Case Study_Human Resourse Dataset

- **Human_Resources.csv Analysis**
- Apply K mean Clustering
- Apply PCA
- Apply Autoencoder

Task 1:Import your libraries

In [ ]:
```
#Import the libraries here
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import warnings
from sklearn.preprocessing import OneHotEncoder
```

In [17]:
```
df = pd.read_csv('Human_Resources.csv')
df.head()
```

Out[17]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educat
0	41	Yes	Travel_Rarely	1102	Sales	1	
1	49	No	Travel_Frequently	279	Research & Development	8	
2	37	Yes	Travel_Rarely	1373	Research & Development	2	
3	33	No	Travel_Frequently	1392	Research & Development	3	
4	27	No	Travel_Rarely	591	Research & Development	2	

5 rows × 35 columns

In [18]:
```
# show all the file data types
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1470 entries, 0 to 1469
Data columns (total 35 columns):
 #   Column                    Non-Null Count  Dtype 
---  ------                    --------------  ----- 
 0   Age                       1470 non-null   int64 
 1   Attrition                 1470 non-null   object
 2   BusinessTravel            1470 non-null   object
 3   DailyRate                 1470 non-null   int64 
 4   Department                1470 non-null   object
 5   DistanceFromHome          1470 non-null   int64 
 6   Education                 1470 non-null   int64 
 7   EducationField            1470 non-null   object
 8   EmployeeCount             1470 non-null   int64 
 9   EmployeeNumber            1470 non-null   int64 
 10  EnvironmentSatisfaction   1470 non-null   int64 
 11  Gender                    1470 non-null   object
 12  HourlyRate                1470 non-null   int64 
 13  JobInvolvement            1470 non-null   int64 
 14  JobLevel                  1470 non-null   int64 
 15  JobRole                   1470 non-null   object
 16  JobSatisfaction           1470 non-null   int64 
 17  MaritalStatus             1470 non-null   object
 18  MonthlyIncome             1470 non-null   int64 
 19  MonthlyRate               1470 non-null   int64 
 20  NumCompaniesWorked        1470 non-null   int64 
 21  Over18                    1470 non-null   object
 22  OverTime                  1470 non-null   object
 23  PercentSalaryHike         1470 non-null   int64 
 24  PerformanceRating         1470 non-null   int64 
 25  RelationshipSatisfaction  1470 non-null   int64 
 26  StandardHours             1470 non-null   int64 
 27  StockOptionLevel          1470 non-null   int64 
 28  TotalWorkingYears         1470 non-null   int64 
 29  TrainingTimesLastYear     1470 non-null   int64 
 30  WorkLifeBalance           1470 non-null   int64 
 31  YearsAtCompany            1470 non-null   int64 
 32  YearsInCurrentRole        1470 non-null   int64 
 33  YearsSinceLastPromotion   1470 non-null   int64 
 34  YearsWithCurrManager      1470 non-null   int64 
dtypes: int64(26), object(9)
memory usage: 402.1+ KB
```

In [19]:
```
# Show the following basic statistics
df.describe()
```

Out[19]:

	Age	DailyRate	DistanceFromHome	Education	EmployeeCount	Em
count	1470.000000	1470.000000	1470.000000	1470.000000	1470.0	
mean	36.923810	802.485714	9.192517	2.912925	1.0	
std	9.135373	403.509100	8.106864	1.024165	0.0	
min	18.000000	102.000000	1.000000	1.000000	1.0	
25%	30.000000	465.000000	2.000000	2.000000	1.0	
50%	36.000000	802.000000	7.000000	3.000000	1.0	
75%	43.000000	1157.000000	14.000000	4.000000	1.0	
max	60.000000	1499.000000	29.000000	5.000000	1.0	

8 rows × 26 columns

Task 2:VISUALIZE DATASET

In [20]:
```
# Replace 'Attritition','Overtime' and 'Over18' columns with integers before per
# Replace 'Attrition' column with integers
df['Attrition'] = df['Attrition'].apply(lambda x: 1 if x == 'Yes' else 0)

# Replace 'OverTime' column with integers
df['OverTime'] = df['OverTime'].apply(lambda x: 1 if x == 'Yes' else 0)

# Replace 'Over18' column with integers
df['Over18'] = df['Over18'].apply(lambda x: 1 if x == 'Y' else 0)
```

In [21]:
```
# display the current first four records
df.head(4)
```

Out[21]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educat
0	41	1	Travel_Rarely	1102	Sales	1	
1	49	0	Travel_Frequently	279	Research & Development	8	
2	37	1	Travel_Rarely	1373	Research & Development	2	
3	33	0	Travel_Frequently	1392	Research & Development	3	

4 rows × 35 columns

In [22]:
```
# Drop EmployeeNumber',EmployeeCount' ,'Standardhours' and 'Over18' since they d
df.drop(['EmployeeNumber', 'EmployeeCount', 'StandardHours', 'Over18'], axis=1,
df.head()
```

Out[22]:

	Age	Attrition	BusinessTravel	DailyRate	Department	DistanceFromHome	Educat
0	41	1	Travel_Rarely	1102	Sales	1	
1	49	0	Travel_Frequently	279	Research & Development	8	
2	37	1	Travel_Rarely	1373	Research & Development	2	
3	33	0	Travel_Frequently	1392	Research & Development	3	
4	27	0	Travel_Rarely	591	Research & Development	2	

5 rows × 31 columns

In [23]:

```
employee_df = df.copy()
# Let's see how many employees left the company!
left_df        = employee_df[employee_df['Attrition'] == 1]
stayed_df      = employee_df[employee_df['Attrition'] == 0]
```

In [24]:

