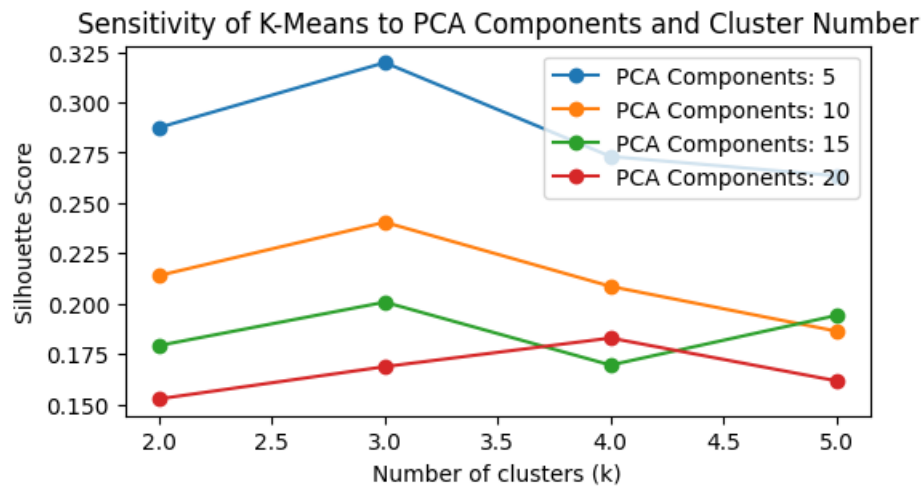
        # Loop over different number of clusters (k)
        for k in K:
            kmeans = KMeans(n_clusters=k, random_state=42)
            kmeans.fit(pca_data)
            inertia.append(kmeans.inertia_)
            labels = kmeans.labels_
            silhouette = silhouette_score(pca_data, labels)
            silhouette_scores_pca.append(silhouette)

        silhouette_scores[n_components] = silhouette_scores_pca

    # Plotting Silhouette Scores for different PCA components
    fig, ax = plt.subplots(figsize=(6, 3))
    for n_components, scores in silhouette_scores.items():
        ax.plot(K, scores, marker='o', label=f'PCA Components: {n_components}')

    ax.set_xlabel('Number of clusters (k)')
    ax.set_ylabel('Silhouette Score')
    ax.set_title('Sensitivity of K-Means to PCA Components and Cluster Number')
    ax.legend()
    plt.show()

# Run sensitivity analysis
sensitivity_analysis_pca(scaled_features, max_pca=20, max_k=5)
```

Insights:

The plot illustrates the sensitivity of the Silhouette Score for K-Means clustering to different numbers of PCA components and clusters (k). We observe that when using 5 PCA components, the clustering achieves the highest Silhouette Score at $k = 3$, indicating that this configuration results in the best-defined clusters. As the number of PCA components increases, the overall Silhouette Score tends to decline, especially when 20 components are used, showing that retaining too many components may negatively affect cluster quality. Across all tested configurations, $k = 3$ consistently provides better clustering performance, although the magnitude of improvement varies depending on the number of PCA components. This suggests that both $k = 3$ and a reduced number of PCA components (5 to 10) are optimal for obtaining more distinct clusters in this dataset.

```
# Reduce dimensions to 5 components
pca_n = PCA(n_components=5, random_state=42)
pca_components_n = pca_n.fit_transform(scaled_features)

# Perform K-Means clustering on the PCA-reduced data
kmeans_pca_5 = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_pca_labels_5 = kmeans_pca_5.fit_predict(pca_components_n)

# Calculate Silhouette Score
silhouette_pca_5 = silhouette_score(pca_components_n, kmeans_pca_labels_5)
print(f'Silhouette Score for K-Means on PCA-reduced data (5 components): {silhouette_pca_5}')
```



Silhouette Score for K-Means on PCA-reduced data (5 components): 0.31973319677790896

Applying t-SNE

Alternatively, we can apply t-Distributed Stochastic Neighbor Embedding (t-SNE).

```
# Perform t-SNE on the original scaled data
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2, perplexity=30, n_iter=1000, random_state=42)
tsne_components = tsne.fit_transform(scaled_features)

# Create DataFrame
tsne_df = pd.DataFrame(tsne_components, columns=['TSNE1', 'TSNE2'])

# Perform K-Means clustering on the original scaled data
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
```

```

kmeans_labels = kmeans.fit_predict(scaled_features)

# Add cluster labels to the DataFrame
tsne_df['Cluster'] = kmeans_labels

# Perform t-SNE on the PCA-reduced data
tsne_pca = TSNE(n_components=2, perplexity=30, n_iter=1000, random_state=42)
tsne_pca_components = tsne_pca.fit_transform(pca_components_n)

# Create DataFrame
tsne_pca_df = pd.DataFrame(tsne_pca_components, columns=['TSNE1', 'TSNE2'])

# Perform K-Means clustering on the PCA-reduced data
kmeans_pca = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_pca_labels = kmeans_pca.fit_predict(pca_components_n)

