Libraries

```
import numpy as np
import pandas as pd
#Visualization
import missingno as msno
import seaborn as sns
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
#Hot Encoding Technique
from sklearn.preprocessing import OneHotEncoder
#Feature Scaling
from sklearn.preprocessing import StandardScaler
#Training and testing
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
#ANN
from keras.models import Sequential
from keras.layers import Dense
#GBM
import xgboost as xgb
#ensemble Learning
from sklearn.ensemble import RandomForestClassifier
#Cluster Analysis
from sklearn.cluster import KMeans
#PCA Feature Selection
from sklearn.decomposition import PCA
#Confusion Matrix
from sklearn.metrics import confusion_matrix
```

Load the Dataset

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
pd.set_option('display.max_columns', None)
df = pd.read_csv("/content/drive/MyDrive/machinelearning/Uzair_work/Assignment_1_task_1/bank-additional.csv",sep=";")
df.head()
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	po
0	56	housemaid	married	basic.4y	no	no	no	telephone	may	mon	261	1	999	0	nor
1	57	services	married	high.school	unknown	no	no	telephone	may	mon	149	1	999	0	nor
2	37	services	married	high.school	no	yes	no	telephone	may	mon	226	1	999	0	nor
3	40	admin.	married	basic.6y	no	no	no	telephone	may	mon	151	1	999	0	nor
4	56	services	married	high.school	no	no	yes	telephone	may	mon	307	1	999	0	nor

```
df.tail(5)
```

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	prev
41183	73	retired	married	professional.course	no	yes	no	cellular	nov	fri	334	1	999	
41184	46	blue-collar	married	professional.course	no	no	no	cellular	nov	fri	383	1	999	
41185	56	retired	married	university.degree	no	yes	no	cellular	nov	fri	189	2	999	
41186	44	technician	married	professional.course	no	no	no	cellular	nov	fri	442	1	999	
41187	74	retired	married	professional.course	no	yes	no	cellular	nov	fri	239	3	999	

```
df.columns.tolist()
```

```
['age',
 'job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'poutcome',
 'emp.var.rate',
 'cons.price.idx',
 'cons.conf.idx',
 'euribor3m',
 'nr.employed',
 'y']
```

```
print("Number of Observations : ", df.shape[0])
print("Number of Atributes : ", df.shape[1])
```

```
Number of Observations :  41188
Number of Atributes :  21
```

```
df.dtypes
```

```
age                 int64
job                object
marital            object
education          object
default            object
housing            object
loan               object
contact            object
month              object
day_of_week        object
duration            int64
campaign            int64
pdays               int64
previous            int64
poutcome           object
emp.var.rate      float64
cons.price.idx    float64
cons.conf.idx     float64
euribor3m         float64
nr.employed       float64
y                  object
dtype: object
```

```
df.isna().sum()
```

```
age               0
job               0
marital           0
education         0
default           0
housing           0
loan              0
contact           0
month             0
day_of_week       0
duration          0
campaign          0
pdays             0
previous          0
poutcome          0
emp.var.rate      0
cons.price.idx    0
cons.conf.idx     0
euribor3m         0
nr.employed       0
y                 0
dtype: int64
```

```
col_names = df.columns.tolist()
print(col_names)
```

```
['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pda
```

```
categorical_cols = df.select_dtypes(include=['object']).columns.tolist()

print('Categorical columns:', categorical_cols)
```

```
Categorical columns: ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome', 'y
```

```
Number_cols = []
for i in col_names:
    if i not in categorical_cols:
        Number_cols.append(i)

print('Numerical columns:', Number_cols)
```

```
Numerical columns: ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3
```

Categorical Dataset

```
categorical_cols = ['job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'poutcome']
df_categorical = df[categorical_cols]
```

```
df_categorical.head()
```