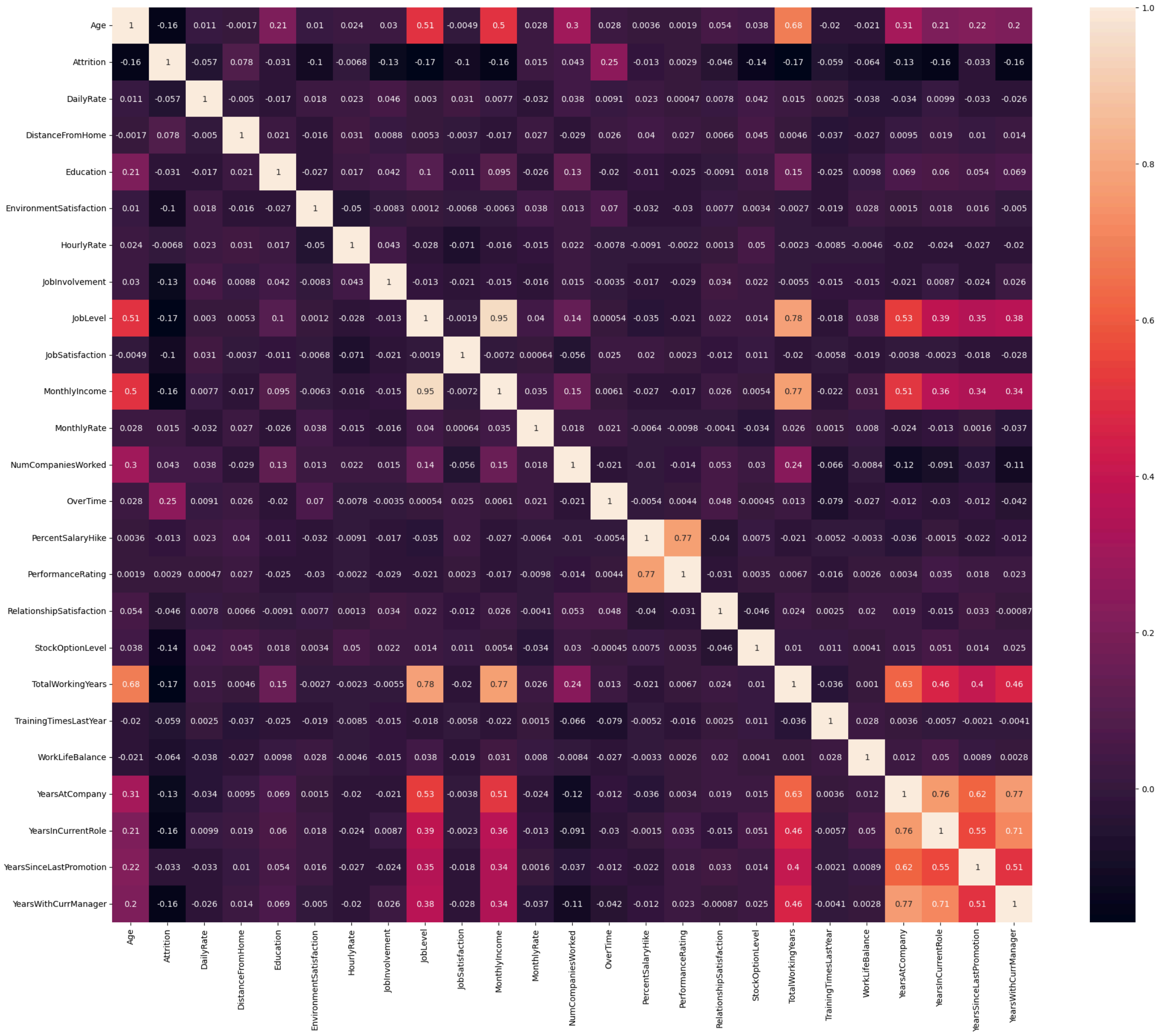
```
# Count the number of employees who stayed and left
# It seems that we are dealing with an imbalanced dataset

total_employees = len(employee_df)
left_count = len(left_df)
stayed_count = len(stayed_df)
left_percentage = (left_count / total_employees) * 100
stayed_percentage = (stayed_count / total_employees) * 100

print(f"Total employees: {total_employees}")
print(f"Number of employees who left the company: {left_count}")
print(f"Percentage of employees who left the company: {left_percentage:.2f} %")
print(f"Number of employees who did not leave the company (stayed): {stayed_coun
print(f"Percentage of employees who did not leave the company (stayed): {stayed_
```

```
Total employees: 1470
Number of employees who left the company: 237
Percentage of employees who left the company: 16.12 %
Number of employees who did not leave the company (stayed): 1233
Percentage of employees who did not leave the company (stayed): 83.88 %
```

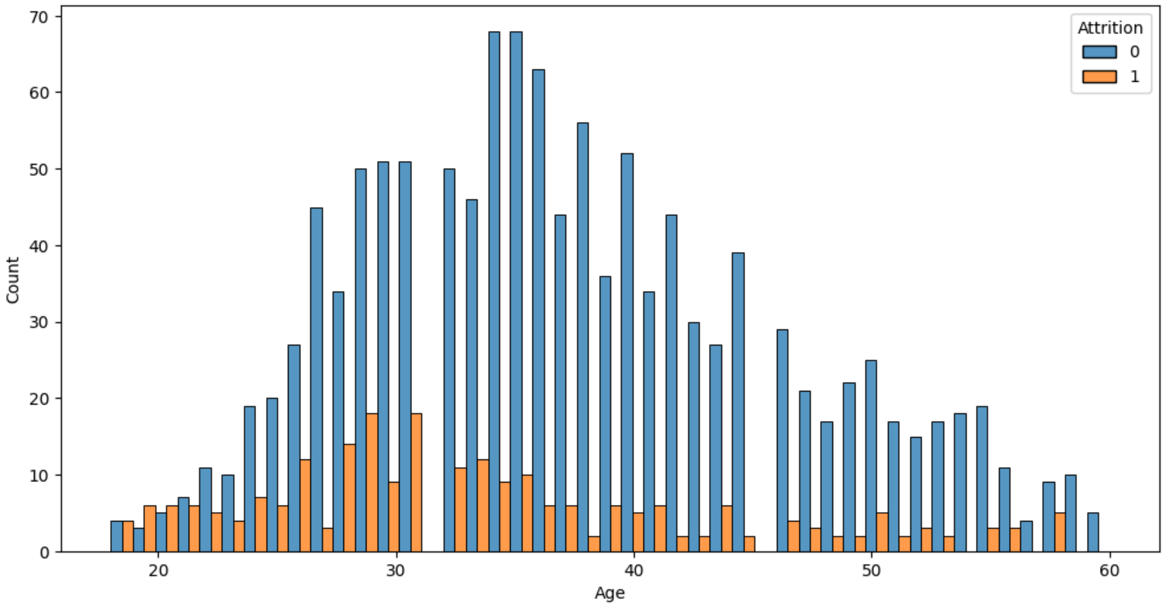
In [27]:

```
import matplotlib.pyplot as plt
import seaborn as sns
# Select Columns for Correlation
numeric_df = df.select_dtypes(include=[float, int])
# Calculate Correlation Matrix for Selected Columns
df_correlation = numeric_df.corr()

plt.figure(figsize=(25,20))
sns.heatmap(df_correlation, annot=True)
```

