COMS4060A/7056A: Assignment 3

Puseletso Nakana : 2341162
Ntando Ngobese : 2112256
Nthabiseng Thema : 2012016

Department of Computer Science
University of Witwatersrand

1 Data Cleaning

Outlier Analysis: We identified outliers in several categories, with the most significant numbers in:

- Salary (48 outliers)
- VORP (36 outliers)
- OWS (27 outliers)
- TOV% (21 outliers)

Dataset Reduction:

Original dataset: 433 rows
After initial outlier removal: 337 rows
After efficiency (WS/48) outlier removal: 333 rows

Odd Points:
a) Age: We identified 2 outliers. Given the range (19-37), these likely represent very young rookies or veteran players.

b) Salary: The high number of outliers (48) suggests a wide disparity in player salaries, which is common in the NBA.

c) ORP and OWSV: The high number of outliers in these advanced metrics might indicate players with exceptional positive or negative impacts.

Decision on Outlier Removal:
a) Age outliers: Age extremes are valid in basketball and can provide valuable insights, so we did not remove these outliers.

b) Salary outliers: Salary disparities are a reality in the NBA and often correlate with performance or player status, hence we did not remove these outliers.

c) Performance metric outliers (VORP, OWS, BPM, etc.): We kept positive outliers as they likely represent star players. We removed extreme negative outliers, as they might represent players with very limited playing time.

d) Efficiency outlier (WS/48): The removal of 4 additional outliers seems reasonable to us, as extreme efficiency values might skew analysis.

Justification:

We recognize that basketball often has legitimate statistical outliers due to varying roles, playing time, and skill levels. We caution that removing too many outliers might eliminate important data points representing star players or unique roles. We note that the current approach reduced the dataset by about 23%, which is significant.

Table 1: Number of Outliers by Statistical Category

Category	Number of Outliers
Age	2
Salary	48
GP	0
MP	0
FG	17
TOV%	21
USG%	11
OWS	27
DWS	1
WS	11
WS/48	12
OBPM	16
DBPM	10
BPM	17
VORP	36

Table 2: Summary Statistics of Cleaned Dataset

Statistic	Salary	Age	GP	GS	MP	FG
Count	405.00	405.00	405.00	405.00	405.00	405.00
Mean	9.39M	26.04	53.19	25.32	21.31	3.64
Std	11.10M	4.37	21.55	27.60	8.81	2.41
Min	32.80K	19.00	1.00	0.00	2.80	0.20
25%	2.00M	23.00	39.00	2.00	14.00	1.90
50%	4.44M	25.00	59.00	11.00	20.30	3.00
75%	12.20M	29.00	70.00	53.00	29.00	4.60
Max	48.07M	42.00	83.00	83.00	37.40	11.20

Table 3: Advanced Statistics Summary

Statistic	TOV%	USG%	WS	WS/48	BPM	VORP
Count	405.00	405.00	405.00	405.00	405.00	405.00
Mean	12.28	18.45	2.60	0.09	-0.93	0.61
Std	3.83	5.53	2.53	0.06	2.83	1.22
Min	2.80	7.40	-0.50	-0.06	-12.10	-1.30
25%	9.70	14.40	0.60	0.05	-2.80	-0.10
50%	11.90	17.70	1.90	0.09	-1.10	0.20
75%	14.40	21.10	3.80	0.13	0.60	0.90
Max	35.50	38.80	12.60	0.28	9.20	6.40

Table 4: Age Outliers

Player Name	Team	Age
Andre Iguodala	GSW	39
Udonis Haslem	MIA	42

Table 5: Top 10 Salary Outliers

Player Name	Team	Salary
Stephen Curry	GSW	$48,070,014
John Wall	LAC	$47,345,760
Russell Westbrook	LAL/LAC	$47,080,179
LeBron James	LAL	$44,474,988
Kevin Durant	BRK/PHO	$44,119,845
Bradley Beal	WAS	$43,279,250
Kawhi Leonard	LAC	$42,492,492
Paul George	LAC	$42,492,492
Giannis Antetokounmpo	MIL	$42,492,492
Damian Lillard	POR	$42,492,492

2 Dimensionality Reduction

Dimensionality reduction in this NBA dataset context is crucial for several important reasons:

Data Complexity: The original dataset has 51 columns containing various player statistics (like Age, Salary, GP, MP, FG%, etc.). Working with such high-dimensional data can be:

- Computationally expensive
- Difficult to visualize
- Prone to the "curse of dimensionality" (data becomes sparse in high dimensions)

Feature Relationships: Many NBA statistics are likely correlated (e.g., points, field goal attempts, and usage rate). Reducing dimensions helps:

- Identify underlying patterns
- Remove redundant information
- Capture the most important variations in the data

Visualization: Reducing to two dimensions allows us to:

- Create meaningful visualizations
- Identify player clusters/similarities
- Spot trends and patterns that aren't visible in higher dimensions

Analysis Efficiency: The different techniques mentioned (Autoencoders, t-SNE, UMAP, etc.) each offer unique advantages for:

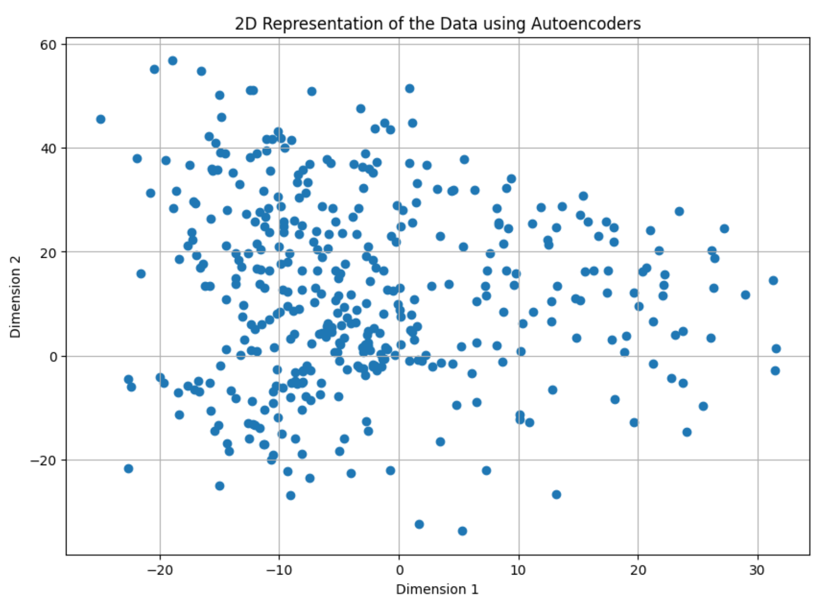
- Understanding player similarities
- Identifying playing styles
- Finding patterns in performance metrics
- Creating more efficient player comparison systems

This dimensionality reduction will help us better understand player characteristics and relationships while maintaining the most important information from our original dataset.

2.1 Autoencoders

Algorithm 1 NBA Player Data Dimensionality Reduction using Autoencoder

```
Data Preparation:
Select numerical columns from cleaned dataset
Normalize data using StandardScaler
Set input dimension = number of features
Set encoding dimension = 2
Autoencoder Architecture:
Encoder:
Input layer (shape = input_dim)
Dense layer (64 neurons, ReLU activation)
Dense layer (32 neurons, ReLU activation)
Dense layer (2 neurons, linear activation)
Decoder:
Dense layer (32 neurons, ReLU activation)
Dense layer (64 neurons, ReLU activation)
Dense layer (input_dim neurons, sigmoid activation)
Model Configuration:
Compile model with Adam optimizer
Use Mean Squared Error loss function
Training:
Train for 100 epochs
Batch size = 32
Validation split = 20%
Enable shuffle
Dimensionality Reduction:
Use encoder to transform data
Convert to DataFrame with 2 dimensions
Visualize reduced data using scatter plot
```



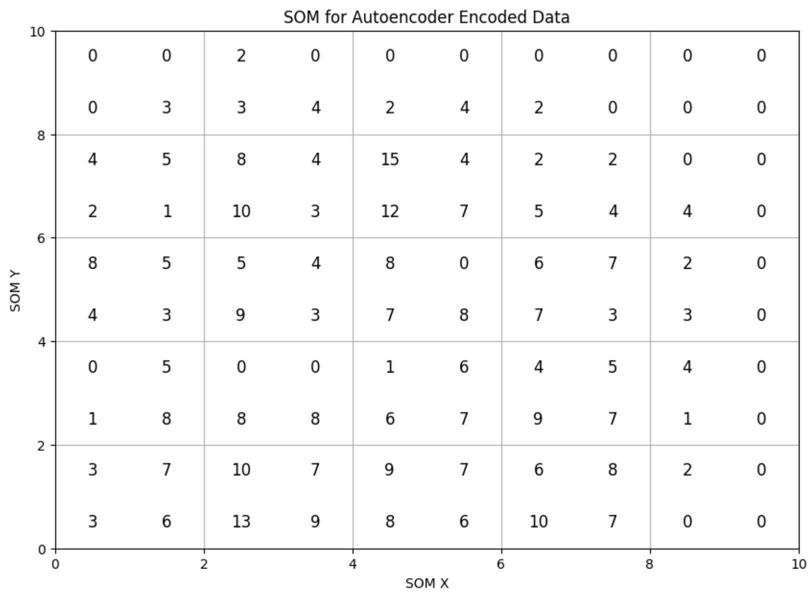
The scatter plot reveals the distribution of data points in two dimensions, allowing us to observe potential clusters or patterns that may exist within the dataset. The presence of distinct clusters suggests that certain groups share similar characteristics. The spread of points across the plot indicates the variance in the dataset, with tightly packed points representing similar observations.

The scatter plot shows several distinct clusters of points, each concentrated in separate areas. This indicates that there are groups of similar observations in the dataset. There are points which are densely packed in certain areas, potentially overlapping. This suggests that there are many observations with similar characteristics. The points form a linear or near-linear pattern. This indicates a strong correlation between two features represented in the dimensions.