	job	marital	education	default	housing	loan	contact	month	day_of_week	poutcome
0	housemaid	married	basic.4y	no	no	no	telephone	may	mon	nonexistent
1	services	married	high.school	unknown	no	no	telephone	may	mon	nonexistent
2	services	married	high.school	no	yes	no	telephone	may	mon	nonexistent
3	admin.	married	basic.6y	no	no	no	telephone	may	mon	nonexistent
4	services	married	high.school	no	no	yes	telephone	may	mon	nonexistent

```
df_encoded = pd.get_dummies(df_categorical, columns=categorical_cols)
```

```
Number_cols = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx', 'cons.conf.idx', 'euribor3m', 'nr.e
df_pre = pd.concat([df_encoded, df[Number_cols]], axis=1)
```

```
scaler = StandardScaler()
df_pre[Number_cols] = scaler.fit_transform(df_pre[Number_cols])
```

```
df_pre.columns.tolist()
```

```
['job_admin.',
 'job_blue-collar',
 'job_entrepreneur',
 'job_housemaid',
 'job_management',
 'job_retired',
 'job_self-employed',
 'job_services',
 'job_student',
 'job_technician',
 'job_unemployed',
 'job_unknown',
 'marital_divorced',
 'marital_married',
 'marital_single',
 'marital_unknown',
 'education_basic.4y',
 'education_basic.6y',
 'education_basic.9y',
 'education_high.school',
 'education_illiterate',
 'education_professional.course',
 'education_university.degree',
 'education_unknown',
 'default_no',
 'default_unknown',
 'default_yes',
 'housing_no',
 'housing_unknown',
 'housing_yes',
 'loan_no',
 'loan_unknown',
 'loan_yes',
 'contact_cellular',
 'contact_telephone',
```

```
       'month_apr',
       'month_aug',
       'month_dec',
       'month_jul',
       'month_jun',
       'month_mar',
       'month_may',
       'month_nov',
       'month_oct',
       'month_sep',
       'day_of_week_fri',
       'day_of_week_mon',
       'day_of_week_thu',
       'day_of_week_tue',
       'day_of_week_wed',
       'poutcome_failure',
       'poutcome_nonexistent',
       'poutcome_success',
       'age',
       'duration',
       'campaign',
       'pdays',
       'previous',
```

```
df_pre.shape
```

```
(41188, 63)
```

```
# Convert target variable to numeric values
# y = y.replace({"yes": 1, "no": 0})
```

```
X = df_pre
y = df['y']
y = y.replace({"yes": 1, "no": 0})

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the ANN model
model = Sequential()

# Add input layer and hidden layers
model.add(Dense(units=64, activation='relu', input_dim=X_train.shape[1]))
model.add(Dense(units=64, activation='relu'))
model.add(Dense(units=64, activation='relu'))

# Add output layer
model.add(Dense(units=1, activation='sigmoid'))

# Compile the model
# model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
# model.compile(optimizer='adam', loss='mae', metrics=['accuracy'])
model.compile(optimizer='adam', loss='mse', metrics=['accuracy'])
```

```
# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32)

# Evaluate the model on the testing set
loss, accuracy_ann = model.evaluate(X_test, y_test)
print("Loss:", loss)
print("Accuracy of Artificial Neural Network : {:.2f}%".format(accuracy_ann * 100))
```

```
Epoch 1/10
1030/1030 [==============================] - 3s 2ms/step - loss: 0.0654 - accuracy: 0.9063
Epoch 2/10
1030/1030 [==============================] - 2s 2ms/step - loss: 0.0588 - accuracy: 0.9130
Epoch 3/10
1030/1030 [==============================] - 2s 2ms/step - loss: 0.0575 - accuracy: 0.9156
Epoch 4/10
1030/1030 [==============================] - 2s 2ms/step - loss: 0.0566 - accuracy: 0.9164
Epoch 5/10
1030/1030 [==============================] - 3s 3ms/step - loss: 0.0553 - accuracy: 0.9194
Epoch 6/10
1030/1030 [==============================] - 2s 2ms/step - loss: 0.0545 - accuracy: 0.9209
Epoch 7/10
1030/1030 [==============================] - 2s 2ms/step - loss: 0.0533 - accuracy: 0.9228
Epoch 8/10
1030/1030 [==============================] - 2s 2ms/step - loss: 0.0524 - accuracy: 0.9254
Epoch 9/10
1030/1030 [==============================] - 2s 2ms/step - loss: 0.0514 - accuracy: 0.9275
Epoch 10/10
```

```
1030/1030 [==============================] - 2s 2ms/step - loss: 0.0503 - accuracy: 0.9282
258/258 [==============================] - 1s 2ms/step - loss: 0.0632 - accuracy: 0.9077
Loss: 0.06321562081575394
Accuracy of Artificial Neural Network : 90.77%
```

```
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 64)                4096

 dense_5 (Dense)             (None, 64)                4160

 dense_6 (Dense)             (None, 64)                4160

 dense_7 (Dense)             (None, 1)                 65

=================================================================
Total params: 12,481
Trainable params: 12,481
Non-trainable params: 0
_________________________________________________________________
None
```