Out[27]: <Axes: >

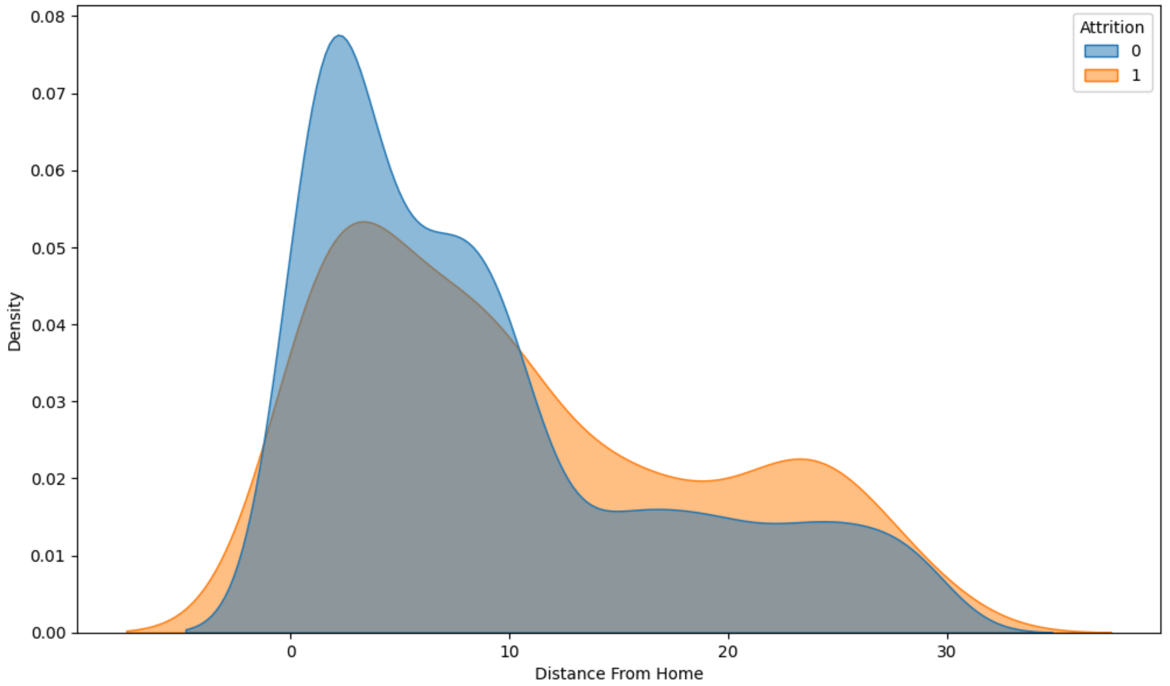


In [28]:
```python
# Plotting a histogram of the 'Age' column
plt.figure(figsize=(12, 6))
sns.histplot(data=df,x='Age',bins=45,hue='Attrition',multiple="dodge")
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```

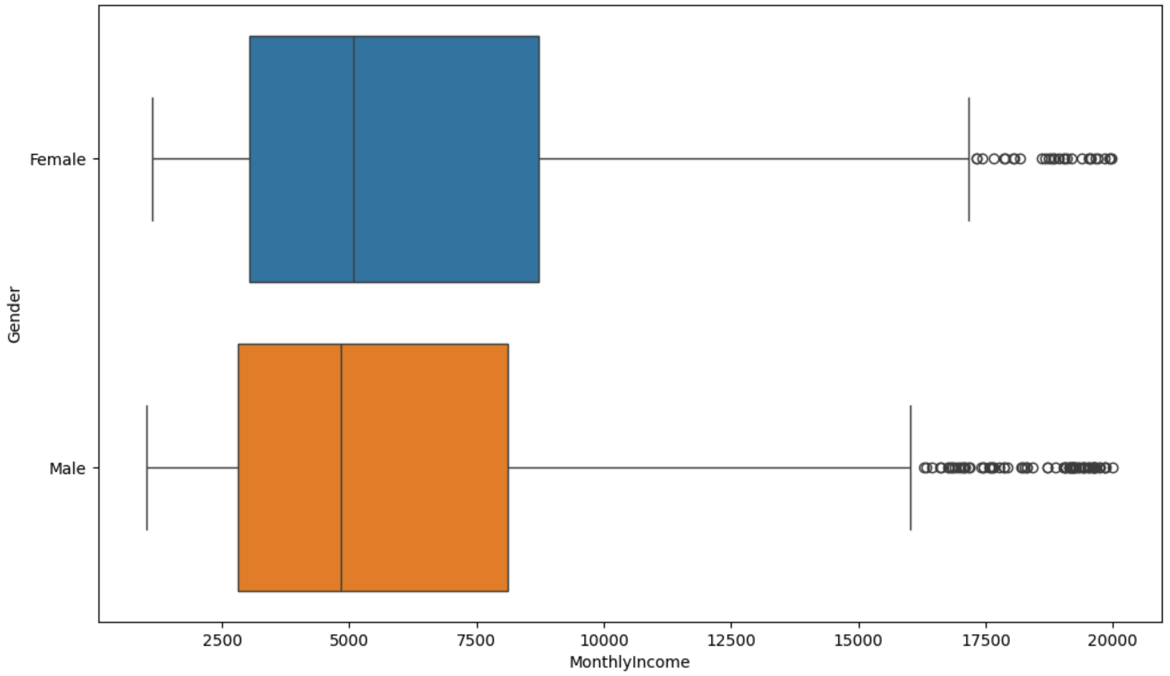


In [29]:
```python
# create a Kernel Density Estimate comparing 'Employees who left' and 'Employees
plt.figure(figsize=(12,7))
```

```
sns.kdeplot(data=df,x='DistanceFromHome',hue='Attrition',fill=True,common_norm=F
plt.xlabel('Distance From Home')
plt.ylabel('Density')
plt.show()
```



In [30]:
```
# Let's see the Gender vs. Monthly Income using box plots
plt.figure(figsize=(12,7))
sns.boxplot(data=df,x='MonthlyIncome',y='Gender',hue='Gender')
plt.xlabel('MonthlyIncome')
plt.ylabel('Gender')
plt.show()
```



Task 4: CREATE TESTING AND TRAINING DATASET & PERFORM DATA CLEANING

In [ ]:

In [32]:

```
categorical_cols = employee_df.select_dtypes(include=['object']).columns
encoder = OneHotEncoder(drop='first', sparse_output=False)
encoded_features = encoder.fit_transform(employee_df[categorical_cols])

# Convert encoded features to a DataFrame
encoded_df = pd.DataFrame(encoded_features, columns=encoder.get_feature_names_ou

# Drop original categorical columns and concatenate encoded ones
employee_df = employee_df.drop(columns=categorical_cols).reset_index(drop=True)
employee_df = pd.concat([employee_df, encoded_df], axis=1)
employee_df
# Initialize the OneHotEncoder
encoder = OneHotEncoder(sparse_output=False)


# Identify categorical columns (object dtype)
categorical_columns = df.select_dtypes(include=['object']).columns

# Fit and transform the categorical columns
encoded_data = encoder.fit_transform(df[categorical_columns])

# Create a DataFrame with the encoded data
encoded_df = pd.DataFrame(encoded_data, columns=encoder.get_feature_names_out(ca