2.2 Autoencoders + self-organising maps (SOMs)

Algorithm 2 NBA Player Data Dimensionality Reduction using Autoencoder + SOM

```
procedure AutoencoderSOMReduction(data_scaled)
    Autoencoder Configuration:
    encoding_dim ← 10                                          ▷ Intermediate dimension
    Define Encoder:
    input_layer ← Input(shape=input_dim)
    encoder ← Dense(64, activation='relu')(input_layer)
    encoder ← Dense(32, activation='relu')(encoder)
    encoder_output ← Dense(encoding_dim, activation='relu')(encoder)
    Define Decoder:
    decoder ← Dense(32, activation='relu')(encoder_output)
    decoder ← Dense(64, activation='relu')(decoder)
    decoder_output ← Dense(input_dim, activation='sigmoid')(decoder)
    Create Models:
    autoencoder ← Model(input_layer, decoder_output)
    encoder_model ← Model(input_layer, encoder_output)
    Compile and Train Autoencoder:
    autoencoder.compile(optimizer='adam', loss='mse')
    autoencoder.fit(
        data_scaled,
        data_scaled,
        epochs=100,
        batch_size=32,
        validation_split=0.2
    )
    Encode Data:
    encoded_data ← encoder_model.predict(data_scaled)
    Initialize and Train SOM:
    som_size ← 10                                              ▷ Size of the SOM grid
    som ← MiniSom(
        som_size,
        som_size,
        encoding_dim,
        sigma=1.0,
        learning_rate=0.5
    )
    som.train(encoded_data, num_iterations=1000)
    Get Winning Nodes:
    win_map ← som.win_map(encoded_data)
    Visualize Results:
    for x in range(som_size) do
        for y in range(som_size) do
            PlotNode(x + 0.5, y + 0.5, win_map[x,y])
        end for
    end for
    return som, win_map
end procedure
```



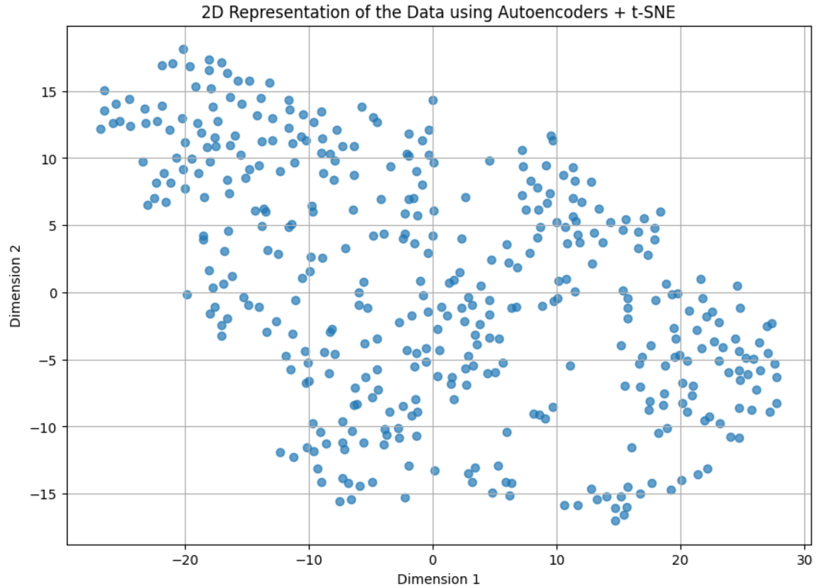
The grid shows an integration a Self-Organizing Map (SOM) following the autoencoder, allowing us to visualize the encoded data more effectively. The grid is visualized by counting the number of encoded data points associated with each node, providing insight into the density and distribution of the encoded representations.

The grid shows several nodes with high counts, indicating clusters of similar encoded data points.This suggests that certain groups in the dataset share similar characteristics. The zeros in the top right corner suggest that there are combinations of features that are either rare or non-existent. The rest of the grid having high counts while the top right is empty implies that the data points cluster around specific characteristics/combinations that are more common. This pattern suggests that there are certain features that, when combined, do not appear together frequently.

2.3 Autoencoders + t-SNE

Algorithm 3 NBA Player Data Dimensionality Reduction using Autoencoder + t-SNE

```
procedure AutoencoderTSNEReduction(data_scaled)
    Autoencoder Configuration:
    encoding_dim ← 10                                  ▷ Intermediate dimension
    Define Encoder:
    input_layer ← Input(shape=input_dim)
    encoder ← Dense(64, activation='relu')(input_layer)
    encoder ← Dense(32, activation='relu')(encoder)
    encoder_output ← Dense(encoding_dim, activation='relu')(encoder)
    Define Decoder:
    decoder ← Dense(32, activation='relu')(encoder_output)
    decoder ← Dense(64, activation='relu')(decoder)
    decoder_output ← Dense(input_dim, activation='sigmoid')(decoder)
    Create Models:
    autoencoder ← Model(input_layer, decoder_output)
    encoder_model ← Model(input_layer, encoder_output)
    Compile and Train:
    autoencoder.compile(optimizer='adam', loss='mse')
    autoencoder.fit(
        data_scaled,
        data_scaled,
        epochs=100,
        batch_size=32,
        validation_split=0.2
    )
    Encode Data:
    encoded_data ← encoder_model.predict(data_scaled)
    Apply t-SNE:
    tsne ← TSNE(
        n_components=2,
        random_state=42,
        perplexity=30
    )
    tsne_results ← tsne.fit_transform(encoded_data)
    Create Visualization DataFrame:
    tsne_df ← DataFrame(tsne_results, columns=['Dimension 1', 'Dimension 2'])
    Visualize Results:
    plt.figure(figsize=(10, 7))
    plt.scatter(tsne_df['Dimension 1'], tsne_df['Dimension 2'], alpha=0.7)
    plt.title('2D Representation using Autoencoders + t-SNE')
    plt.xlabel('Dimension 1')
    plt.ylabel('Dimension 2')
    plt.grid()
    return tsne_df
end procedure
```