# Add cluster labels to the DataFrame
tsne_pca_df['Cluster'] = kmeans_pca_labels

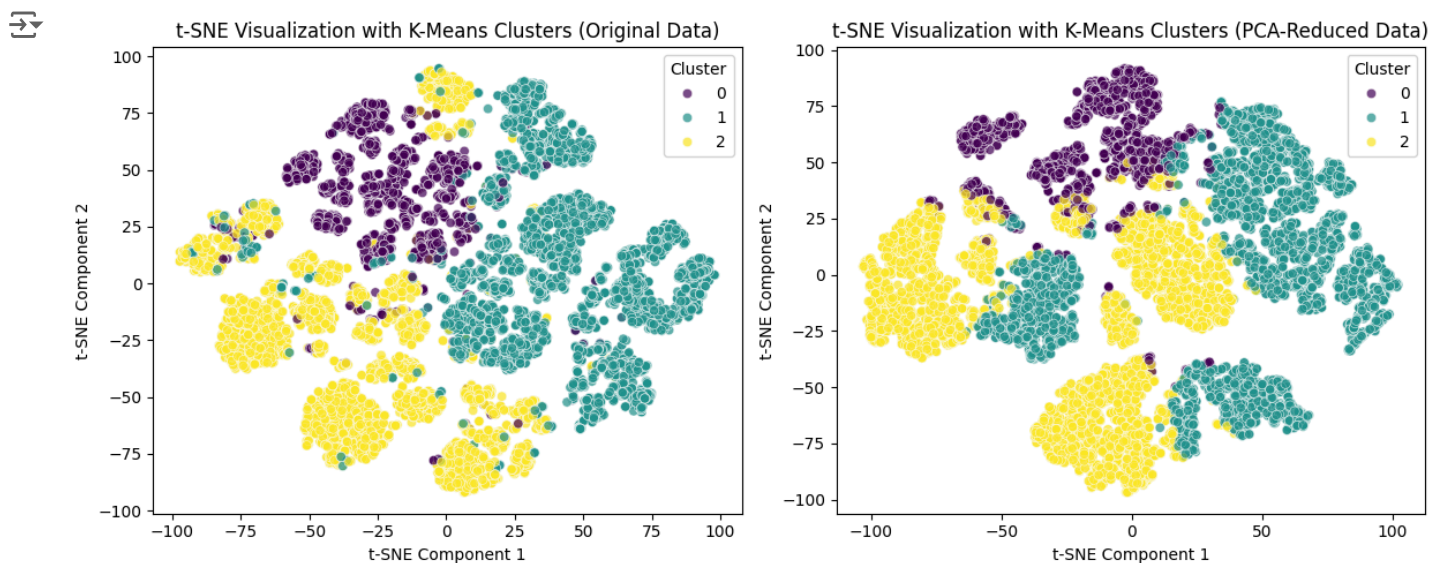
# Create subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 5))

# Plot 1: t-SNE Visualization on Original Scaled Data
sns.scatterplot(
    data=tsne_df,
    x='TSNE1',
    y='TSNE2',
    hue='Cluster',
    palette='viridis',
    alpha=0.7,
    ax=ax1
)
ax1.set_title('t-SNE Visualization with K-Means Clusters (Original Data)')
ax1.set_xlabel('t-SNE Component 1')
ax1.set_ylabel('t-SNE Component 2')
ax1.legend(title='Cluster')

# Plot 2: t-SNE Visualization on PCA-Reduced Data
sns.scatterplot(
    data=tsne_pca_df,
    x='TSNE1',
    y='TSNE2',
    hue='Cluster',
    palette='viridis',
    alpha=0.7,
    ax=ax2
)
ax2.set_title('t-SNE Visualization with K-Means Clusters (PCA-Reduced Data)')
ax2.set_xlabel('t-SNE Component 1')
ax2.set_ylabel('t-SNE Component 2')
ax2.legend(title='Cluster')

# Adjust layout
plt.tight_layout()
plt.show()

```



Insights:

After implementing t-SNE on both the original and PCA-reduced data, we can compare the clustering results side by side. In both visualizations, the teal (Cluster 2) and purple (Cluster 0) clusters remain relatively well-defined, though some overlap is still present, particularly with yellow (Cluster 1). Interestingly, the t-SNE plot on the PCA-reduced data (right) shows a more fragmented structure, where the clusters appear more spread out and disjointed compared to the original data (left), where the clusters are more compact. This fragmentation could indicate that while PCA helped reduce dimensionality, it may have discarded some subtle variations in the data that were important for defining cluster boundaries. Overall, t-SNE is effective in visualizing local patterns, but the PCA-reduced t-SNE plot suggests that using fewer components may not always yield clearer separations.

Next Step:

I am going to implement other clustering algorithms like Gaussian Mixture Models (GMMs), Hierarchical Agglomerative Clustering, and DBSCAN. While K-Means is a widely used and straightforward clustering algorithm, it makes certain assumptions about the data that may not always hold true. Exploring different clustering methods can provide additional insights.

✓ Part IV. Other Clustering Algorithms

- Gaussian Mixture Models (GMMs)
- Hierarchical Agglomerative Clustering
- DBSCAN
- Applying t-SNE with Clustering Visualization

Gaussian Mixture Models (GMMs)

- GMMs are probabilistic models that assume the data is generated from a mixture of several Gaussian distributions with unknown parameters.
- Unlike K-Means, which assigns each point to the nearest cluster center, GMMs assign probabilities to each point belonging to each cluster.