# Drop the original categorical columns from the DataFrame
df = df.drop(categorical_columns, axis=1)

# Concatenate the original DataFrame with the encoded DataFrame
df = pd.concat([df, encoded_df], axis=1)
df
```

Out[32]:

	Age	Attrition	DailyRate	DistanceFromHome	Education	EnvironmentSatisfactio
0	41	1	1102	1	2	
1	49	0	279	8	1	
2	37	1	1373	2	2	
3	33	0	1392	3	4	
4	27	0	591	2	1	
...	...	...	...	...	...	
1465	36	0	884	23	2	
1466	39	0	613	6	1	
1467	27	0	155	4	3	
1468	49	0	1023	2	3	
1469	34	0	628	8	3	

1470 rows × 51 columns

In [34]:
```
# select your features here i.e. drop the target 'Atrittion'
X = df.drop('Attrition', axis=1)
X
```

Out[34]:

	Age	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	HourlyR
0	41	1102	1	2	2	
1	49	279	8	1	3	
2	37	1373	2	2	4	
3	33	1392	3	4	4	
4	27	591	2	1	1	
...	...	...	...	...	...	
1465	36	884	23	2	3	
1466	39	613	6	1	4	
1467	27	155	4	3	2	
1468	49	1023	2	3	4	
1469	34	628	8	3	2	

1470 rows × 50 columns

In [35]:
```
from sklearn.preprocessing import StandardScaler

# scale your features data assigning it variable X
scaler = StandardScaler()
X = scaler.fit_transform(df_features)
X
```

Out[35]:
```
array([[ 0.4463504 ,  0.74252653, -1.01090934, ..., -0.53487311,
        -0.91892141,  1.45864991],
       [ 1.32236521, -1.2977746 , -0.14714972, ..., -0.53487311,
         1.08823234, -0.68556546],
       [ 0.008343  ,  1.41436324, -0.88751511, ..., -0.53487311,
        -0.91892141,  1.45864991],
       ...,
       [-1.08667552, -1.60518328, -0.64072665, ..., -0.53487311,
         1.08823234, -0.68556546],
       [ 1.32236521,  0.54667746, -0.88751511, ..., -0.53487311,
         1.08823234, -0.68556546],
       [-0.32016256, -0.43256792, -0.14714972, ..., -0.53487311,
         1.08823234, -0.68556546]])
```

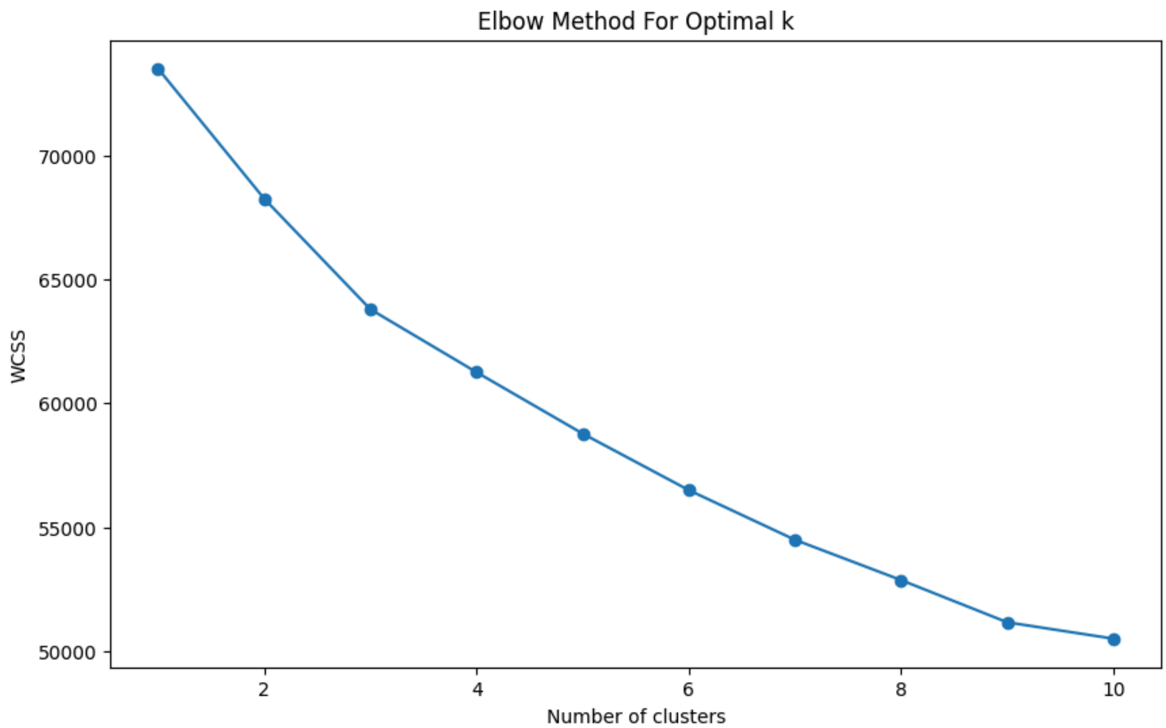
In [36]:
```
# select your dependent, target or response data as "Attrition" using variable y
y= df['Attrition']
y
```

Out[36]:
```
0       1
1       0
2       1
3       0
4       0
       ..
1465    0
1466    0
1467    0
1468    0
1469    0
Name: Attrition, Length: 1470, dtype: int64
```

FIND THE OPTIMAL NUMBER OF CLUSTERS USING ELBOW METHOD

In [41]:
```
from sklearn.cluster import KMeans

# Compute 'within cluster sum of squares' or WCSS metric for a range of k cluste
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, ran
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

In [42]:
```
# Create a visualization for Finding the right number of clusters - Elbow method
plt.figure(figsize=(10, 6))
plt.plot(range(1, 11), wcss, marker='o')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```



APPLY K-MEANS METHOD

In [43]:
```
# Apply KMeans with the optimal number of clusters (e.g., 3)
optimal_clusters = 3
kmeans = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=300, n_i
kmeans.fit(X)

# Add the cluster labels to the dataframe
employee_df['Cluster'] = kmeans.labels_

# Display the first few rows of the dataframe with the cluster labels
employee_df.head()
```

Out[43]:

	Age	Attrition	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	H
0	41	1	1102	1	2	2	
1	49	0	279	8	1	3	
2	37	1	1373	2	2	4	
3	33	0	1392	3	4	4	
4	27	0	591	2	1	1	

5 rows × 46 columns

In [ ]:
```
# Check size of each cluster - Are they all representative ?
# Check the size of each cluster
cluster_sizes = employee_df['Cluster'].value_counts()
print(cluster_sizes)
```