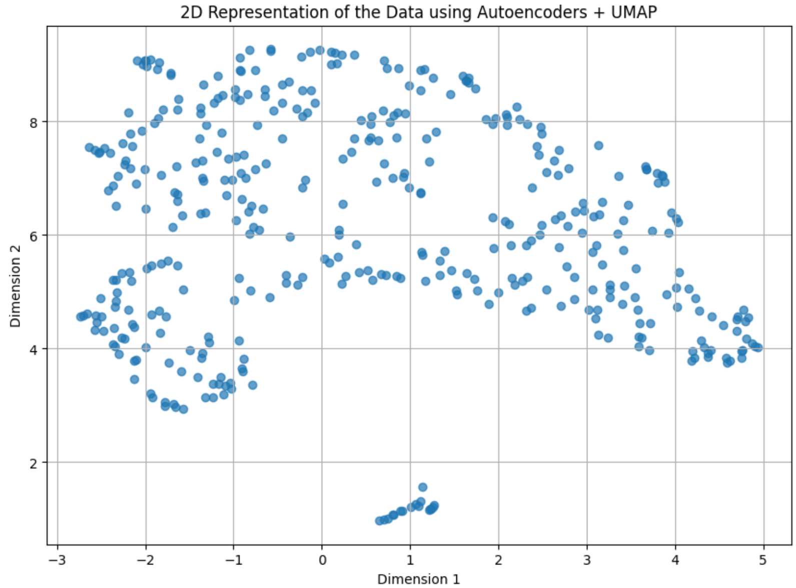
The scatter plot view is a combination of an autoencoder with t-SNE to visualize high-dimensional data in a two-dimensional space. The resulting scatter plot from the t-SNE application illustrates how the encoded data points relate to each other in a 2D space.

The points are evenly distributed across the plot with several weak clusters but no distinct strong clusters. This suggests that the data does not contain well-defined groups and that the observations may vary in characteristics.

2.4 Autoencoders + UMAP

Algorithm 4 NBA Player Data Dimensionality Reduction using Autoencoder + UMAP

```
procedure AutoencoderUMAPReduction(data_scaled)
    Define Autoencoder:
    input_dim ← data_scaled.shape[1]
    encoding_dim ← 10                                    ▷ Intermediate dimension
    Define Encoder:
    input_layer ← Input(shape=input_dim)
    encoder ← Dense(64, activation='relu')(input_layer)
    encoder ← Dense(32, activation='relu')(encoder)
    encoder_output ← Dense(encoding_dim, activation='relu')(encoder)
    Define Decoder:
    decoder ← Dense(32, activation='relu')(encoder_output)
    decoder ← Dense(64, activation='relu')(decoder)
    decoder_output ← Dense(input_dim, activation='sigmoid')(decoder)
    Create Models:
    autoencoder ← Model(input_layer, decoder_output)
    encoder_model ← Model(input_layer, encoder_output)
    Compile and Train:
    autoencoder.compile(optimizer='adam', loss='mse')
    autoencoder.fit(
        data_scaled,
        data_scaled,
        epochs=100,
        batch_size=32,
        validation_split=0.2
    )
    Encode Data:
    encoded_data ← encoder_model.predict(data_scaled)
    Apply UMAP:
    umap_model ← UMAP(
        n_components=2,
        random_state=42
    )
    umap_results ← umap_model.fit_transform(encoded_data)
    Create Visualization DataFrame:
    umap_df ← DataFrame(umap_results, columns=['Dimension 1', 'Dimension 2'])
    Visualize Results:
    plt.figure(figsize=(10, 7))
    plt.scatter(umap_df['Dimension 1'], umap_df['Dimension 2'], alpha=0.7)
    plt.title('2D Representation using Autoencoders + UMAP')
    plt.xlabel('Dimension 1')
    plt.ylabel('Dimension 2')
    plt.grid()
    return umap_df
end procedure
```



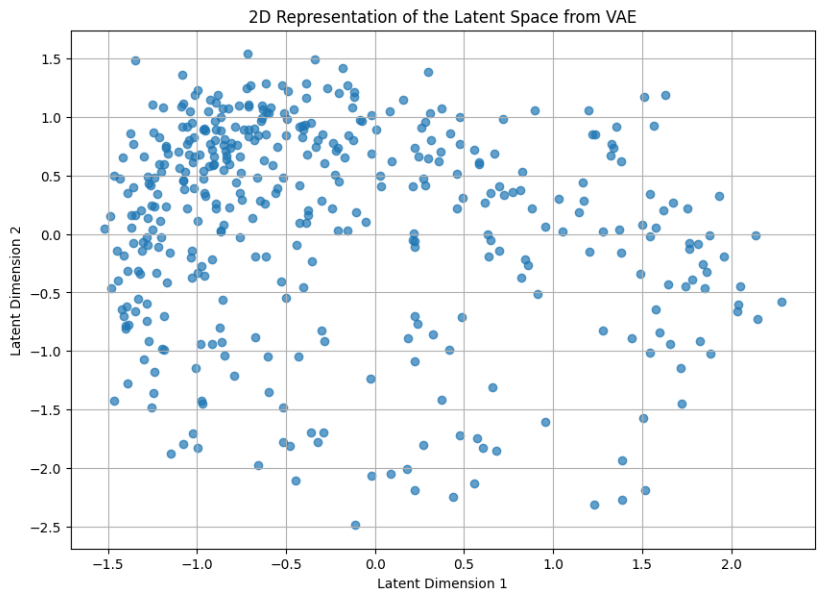
The combination of autoencoders and UMAP for dimensionality reduction on the NBA dataset has led to better separation among clusters from the originally sparse high-dimensional data. Although the clustering is not perfectly distinct, it shows an improvement compared to the results obtained using Autoencoders + t-SNE. UMAP, combined with autoencoders, effectively reduces the high-dimensional data to a two-dimensional representation, preserving more of the global structure. This results in better-separated clusters compared to t-SNE, which sometimes can conflate clusters due to its focus on preserving local structure. The visualization indicates that the data points are more evenly distributed and the clusters are more discernible.