```
from sklearn.mixture import GaussianMixture

# Decide on the number of components (clusters)
n_components = optimal_k # Use the same k as in K-Means or explore different values

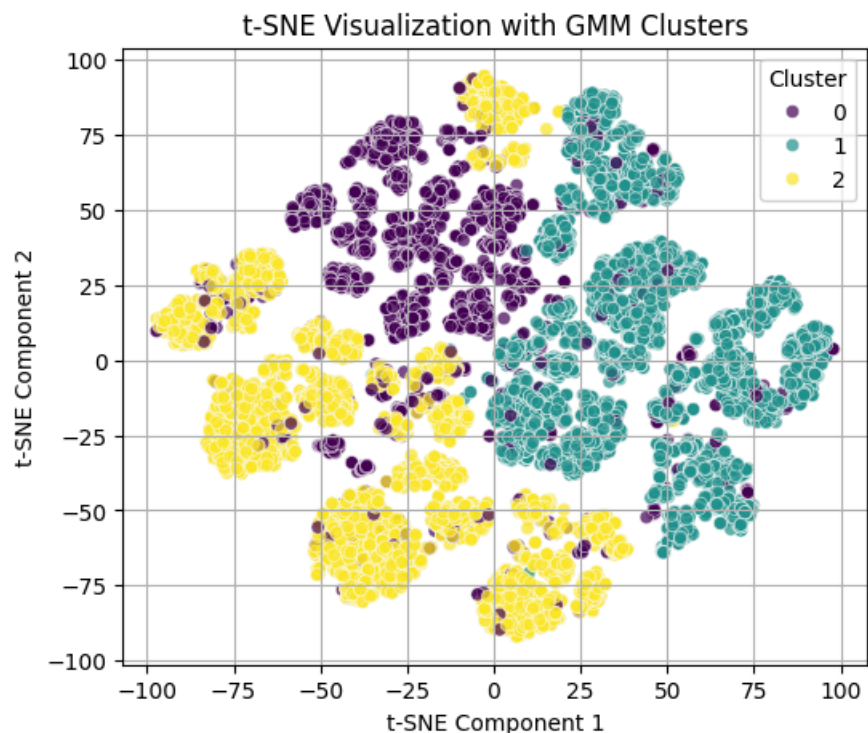
# Fit the GMM
gmm = GaussianMixture(n_components=n_components, covariance_type='full', random_state=42)
gmm_labels = gmm.fit_predict(scaled_features)

# Calculate Silhouette Score
from sklearn.metrics import silhouette_score
silhouette_gmm = silhouette_score(scaled_features, gmm_labels)
print(f'Silhouette Score for GMM: {silhouette_gmm:.4f}')

# Visualization (e.g., using t-SNE)
tsne_df['GMM_Cluster'] = gmm_labels

➡ Silhouette Score for GMM: 0.1371

# Plot the t-SNE results with GMM cluster labels
plt.figure(figsize=(6, 5))
sns.scatterplot(
    data=tsne_df,
    x='TSNE1',
    y='TSNE2',
    hue='GMM_Cluster',
    palette='viridis',
    alpha=0.7
)
plt.title('t-SNE Visualization with GMM Clusters')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()
```



Insights:

The Silhouette Score for GMM (Gaussian Mixture Model) is 0.1371, which is relatively low, indicating that the clustering results are not well-separated or cohesive. In the t-SNE visualization, we can observe that the teal (Cluster 2) and purple (Cluster 0) clusters have significant overlap with the yellow (Cluster 1), which suggests that GMM struggled to form clear, distinct boundaries between the clusters. GMM uses a probabilistic approach to assign points to clusters, which could lead to more flexibility in cluster assignments, but in this case, it results in ambiguity and poor separation.

Hierarchical Agglomerative Clustering

- Builds a hierarchy of clusters by recursively merging or splitting clusters.

```
from sklearn.cluster import AgglomerativeClustering

# Fit the model
agglo = AgglomerativeClustering(n_clusters=optimal_k, metric='euclidean', linkage='ward')
agglo_labels = agglo.fit_predict(scaled_features)

# Calculate Silhouette Score
silhouette_agglo = silhouette_score(scaled_features, agglo_labels)
print(f'Silhouette Score for Agglomerative Clustering: {silhouette_agglo:.4f}')