```
Cluster
0    818
2    399
1    253
Name: count, dtype: int64
```

In [ ]:

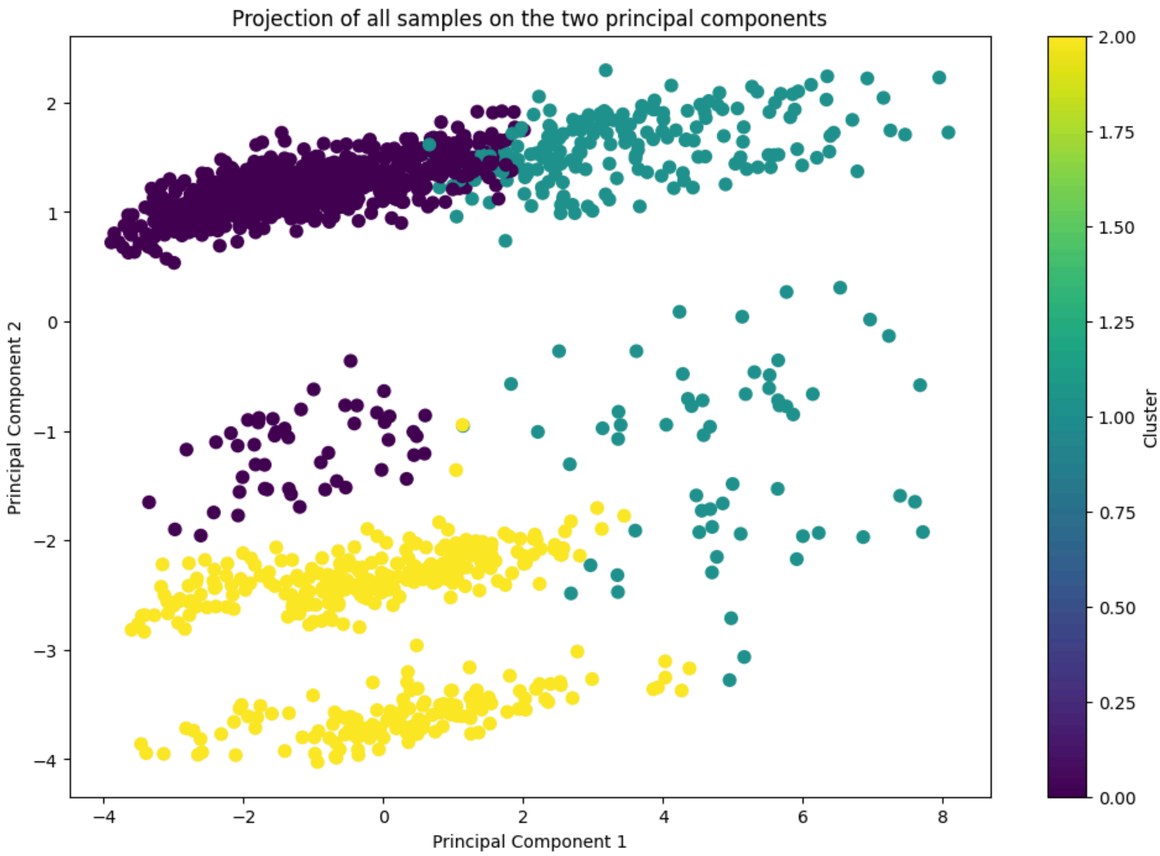
In [ ]:

APPLY PRINCIPAL COMPONENT ANALYSIS AND VISUALIZE THE RESULTS

In [46]:
```
# Obtain the principal components
pca = PCA(n_components=2)
principal_components = pca.fit_transform(X)
principal_components
```

Out[46]:
```
array([[-0.0234771 , -2.29498375],
       [ 0.04824099,  1.53280418],
       [-2.91944898,  1.0023716 ],
       ...,
       [-1.06745272,  1.27022263],
       [ 1.18308284, -1.99420826],
       [-1.48078064,  1.10151924]])
```

In [48]:
```
# All samples projected on the two principal components
plt.figure(figsize=(12, 8))
plt.scatter(principal_components[:, 0], principal_components[:, 1], c=kmeans.lab
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Projection of all samples on the two principal components')
plt.colorbar(label='Cluster')
plt.show()
principal_components.shape
```



Out[48]: (1470, 2)

In [49]:
```
# Create a dataframe with the two components
pca_df = pd.DataFrame(data=principal_components, columns=['Principal Component 1
pca_df.head()
```

Out[49]:

	Principal Component 1	Principal Component 2
0	-0.023477	-2.294984
1	0.048241	1.532804
2	-2.919449	1.002372
3	-1.185131	1.061926
4	-2.124228	1.178367

In [51]:
```
# Concatenate the clusters labels to the dataframe
employee_df = pd.concat([employee_df, pd.DataFrame(kmeans.labels_, columns=['Clu
employee_df.head()
```

Out[51]:

	Age	Attrition	DailyRate	DistanceFromHome	Education	EnvironmentSatisfaction	H
0	41	1	1102	1	2	2	
1	49	0	279	8	1	3	
2	37	1	1373	2	2	4	
3	33	0	1392	3	4	4	
4	27	0	591	2	1	1	

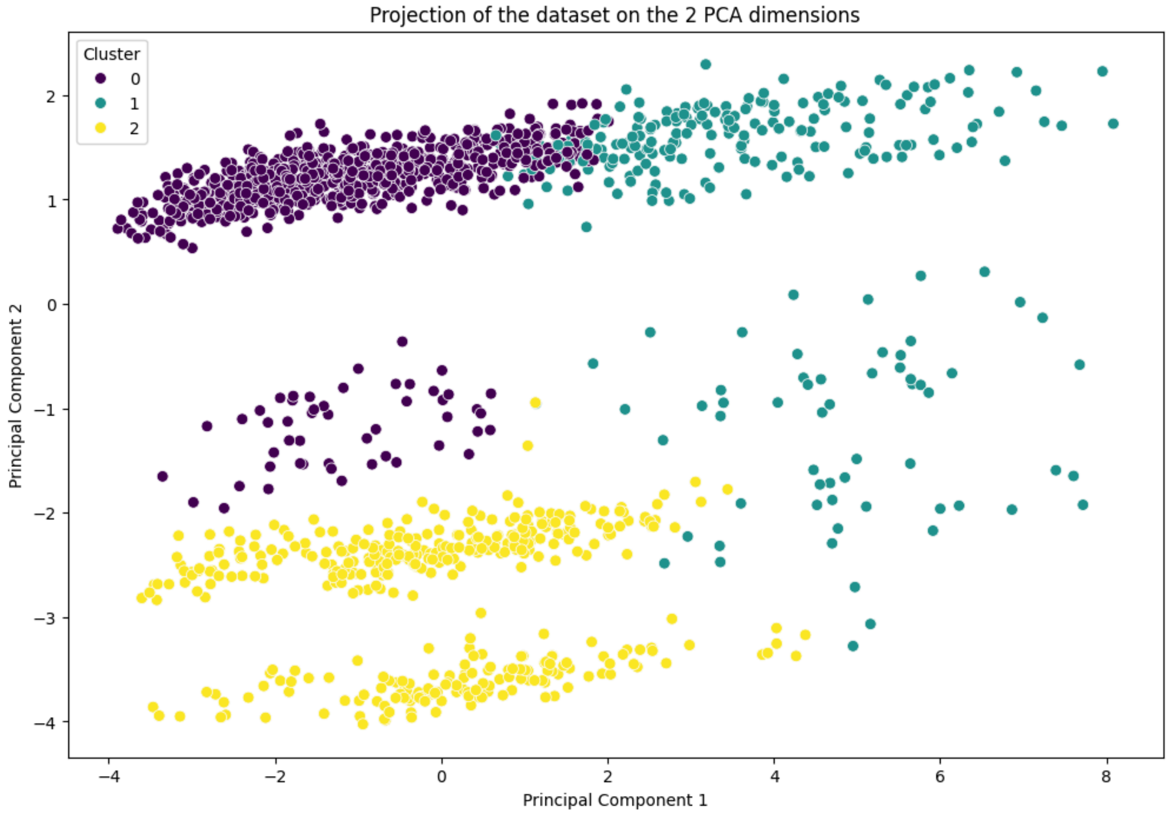
5 rows × 48 columns

In [ ]:

In [ ]:

In [ ]:

In [52]:
```
# Create a scatterplot visual of Projection of the dataset on the 2 PCA dimensio
plt.figure(figsize=(12, 8))
sns.scatterplot(x=principal_components[:, 0], y=principal_components[:, 1], hue=
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Projection of the dataset on the 2 PCA dimensions')
plt.legend(title='Cluster')
plt.show()
```



In [53]:
```
# show the % of the total variance explained by each principal component. Overal
explained_variance = pca.explained_variance_ratio_
print(f"Explained variance by each component: {explained_variance}")
print(f"Total explained variance by the first two components: {explained_varianc
```