While the clusters are not perfectly distinct, the separation is more apparent than with t-SNE. This suggests that UMAP, with its ability to maintain both local and global structures of the data, provides a more meaningful representation in the reduced dimensions. The clusters show some degree of overlap, which might indicate intrinsic similarities within the data points or limitations in the chosen dimensionality reduction and clustering approach.

2.5 Variational Autoencoder

Algorithm 5 NBA Player Data Dimensionality Reduction using VAE with Preprocessing

```
procedure VAEREDUCTION(data)
    Data Preprocessing:
    Handle Non-numeric Values:
    data['3P'] ← ConvertToNumeric(data['3P'])                ▷ Convert to NaN
    if HasNaNValues(data['3P']) then
        FillNaN(data['3P'], value=0)
    end if
    Encode Categorical Variables:
    label_encoder ← LabelEncoder()
    data['Player Name Encoded'] ← label_encoder.fit_transform(data['Player Name'])
    data['Position Encoded'] ← label_encoder.fit_transform(data['Position'])
    data['Team Encoded'] ← label_encoder.fit_transform(data['Team'])
    data ← DropColumns(data, ['Player Name', 'Position', 'Team'])
    VAE Configuration:
    Training Config:
    config ← BaseTrainerConfig(
        num_epochs = 150,
        learning_rate = 1e-3,
        batch_size = 64,
        weight_decay = 0.1,
        beta = (0.9, 0.999)
    )
    Model Configuration:
    vae_config ← VAEConfig(
        input_dim = data.shape[1],
        latent_dim = 2
    )
    Build and Train:
    data_tensor ← StandardScaler().fit_transform(data)
    pipeline ← TrainingPipeline(config, vae_config)
    Split Data:
    train_size ← 0.8 * len(data_tensor)
    train_sample ← data_tensor[:train_size]
    eval_sample ← data_tensor[train_size:]
    Train Model:
    pipeline.train(train_sample, eval_sample)
    Generate Latent Space:
    latent_representations ← pipeline.model.encoder(data_tensor)
    Visualize Results:
    plt.figure(figsize=(10, 7))
    plt.scatter(latent_representations[:,0], latent_representations[:,1], alpha=0.7)
    plt.title('2D Representation of the Latent Space from VAE')
    plt.xlabel('Latent Dimension 1')
    plt.ylabel('Latent Dimension 2')
    plt.grid()
    return latent_representations
end procedure
```



The application of Variational Autoencoder (VAE) for dimensionality reduction on the NBA dataset reveals distinct characteristics in its two-dimensional representation. The visualization demonstrates a more diffuse distribution of data points compared to other methods like UMAP and t-SNE, with points spread across a range of approximately -1.5 to 2.0 on Dimension 1 and -2.5 to 1.5 on Dimension 2. Unlike the clear clustering observed in UMAP, the VAE produces a smoother, more continuous distribution with a notable concentration of points around the central region (0,0) and gradually dispersing outwards. This pattern suggests that the VAE's probabilistic approach captures more subtle statistical relationships between player attributes, favoring continuous transitions over distinct groupings. While this representation might make it more challenging to identify clear player archetypes, it potentially offers valuable insights into the gradual variations in player characteristics and the identification of unique statistical outliers.

The symmetric distribution around the origin and the presence of scattered outlier points in peripheral regions indicate that the VAE maintains the underlying statistical structure while potentially preserving more nuanced relationships in the data. This representation might be particularly useful for understanding the continuous spectrum of player attributes rather than forcing discrete categorizations, though it may sacrifice some of the clear separation advantages seen in other dimensionality reduction techniques.