# Visualization
tsne_df['Agglo_Cluster'] = agglo_labels
```



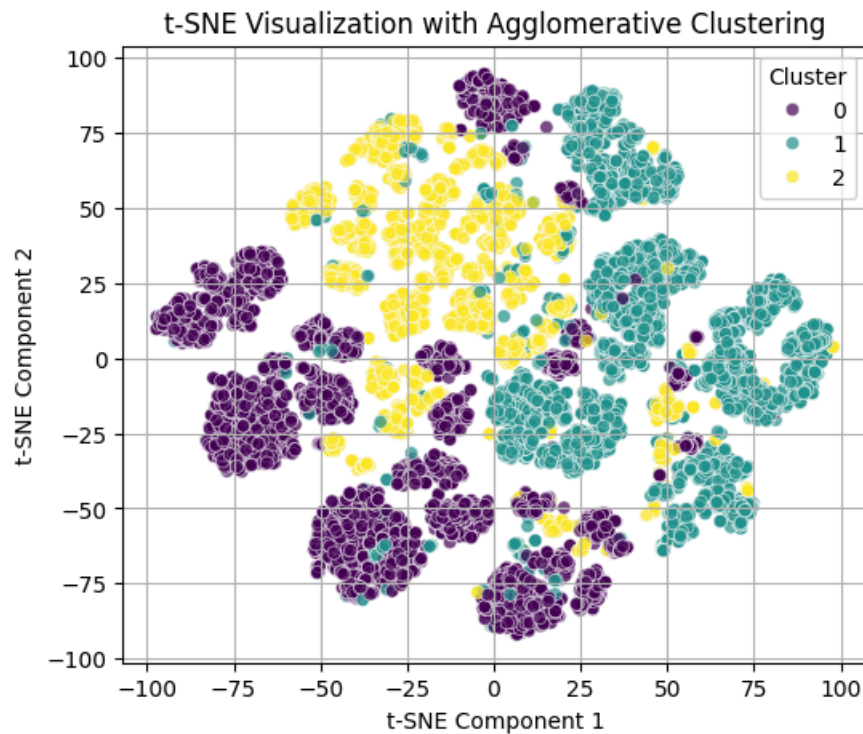
Silhouette Score for Agglomerative Clustering: 0.1201

```
# Plot the t-SNE results with Agglomerative Clustering labels
plt.figure(figsize=(6, 5))
sns.scatterplot(
    data=tsne_df,
    x='TSNE1',
    y='TSNE2',
```

```

    hue='Agglo_Cluster',
    palette='viridis',
    alpha=0.7
)
plt.title('t-SNE Visualization with Agglomerative Clustering')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.legend(title='Cluster')
plt.grid(True)
plt.show()

```



Insights:

The Silhouette Score for Agglomerative Clustering is 0.1201, which is relatively low, indicating poor cluster separation and cohesion. From the t-SNE visualization, we can see that the clusters have significant overlap, especially between the yellow (Cluster 1) and the other clusters (purple and teal). This suggests that Agglomerative Clustering has struggled to clearly differentiate between the data points. The low silhouette score confirms that many points are near the boundaries of their clusters or may have been misclassified. Overall, while Agglomerative Clustering may identify some structure in the data, the results here show that it does not perform well on this dataset, and further adjustments or different algorithms may yield better results.

```

import scipy.cluster.hierarchy as shc

# Generate the linkage matrix
linkage_matrix = shc.linkage(scaled_features, method='ward')

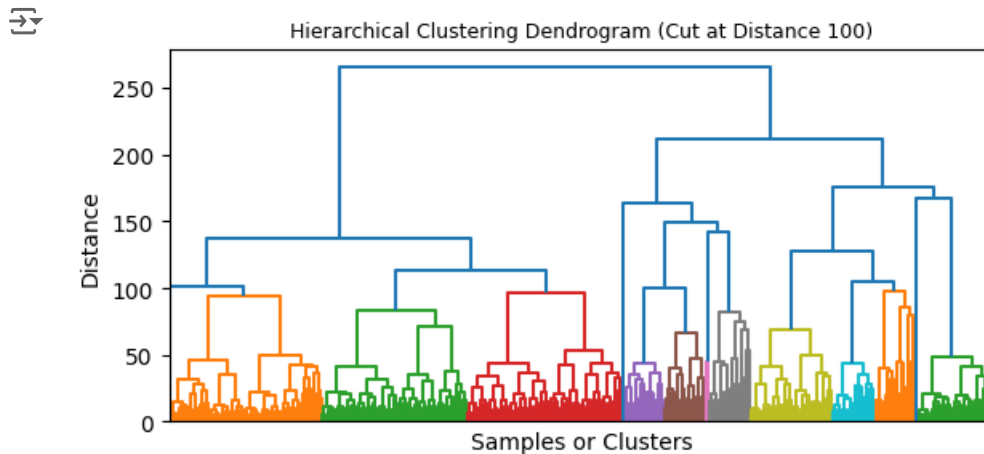
# Plot the dendrogram
plt.figure(figsize=(6, 3)) # Adjust the figure size
shc.dendrogram(
    linkage_matrix,
    color_threshold=100, # Cut-off threshold at distance 100
    truncate_mode='level', # Show only the top 5 levels of the hierarchy
    p=10, # Show only the last 10 merged clusters (adjust if needed)
    leaf_rotation=45, # Rotate labels to make them more readable
    leaf_font_size=10, # Font size for the leaf labels
)

```

```

    no_labels=True
)
plt.title('Hierarchical Clustering Dendrogram (Cut at Distance 100)', fontsize=9)
plt.xlabel('Samples or Clusters')
plt.ylabel('Distance')
plt.tight_layout()
plt.show()

```



DBSCAN (Density-Based Spatial Clustering of Applications with Noise)

- Clusters data based on the density of data points.
- Identifies core samples in high-density regions and expands clusters from them.