```
Explained variance by each component: [0.10704652 0.07010867]
Total explained variance by the first two components: 17.72%
```

AUTOENCODERS (PERFORM DIMENSIONALITY REDUCTION USING AUTOENCODERS)

In [54]:
```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.optimizers import Adam

#import the autoencoder libraries
```

In [57]:
```
# create your autoencoder with all the features showing Encoder, bottleneck, dec
# compile the autoencoder using optimizer='adam', loss='mean_squared_error'
input_dim = encoded_data.shape[1]
encoding_dim = 14  # This can be adjusted

# Encoder
input_layer = Input(shape=(input_dim,))
encoder = Dense(encoding_dim, activation='relu')(input_layer)

# Bottleneck
bottleneck = Dense(encoding_dim // 2, activation='relu')(encoder)

# Decoder
decoder = Dense(encoding_dim, activation='relu')(bottleneck)
output_layer = Dense(input_dim, activation='sigmoid')(decoder)
```

```
# Autoencoder
autoencoder = Model(inputs=input_layer, outputs=output_layer)

# Compile the autoencoder
autoencoder.compile(optimizer='adam', loss='mean_squared_error')
```

In [59]:
```
# show the autoencoder summary
# show the autoencoder summary
autoencoder.summary()
```

Model: "functional_1"

Layer (type)	Output Shape	Param #
input_layer_1 (InputLayer)	(None, 26)	0
dense_4 (Dense)	(None, 14)	378
dense_5 (Dense)	(None, 7)	105
dense_6 (Dense)	(None, 14)	112
dense_7 (Dense)	(None, 26)	390

Total params: 985 (3.85 KB)

Trainable params: 985 (3.85 KB)

Non-trainable params: 0 (0.00 B)

In [60]:
```
## Train autoencoder using input = output
# Train the autoencoder
history = autoencoder.fit(encoded_data, encoded_data, epochs=50, batch_size=32,
```

```
Epoch 1/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 2s 11ms/step - loss: 0.2474 - val_loss: 0.2274
Epoch 2/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.2166 - val_loss: 0.1735
Epoch 3/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1611 - val_loss: 0.1373
Epoch 4/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1322 - val_loss: 0.1259
Epoch 5/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1219 - val_loss: 0.1177
Epoch 6/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1143 - val_loss: 0.1100
Epoch 7/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.1055 - val_loss: 0.1029
Epoch 8/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.1024 - val_loss: 0.0973
Epoch 9/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0951 - val_loss: 0.0921
Epoch 10/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0905 - val_loss: 0.0869
Epoch 11/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0852 - val_loss: 0.0818
Epoch 12/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0796 - val_loss: 0.0761
Epoch 13/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0734 - val_loss: 0.0706
Epoch 14/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0681 - val_loss: 0.0664
Epoch 15/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0639 - val_loss: 0.0630
Epoch 16/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0635 - val_loss: 0.0601
Epoch 17/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step - loss: 0.0609 - val_loss: 0.0577
Epoch 18/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0583 - val_loss: 0.0555
Epoch 19/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0540 - val_loss: 0.0531
Epoch 20/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0528 - val_loss: 0.0509
Epoch 21/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0507 - val_loss: 0.0488
Epoch 22/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0495 - val_loss: 0.0470
Epoch 23/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0478 - val_loss: 0.0455
Epoch 24/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0459 - val_loss: 0.0441
Epoch 25/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0429 - val_loss: 0.0430
Epoch 26/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0415 - val_loss: 0.0420
Epoch 27/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0409 - val_loss: 0.0410
Epoch 28/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 7ms/step - loss: 0.0388 - val_loss: 0.0402
Epoch 29/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0385 - val_loss: 0.0393
Epoch 30/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0387 - val_loss: 0.0385
```

```
Epoch 31/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0370 - val_loss: 0.0378
Epoch 32/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0371 - val_loss: 0.0371
Epoch 33/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0358 - val_loss: 0.0363
Epoch 34/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0353 - val_loss: 0.0358
Epoch 35/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0342 - val_loss: 0.0350
Epoch 36/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0332 - val_loss: 0.0342
Epoch 37/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0329 - val_loss: 0.0334
Epoch 38/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0320 - val_loss: 0.0330
Epoch 39/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0328 - val_loss: 0.0321
Epoch 40/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0311 - val_loss: 0.0313
Epoch 41/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0294 - val_loss: 0.0307
Epoch 42/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0288 - val_loss: 0.0304
Epoch 43/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0287 - val_loss: 0.0297
Epoch 44/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0285 - val_loss: 0.0293
Epoch 45/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - loss: 0.0273 - val_loss: 0.0289
Epoch 46/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0283 - val_loss: 0.0291
Epoch 47/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0279 - val_loss: 0.0283
Epoch 48/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0267 - val_loss: 0.0282
Epoch 49/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - loss: 0.0271 - val_loss: 0.0281
Epoch 50/50
37/37 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - loss: 0.0271 - val_loss: 0.0276
```

In [61]:

```
# Use Autoencoder to reduce the number of features / dimensions and show the dim
# Use the encoder part of the autoencoder to reduce the dimensions
encoder_model = Model(inputs=input_layer, outputs=bottleneck)
encoded_data_reduced = encoder_model.predict(encoded_data)

# Show the dimensions of the reduced data
print(f"Original dimensions: {encoded_data.shape[1]}")
print(f"Reduced dimensions: {encoded_data_reduced.shape[1]}")
```

```
46/46 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step
Original dimensions: 26
Reduced dimensions: 7
```

Apply KMEANS to encoded dataset

In [62]:

```
# Apply KMEANS to encoded dataset here
# Apply KMeans with the optimal number of clusters to the encoded dataset
kmeans_encoded = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=
```

```
kmeans_encoded.fit(encoded_data_reduced)

# Add the cluster labels to the dataframe
encoded_df['Cluster'] = kmeans_encoded.labels_

# Display the first few rows of the dataframe with the cluster labels
encoded_df.head()
```

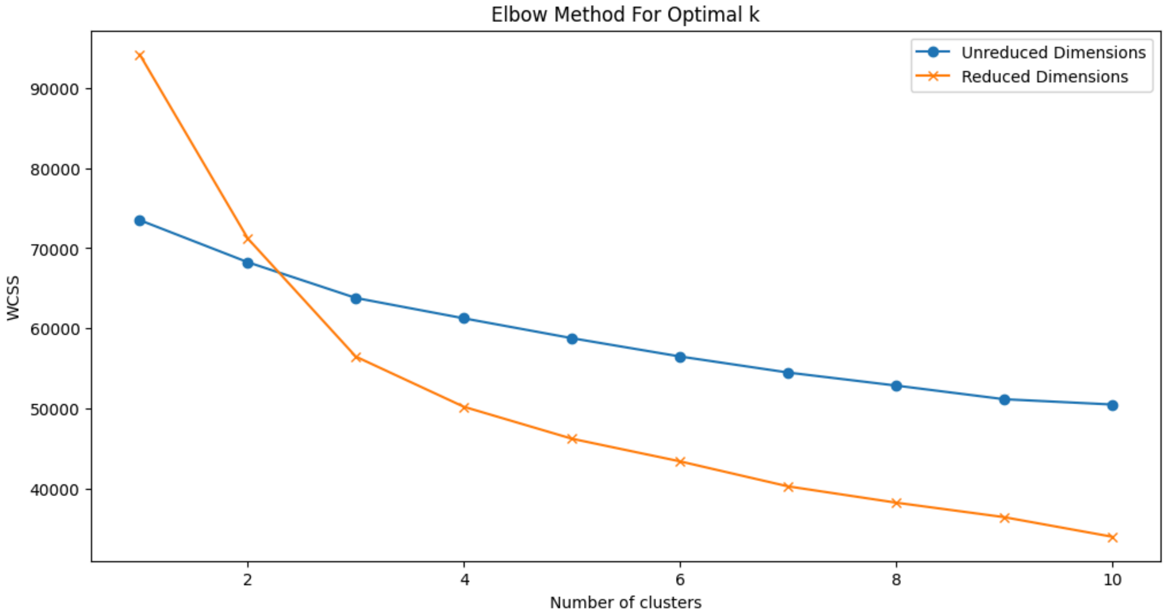
Out[62]:

	BusinessTravel_Non-Travel	BusinessTravel_Travel_Frequently	BusinessTravel_Travel_Rarely	D
0	0.0	0.0	1.0	
1	0.0	1.0	0.0	
2	0.0	0.0	1.0	
3	0.0	1.0	0.0	
4	0.0	0.0	1.0	

5 rows × 27 columns

In [63]:

```
# create a line plot to show the " Pick optimal number of clusters using Elbow m
# Compute WCSS for the reduced dimension data
wcss_reduced = []
for i in range(1, 11):
    kmeans_reduced = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init
    kmeans_reduced.fit(encoded_data_reduced)
    wcss_reduced.append(kmeans_reduced.inertia_)

# Plot the WCSS for both unreduced and reduced dimension data
plt.figure(figsize=(12, 6))
plt.plot(range(1, 11), wcss, marker='o', label='Unreduced Dimensions')
plt.plot(range(1, 11), wcss_reduced, marker='x', label='Reduced Dimensions')
plt.title('Elbow Method For Optimal k')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.legend()
plt.show()
```



In [64]:
```
## Apply the resulting optimal k to find new centroids

# Apply KMeans with the optimal number of clusters to the encoded dataset
kmeans_encoded = KMeans(n_clusters=optimal_clusters, init='k-means++', max_iter=
kmeans_encoded.fit(encoded_data_reduced)

# Get the centroids of the clusters
centroids = kmeans_encoded.cluster_centers_
print("Centroids of the clusters:")
print(centroids)
```

```
Centroids of the clusters:
[[ 7.3965936   3.8773718   1.5245227   6.6510777   8.427682    9.288008
  14.469268 ]
 [ 4.744938    4.3343806   2.8834443   5.316986    4.882946    4.7686677
  10.2445755]
 [ 3.3931005   9.726853    3.0882022   5.2858133   5.7823577 11.105503
   8.543496 ]]
```

In [ ]:
```
## Show the centroids shape
print("Shape of the centroids:", centroids.shape)
```

```
Shape of the centroids: (3, 7)
```

In [66]:
```
# show the clusters shape
print("Shape of the clusters:", encoded_df['Cluster'].shape)
```

```
Shape of the clusters: (1470,)
```

In [67]:
```
# concatenate the clusters to the data
# Concatenate the clusters to the encoded data
encoded_data_with_clusters = np.hstack((encoded_data_reduced, kmeans_encoded.lab
encoded_data_with_clusters
```

Out[67]:
```
array([[ 4.10488987,  9.01912117,  4.68139458, ..., 16.58912277,
        11.01517487,  2.        ],
       [ 5.63051701,  3.05288029,  3.82769704, ...,  0.71763122,
        16.25865555,  1.        ],
       [ 4.22196341,  1.4237268 ,  2.78904057, ...,  8.59499741,
         7.54731846,  1.        ],
       ...,
       [ 4.98810005,  5.4479866 ,  1.40472043, ...,  7.03517246,
        14.31093597,  0.        ],
       [ 0.96433985,  9.34902   ,  4.60331345, ...,  3.13075495,
         8.72610664,  1.        ],
       [ 2.35847521,  3.83578587,  1.95530748, ...,  5.04445076,
        11.84350395,  1.        ]])
```

In [68]:
```
# show the 'Number of samples" in your current consolidated
print(f"Number of samples in encoded_data: {encoded_data.shape[0]}")
print(f"Number of samples in encoded_data_reduced: {encoded_data_reduced.shape[0
print(f"Number of samples in encoded_data_with_clusters: {encoded_data_with_clus
print(f"Number of samples in encoded_df: {encoded_df.shape[0]}")
```