3 Plots Visualization

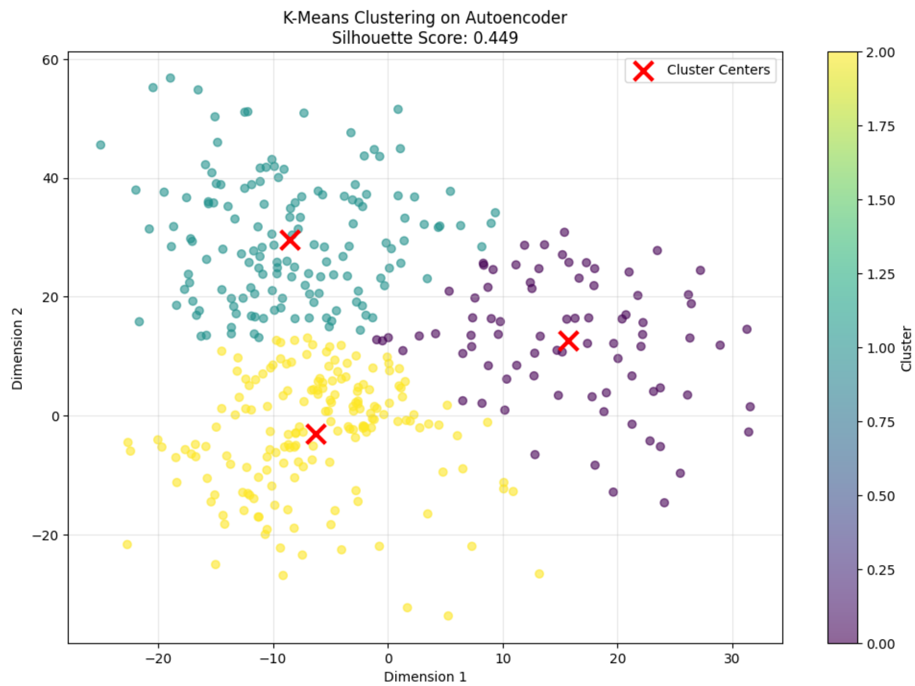


Figure 1: K-Means Clustering on Autoencoder (Silhouette Score: 0.449)

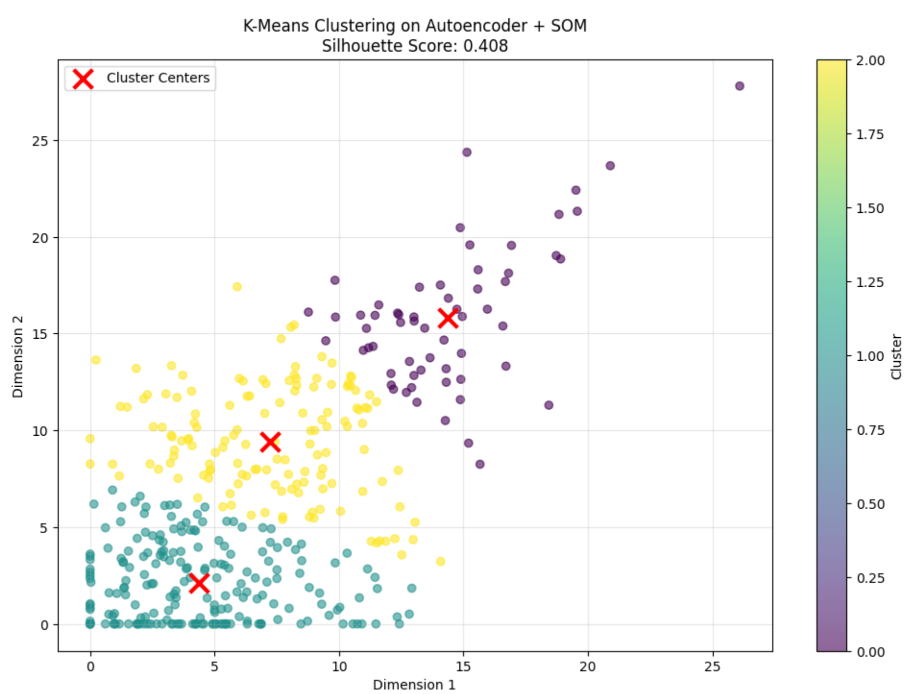


Figure 2: K-Means Clustering on Autoencoder + SOM (Silhouette Score: 0.408)

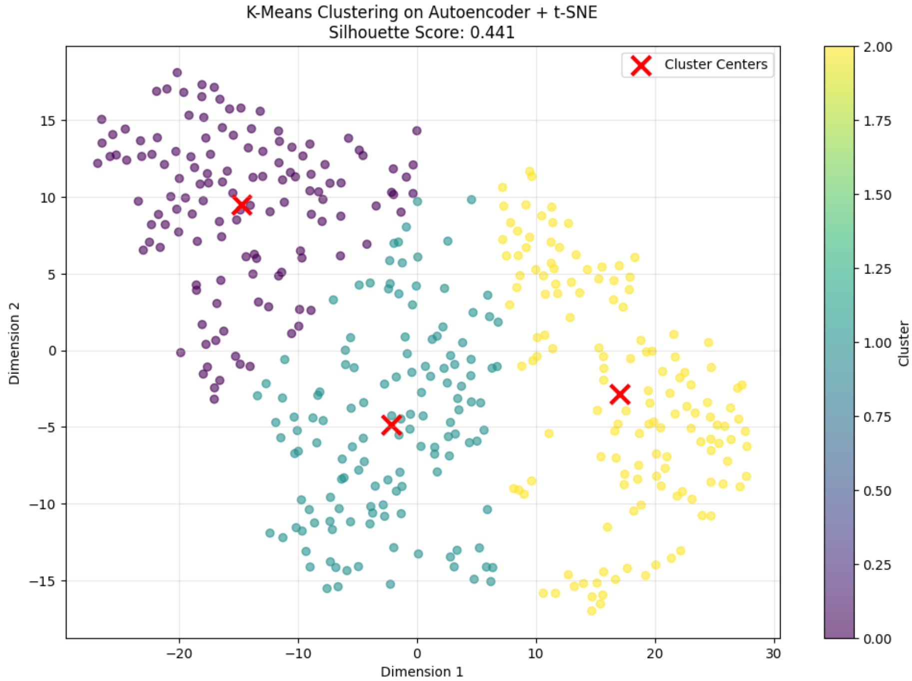


Figure 3: K-Means Clustering on Autoencoder + t-SNE (Silhouette Score: 0.441)