```

from sklearn.cluster import DBSCAN

# Set parameters
eps_value = 0.5 # You may need to adjust this
min_samples_value = 5 # You may need to adjust this

# Fit the model
dbscan = DBSCAN(eps=eps_value, min_samples=min_samples_value)
dbscan_labels = dbscan.fit_predict(scaled_features)

# Calculate Silhouette Score (excluding noise points labeled as -1)
from sklearn.metrics import silhouette_score
labels_without_noise = dbscan_labels[dbscan_labels != -1]
features_without_noise = scaled_features[dbscan_labels != -1]
if len(set(labels_without_noise)) > 1:
    silhouette_dbscan = silhouette_score(features_without_noise, labels_without_noise)
    print(f'Silhouette Score for DBSCAN: {silhouette_dbscan:.4f}')
else:
    print('DBSCAN did not find more than one cluster.')

# Visualization
tsne_df['DBSCAN_Cluster'] = dbscan_labels

```

⇒ Silhouette Score for DBSCAN: 0.5693

```

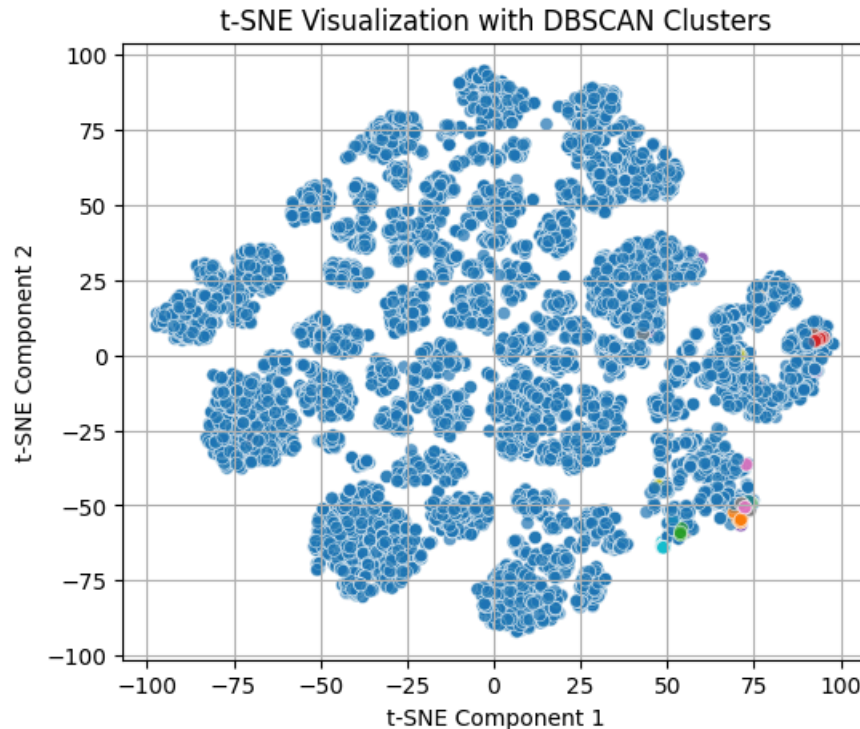
# Plot the t-SNE results with DBSCAN labels
plt.figure(figsize=(6, 5))
sns.scatterplot(
    data=tsne_df,
    x='TSNE1',
    y='TSNE2',
    hue='DBSCAN_Cluster',

```

```

palette='tab10',
alpha=0.7,
legend=False
)
plt.title('t-SNE Visualization with DBSCAN Clusters')
plt.xlabel('t-SNE Component 1')
plt.ylabel('t-SNE Component 2')
plt.grid(True)
plt.show()

```



Insights:

The Silhouette Score for DBSCAN is 0.5693, indicating relatively well-defined clusters. However, the t-SNE plot reveals a significant issue: almost all the data points are grouped into a single dominant cluster (Cluster 0), with only a small number of points forming the remaining clusters. This suggests that while DBSCAN has identified clear groupings in terms of density, it may not be effectively capturing the underlying structure or diversity in the data. The predominance of one cluster could indicate that the DBSCAN hyperparameters (such as `eps` and `min_samples`) may need adjustment to better distinguish more granular clusters. Alternatively, it might suggest that the dataset lacks natural separation between distinct groups.

Part V. Evaluation

- Compare all the models implemented with Silhouette Score

Gener a Dictionary with Model Names and Scores

Each key is a column name, and the values are lists of model names and their corresponding Silhouette Scores.

```

# Create a dictionary with model names and their Silhouette Scores
silhouette_scores = {
    'Model': [
        'K-Means (Original Data)',
        'K-Means (PCA-Reduced Data)',
        'Gaussian Mixture Model',

```





```

        'Agglomerative Clustering',
        'DBSCAN'
    ],
    'Silhouette Score': [
        silhouette_original,
        silhouette_pca_5,
        silhouette_gmm,
        silhouette_agglo,
        silhouette_dbscan
    ]
}