```
Number of samples in encoded_data: 1470
Number of samples in encoded_data_reduced: 1470
Number of samples in encoded_data_with_clusters: 1470
Number of samples in encoded_df: 1470
```

In [69]:
```
## Apply PCA to encoded dataset
# Apply PCA to the encoded dataset
pca = PCA(n_components=2)
principal_components_encoded = pca.fit_transform(encoded_data_reduced)

# Create a DataFrame with the two principal components
pca_encoded_df = pd.DataFrame(data=principal_components_encoded, columns=['Princ

# Concatenate the cluster labels to the DataFrame
pca_encoded_df = pd.concat([pca_encoded_df, pd.DataFrame(kmeans_encoded.labels_,

# Display the first few rows of the DataFrame
pca_encoded_df.head()
```

Out[69]:

	Principal Component 1	Principal Component 2	Cluster
0	-2.724478	8.283110	2
1	4.913633	-6.735337	1
2	-0.841062	-1.577970	1
3	7.145702	-4.268825	0
4	0.022470	-4.115398	1

In [72]:
```
# concatenate the clusters to the data
# Concatenate the clusters to the encoded data
encoded_data_with_clusters = np.hstack((encoded_data_reduced, kmeans_encoded.lab
encoded_data_with_clusters
```

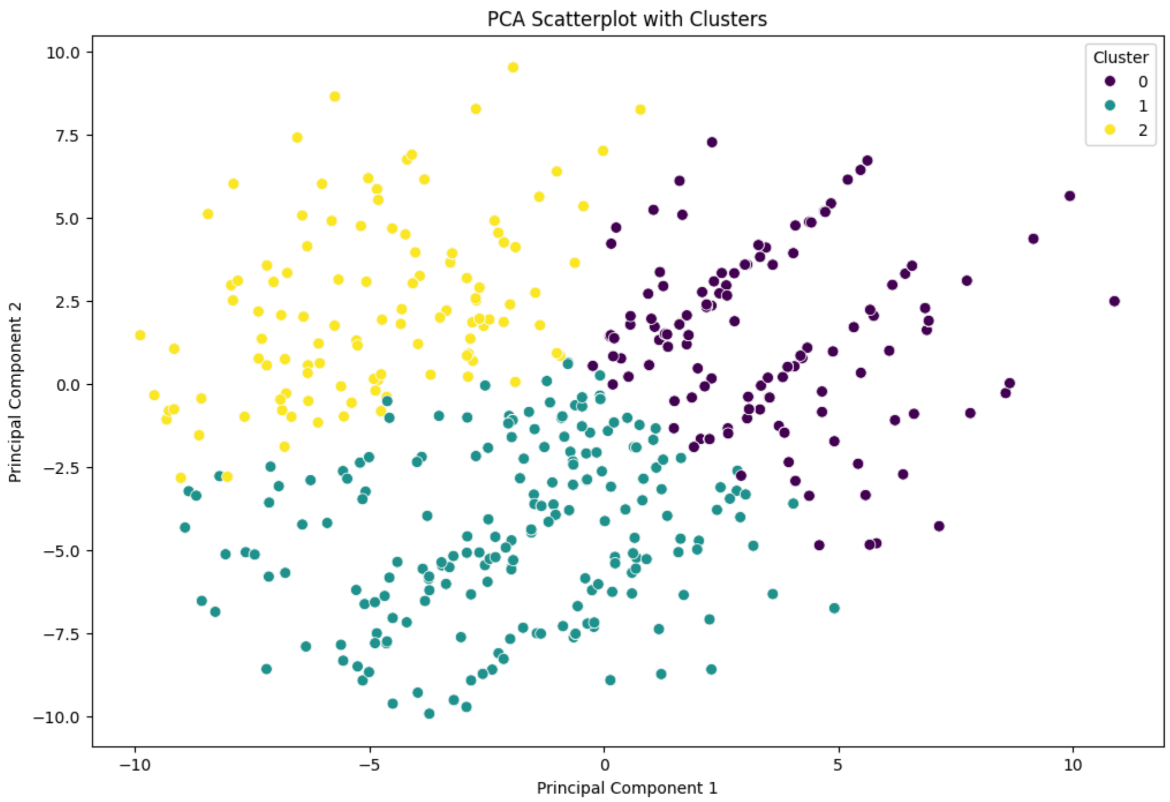
Out[72]:
```
array([[ 4.10488987,  9.01912117,  4.68139458, ..., 16.58912277,
        11.01517487,  2.        ],
       [ 5.63051701,  3.05288029,  3.82769704, ...,  0.71763122,
        16.25865555,  1.        ],
       [ 4.22196341,  1.4237268 ,  2.78904057, ...,  8.59499741,
         7.54731846,  1.        ],
       ...,
       [ 4.98810005,  5.4479866 ,  1.40472043, ...,  7.03517246,
        14.31093597,  0.        ],
       [ 0.96433985,  9.34902   ,  4.60331345, ...,  3.13075495,
         8.72610664,  1.        ],
       [ 2.35847521,  3.83578587,  1.95530748, ...,  5.04445076,
        11.84350395,  1.        ]])
```

In [73]:
```python
## Apply PCA to encoded dataset
# Apply PCA to the encoded dataset
pca = PCA(n_components=2)
principal_components_encoded = pca.fit_transform(encoded_data_reduced)

# Create a DataFrame with the two principal components
pca_encoded_df = pd.DataFrame(data=principal_components_encoded, columns=['Princ

# Concatenate the cluster labels to the DataFrame
pca_encoded_df = pd.concat([pca_encoded_df, pd.DataFrame(kmeans_encoded.labels_,

# Display the first few rows of the DataFrame
pca_encoded_df.head()
```

Out[73]:

	Principal Component 1	Principal Component 2	Cluster
0	-2.724478	8.283110	2
1	4.913633	-6.735337	1
2	-0.841062	-1.577970	1
3	7.145702	-4.268825	0
4	0.022470	-4.115398	1

In [74]:
```python
## Plot your pca scatterplot with clusters as the hue
plt.figure(figsize=(12, 8))
sns.scatterplot(data=pca_encoded_df, x='Principal Component 1', y='Principal Com
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA Scatterplot with Clusters')
plt.legend(title='Cluster')
plt.show()
```


PCA Scatterplot with Clusters
Principal Component 2
Principal Component 1
Cluster
0
1
2
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
−10
−5
0
5
10