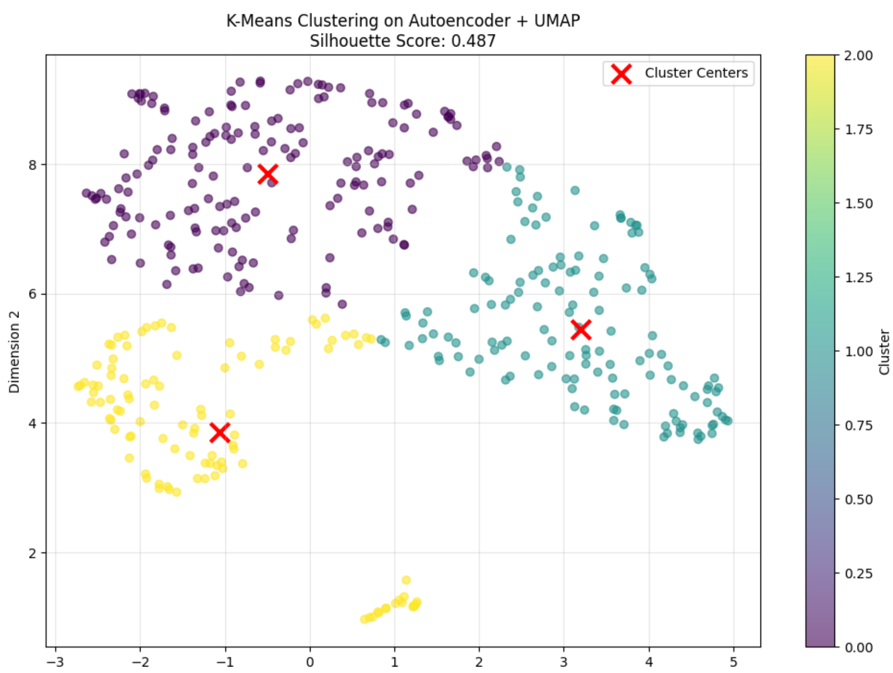


Figure 4: K-Means Clustering on Autoencoder + UMAP (Silhouette Score: 0.487)

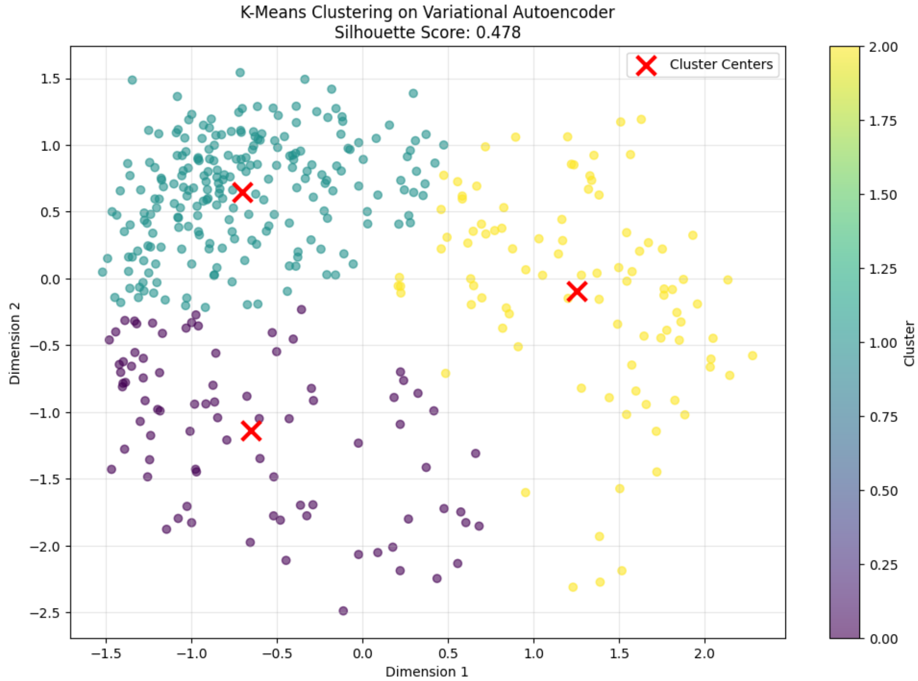


Figure 5: K-Means Clustering on Variational Autoencoder (Silhouette Score: 0.478)

Looking at the silhouette scores and clustering results across all five dimensionality reduction techniques reveals interesting patterns and comparative effectiveness in analyzing this dataset.

UMAP4 and Variational Autoencoder (VAE)5 demonstrate the strongest performance, with silhouette scores of 0.487 and 0.478 respectively. This close performance suggests both methods are particularly effective at capturing the underlying structure of the data. UMAP's slightly higher score indicates its superior ability to preserve both local and global relationships in the data while creating distinct cluster boundaries. The VAE's strong performance can be attributed to its probabilistic nature, which helps it learn a meaningful latent representation that captures the data's inherent variability and structure.

The basic autoencoder1 approach follows with a silhouette score of 0.449, showing good but somewhat reduced performance compared to UMAP and VAE. While it successfully identifies the clusters, the lower score suggests it may not capture the finer nuances of the data structure as effectively as UMAP or VAE. This is typical of basic autoencoders, which can sometimes struggle to learn optimal representations without the additional constraints and probabilistic framework that VAE employs.