# Create a DataFrame
silhouette_df = pd.DataFrame(silhouette_scores)

silhouette_df

```

	Model	Silhouette Score	
0	K-Means (Original Data)	0.154373	
1	K-Means (PCA-Reduced Data)	0.319733	
2	Gaussian Mixture Model	0.137099	
3	Agglomerative Clustering	0.120076	
4	DBSCAN	0.569342	

Next steps:

 [View recommended plots](#)

[New interactive sheet](#)

Insights:

After generating the comparison table of Silhouette Scores for the different clustering algorithms, it is clear that the clustering results are not particularly strong across the methods tested. **K-Means on PCA-reduced data** showed a slight improvement over the original data (Silhouette Score of **0.3197** vs. **0.1544**), but the scores are still low, indicating poor cluster separation and cohesion. Both **Gaussian Mixture Model (GMM)** and **Agglomerative Clustering** resulted in even lower Silhouette Scores (**0.1371** and **0.1201**, respectively), with t-SNE visualizations showing significant overlap between clusters, suggesting these algorithms struggled to find distinct groupings in the data.

Notably, **DBSCAN** achieved a higher Silhouette Score of **0.5693**; however, this is somewhat misleading as the majority of data points were assigned to a single dominant cluster, with only a few points forming smaller clusters. This indicates that while DBSCAN identified dense regions, it did not capture meaningful subdivisions within the data.

In conclusion, the consistently low Silhouette Scores and overlapping clusters suggest that the dataset may not have well-defined clusters detectable by these algorithms with the current parameters. Further exploration, such as tuning hyperparameters, trying different clustering techniques, or even performing additional feature engineering, may be necessary to uncover any underlying patterns or improve clustering performance.

✓ Part VI. Cluster Analysis

- Analyzing clusters based on feature means

We will focus our cluster analysis on the K-Means clustering with PCA-reduced data (5 components) in this situation. Here is why:

- DBSCAN, while having a higher Silhouette Score, resulted in most data points being assigned to a single cluster, which doesn't provide meaningful segmentation for analysis.
- K-Means with PCA (5 components) has the highest Silhouette Score among the clustering algorithms you've implemented, excluding DBSCAN. Although not very high, indicates better cluster separation compared to other algorithms within our implemented methods.

```
# Add cluster labels to the original DataFrame
df['Cluster'] = kmeans_pca_labels_5
```

```
# Analyze clusters by looking at the mean values of features in each cluster
cluster_summary = df.groupby('Cluster').mean()
```

```
# Create a DataFrame
cluster_summary_df = pd.DataFrame(cluster_summary)
```

```
print("K-Means Cluster with PCA5 Analysis:")
cluster_summary_df
```

➡ K-Means Cluster with PCA5 Analysis:

	Intuition_Encoded	Age	Income	Employment_Status	High_Expectation	Industry_Experience
Cluster						
0	2.774564	29.391644	1.292011	0.885216	0.707530	0.652893
1	2.540584	22.041390	0.494356	0.038703	0.539509	0.009317
2	2.687897	31.286366	1.496009	0.898843	0.737579	0.014823

3 rows × 7 columns

```
import math
```

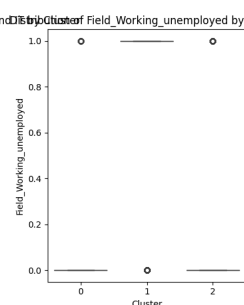
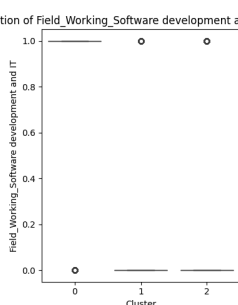
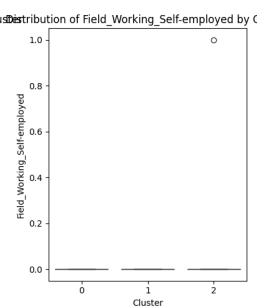
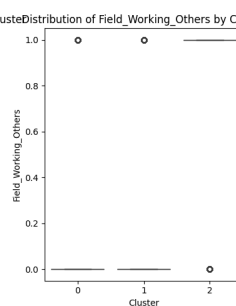
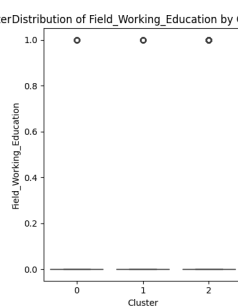
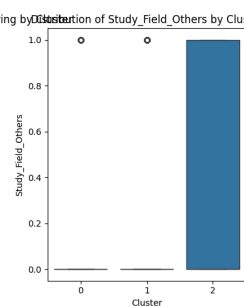
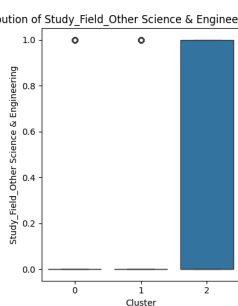
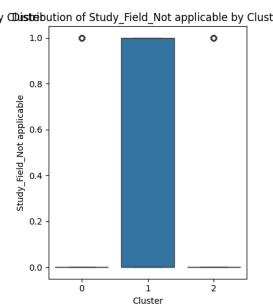
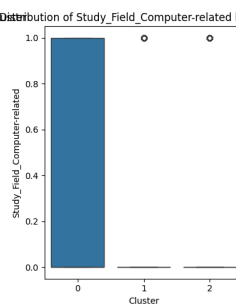
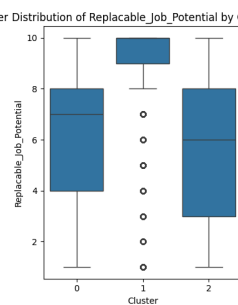
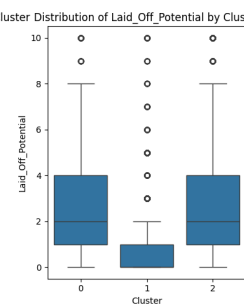
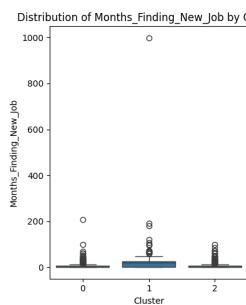
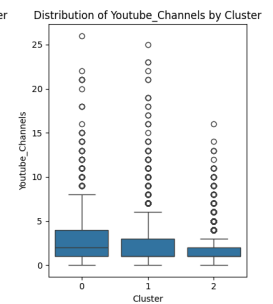
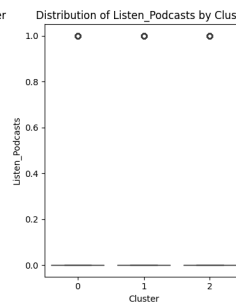
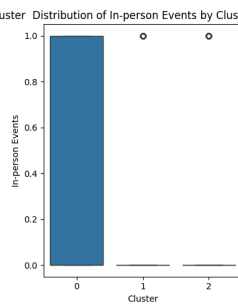
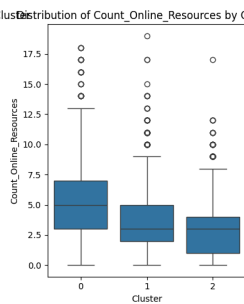
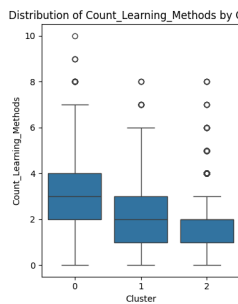
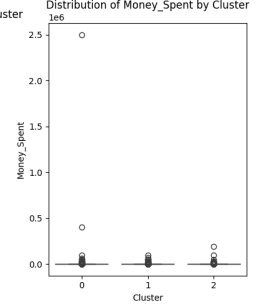
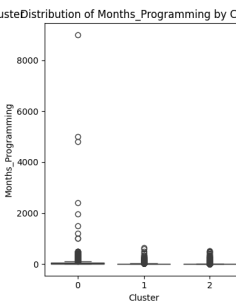
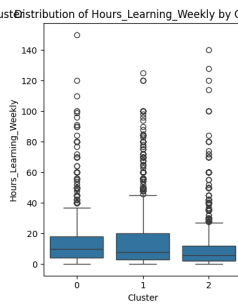
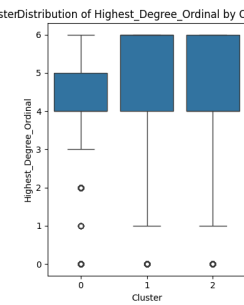
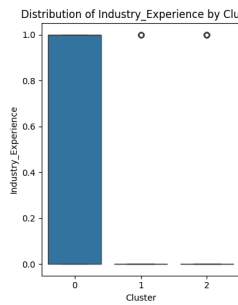
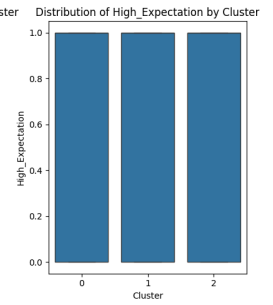
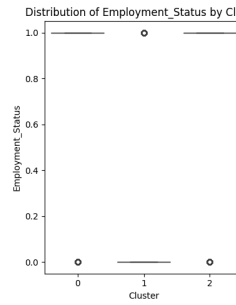
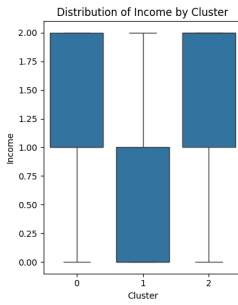
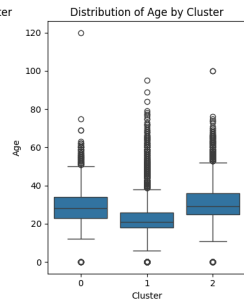
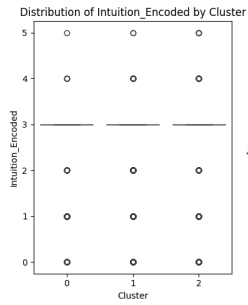
```
features = df.columns.drop('Cluster') # Exclude the 'Cluster' column
num_features = len(features)
plots_per_row = 5 # Number of subplots per row
num_rows = math.ceil(num_features / plots_per_row)
```

```
# Create subplots grid
fig, axes = plt.subplots(num_rows, plots_per_row, figsize=(20, num_rows * 5))
axes = axes.flatten()
```

```
# Plot each feature
for i, feature in enumerate(features):
    sns.boxplot(x='Cluster', y=feature, data=df, ax=axes[i])
    axes[i].set_title(f'Distribution of {feature} by Cluster')
    axes[i].set_xlabel('Cluster')
    axes[i].set_ylabel(feature)
```

```
# Remove any empty subplots
if len(axes) > len(features):
    for j in range(len(features), len(axes)):
        fig.delaxes(axes[j])
```