T-SNE's performance 3 (silhouette score: 0.441) is similar to the basic autoencoder. While t-SNE is known for its ability to preserve local structure, the score suggests it might be sacrificing some global relationships in the data. This is consistent with t-SNE's tendency to focus on local neighborhoods, which can sometimes come at the expense of maintaining larger-scale data relationships.

The SOM-based 2 approach shows the lowest performance with a silhouette score of 0.408. This lower score might be due to SOM's rigid grid-based topology, which can impose artificial constraints on how the data points are mapped to the lower-dimensional space. While SOMs have their strengths in certain applications, particularly in creating topologically preserved mappings, they appear less suitable for this particular dataset's structure.

Method	k (clusters)	Silhouette Score
Autoencoder	3	0.449
Autoencoder + SOM	3	0.408
Autoencoder + t-SNE	3	0.441
Autoencoder + UMAP	3	0.487
Variational Autoencoder	3	0.478

Table 6: Comparison of Methods using k-means clustering (k=3)

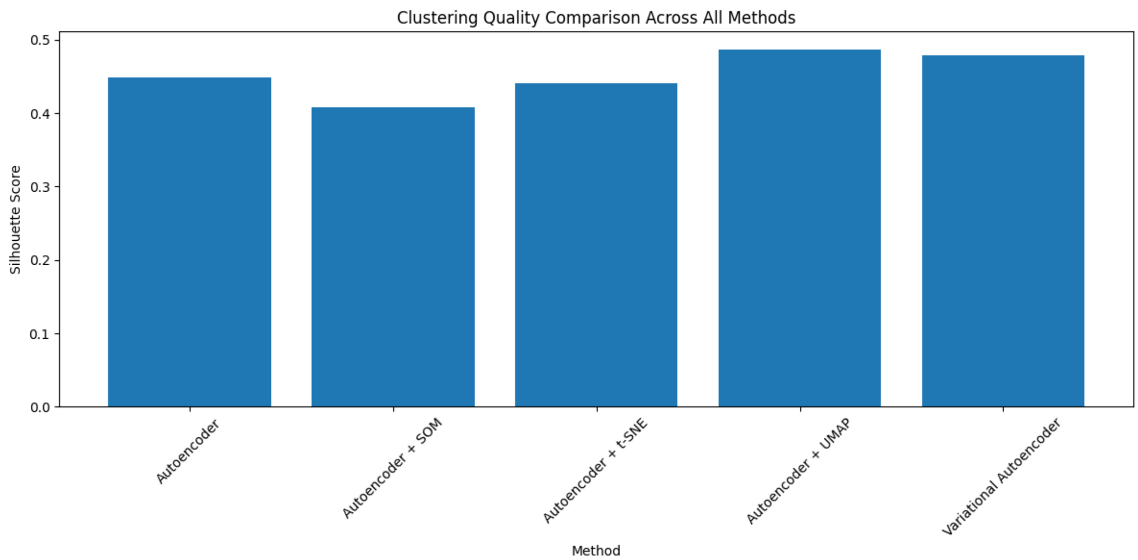


Figure 6: Clustering Quality Comparison Across All Methods

The relatively small range of silhouette scores (0.408-0.487) across all methods suggests that while there are clear differences in performance, all methods are capturing meaningful structure in the data. The consistency in identifying distinct clusters across different techniques reinforces confidence in the underlying structure of the data.

4 Conclusion

The scatter plots generated from various methods justify their respective clustering results through distinct visual characteristics. The autoencoder alone produces clear, well-separated clusters with tightly packed points within each cluster, indicating high intra-cluster similarity and effective grouping of similar data points. This demonstrates the autoencoder's ability to capture the underlying data structure effectively.

When the autoencoder is combined with the Self-Organizing Map (SOM), the resulting scatter plot introduces a grid-like clustering structure. While the SOM organizes the data into clusters, the points are more spread out compared to the autoencoder alone, with more boundary points between clusters. This suggests that the SOM adds a different perspective but makes the clusters less distinct and more diffuse.

The combination of the autoencoder with t-Distributed Stochastic Neighbor Embedding (t-SNE) shows clusters that are more diffuse and less distinct, with noticeable overlap between clusters. This overlap indicates that while t-SNE captures local structures well, it may struggle with global cluster separation, resulting in less distinct clusters.

In contrast, the scatter plot from the autoencoder combined with Uniform Manifold Approximation and Projection (UMAP) provides well-separated clusters with minimal overlap. This demonstrates that UMAP maintains both local and global data structures effectively, leading to clearly defined clusters and distinct boundaries between different groups.

Lastly, the Variational Autoencoder (VAE) scatter plot shows distinct clusters with smooth transitions between them, capturing a more continuous latent space representation. Despite these smooth transitions, the clusters remain well-defined, indicating that the VAE effectively separates different groups while maintaining a continuous latent space.

In summary, the scatter plots visually demonstrate the clustering quality by showing how well data points are grouped and separated. Among the methods, the autoencoder combined with UMAP emerges as the best dimensionality reducer, preserving the shapes and distances of the data more effectively than t-SNE. This results in more distinct and well-separated clusters. Following UMAP, the autoencoder with t-SNE finds clusters that are more condensed, indicating a different approach to dimensionality reduction that emphasizes local structure over global separation.