```
plt.tight_layout()
plt.show()
```



Insights:

This analysis uses K-Means clustering with 5 PCA components and displays the distribution of several features across the three identified clusters. Here's a breakdown of insights based on the provided box plots and statistics:

Cluster 0:

Fact:

- Age: Average age is 29.39, indicating mid-career individuals.
- Income: This group has a moderate income level (1.29), and most members are employed (Employment_Status = 0.89).
- Highest Degree: The average degree ordinal is 4.03, which corresponds closely to a Bachelor's degree based on the degree mapping. This suggests that most individuals in this cluster hold a Bachelor's degree, with some potentially having Associate's degrees or even Master's degrees.
- Tech Engagement: With 43.66% having a computer-related study field, and 90% working in software development or IT, this cluster represents tech professionals.
- Months Programming: Individuals in this cluster have extensive programming experience, with an average of 53 months (4+ years).

Overall: Cluster 0 is a group of mid-career, professionals who are actively engaged in learning and working in high-paying software development/IT roles.

Learning Habits:

- Learning Engagement: On average, individuals in this cluster spend 13.0 hours per week on learning and have relatively high expenditures on learning resources (average of 2433.1), suggesting a focus on continuous development.
- Count_Learning_Methods: The median count of learning methods is around 2-4 methods, indicating a balanced and diverse approach to learning.
- Count_Online_Resources: The median number of online resources used is about 5, indicating moderate engagement with digital learning platforms.
- In-person Events: This cluster has overwhelming participation in in-person events, suggesting that these individuals highly value face-to-face learning or networking opportunities.
- Youtube_Channels: Individuals in this cluster engage with 2-4 YouTube channels on average, with some using up to 25 channels, indicating heavy reliance on YouTube for educational content.
- Listen_Podcasts: Across all clusters, there seems to be very little engagement with podcasts.

Cluster 1:

Fact:

- Age: This is the youngest group, with an average age of 22.04, likely early-career individuals or students.
- Income and Employment: With a low average income (0.49) and very low employment status (Employment_Status = 0.038), this group struggles with employment or is still in school.
- Study Field: Around 20.30% of this group has a computer-related study field, while 21.04% come from other non-related fields.
- Tech Employment: Very few members of this cluster work in software development/IT (1.86%), but most are unemployed (Employment_Status = 0.95).

Overall: Cluster 1 represents younger, early-career individuals (or students) who are heavily focused on learning but face high unemployment rates and low income.

Learning Habits:

- Learning Engagement: Although they spend 13.41 hours per week on learning (higher than Cluster 0), they spend the least amount of money on learning resources (\$281.34), indicating they may rely on free or low-cost educational tools.

- **Count_Learning_Methods:** This cluster shows a similar pattern to Cluster 0, with individuals using multiple learning methods. However, there are more outliers in this cluster using up to 6 or 7 methods, indicating a more varied approach to education.
- **Count_Online_Resources:** Similar to Cluster 0, but with slightly fewer online resources (median around 3), suggesting a slightly lower usage of online resources.
- **In-person Events:** This cluster shows almost no participation in in-person events, indicating that they may prefer online learning or have fewer opportunities to attend such events.
- **Youtube_Channels:** A similar trend, with a slightly lower median engagement, but also with many outliers who use numerous YouTube channels for learning.
- **Listen_Podcasts:** Across all clusters, there seems to be very little engagement with podcasts.

Cluster 2:

Fact:

- **Age:** This is the oldest group, with an average age of 31.29.
- **Income:** Members of this cluster have the highest income (1.50), and the majority are employed (Employment_Status = 0.90).
- **Study Fields:** Only 7.75% have a computer-related study field, and most come from other fields such as science & engineering (34.52%).
- **Field of Work:** Interestingly, a significant portion of this group (81.0%) works in fields other than IT or education, indicating this group may represent mid-career professionals in non-tech industries.

Overall: Cluster 2 consists of experienced, high-income individuals working in a wide range of non-tech fields. They focus less on continuous learning compared to the tech-heavy Cluster 0.

Learning Habits:

- **Learning Engagement:** Although this cluster spends less time on learning weekly (9.29 hours) compared to Clusters 0 and 1, they still allocate a moderate amount on resources (\$316.51).
- **Count_Learning_Methods:** The fewest number of learning methods are used here, with the median being around 2 methods, suggesting that individuals in this cluster rely on a limited number of resources for learning.
- **Count_Online_Resources:** The smallest number of online resources used (median around 3), indicating that this group has limited interaction with online learning platforms compared to the other clusters.
- **In-person Events:** This cluster shows almost no participation in in-person events, indicating that they may prefer online learning or have fewer opportunities to attend such events.
- **Youtube_Channels:** The lowest engagement, with a median of around 3 channels, indicating that this cluster is less reliant on YouTube for learning.
- **Listen_Podcasts:** Across all clusters, there seems to be very little engagement with podcasts.

Thoughts:

The analysis of these three clusters derived from K-Means clustering using 5 PCA components aligns well with real-world trends and provides actionable insights. Cluster 0 represents mid-career tech professionals who are highly engaged in both their work and continuous learning, reflecting their desire to stay competitive in the rapidly evolving tech industry. Cluster 1, on the other hand, consists of younger individuals or students who are heavily focused on learning but are yet to establish