Confussion Matrix of Artifical Neural Network

```
y_pred = model.predict(X_test)

y_pred_binary = np.squeeze(y_pred > 0.5).astype(int)


confusion_mat = confusion_matrix(y_test, y_pred_binary)
print("Confusion Matrix:")
print(confusion_mat)
```

```
258/258 [==============================] - 1s 2ms/step
Confusion Matrix:
[[7148  155]
 [ 605  330]]
```

```
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score

# Assuming you have the confusion matrix stored in the variable 'confusion_mat'
TN, FP, FN, TP = confusion_mat.ravel()

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred_binary)

# Calculate precision
precision = precision_score(y_test, y_pred_binary)

# Calculate recall
recall = recall_score(y_test, y_pred_binary)

# Calculate F1-score
f1 = f1_score(y_test, y_pred_binary)

# Print the evaluation metrics
print("Accuracy:", accuracy*100)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
```

```
Accuracy: 90.77445982034475
Precision: 0.6804123711340206
Recall: 0.35294117647058826
F1-score: 0.46478873239436624
```

Gradient Boosting: *

```
model = xgb.XGBClassifier()
model.fit(X_train, y_train)
y_pred_gb = model.predict(X_test)
```

```
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print("Accuracy of Gradient Boosting : {:.2f}%".format(accuracy_gb * 100))
```

```
Accuracy of Gradient Boosting : 91.56%
```

Confusion Matrix for Gradient Boosting:

```
confusion_mat_gb = confusion_matrix(y_test, y_pred_gb)
print("Confusion Matrix of Gradient Boosting:")
print(confusion_mat_gb)
```

```
Confusion Matrix of Gradient Boosting:
[[7032  271]
 [ 424  511]]
```

```
TN = 7032
FP = 271
FN = 424
TP = 511

precision = TP / (TP + FP)
recall = TP / (TP + FN)
f1_score = 2 * (precision * recall) / (precision + recall)

print("F1-score for Gradient Boosting : ", f1_score)
```

```
F1-score for Gradient Boosting :  0.5952242283051835
```

```
ensemble = RandomForestClassifier(n_estimators=5)  # Adjust the number of estimators as needed

# Train the ensemble on the training data
ensemble.fit(X_train, y_train)

# Make predictions on the test data
y_pred_en = ensemble.predict(X_test)

# Calculate accuracy
accuracy_en = accuracy_score(y_test, y_pred_en)


# Print the accuracy
print("Accuracy of Ensemble Learning : {:.2f}%".format(accuracy_gb * 100))
```

```
Accuracy of Ensemble Learning : 91.56%
```

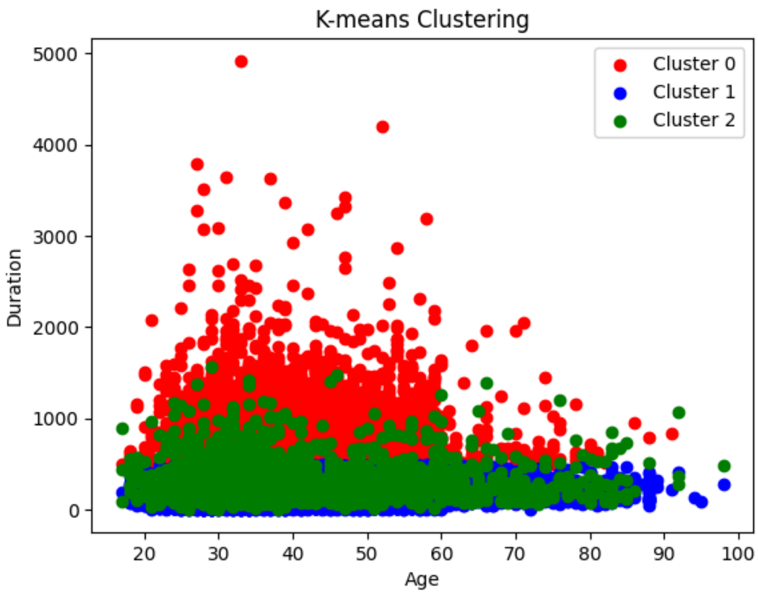
Cluster Analysis

```
cols = df.columns.tolist()
cols = ['age',
 'job',
 'marital',
 'education',
 'default',
 'housing',
 'loan',
 'contact',
 'month',
 'day_of_week',
 'duration',
 'campaign',
 'pdays',
 'previous',
 'poutcome',
 'emp.var.rate',
 'cons.price.idx',
 'cons.conf.idx',
 'euribor3m',
 'nr.employed',
 ]
```

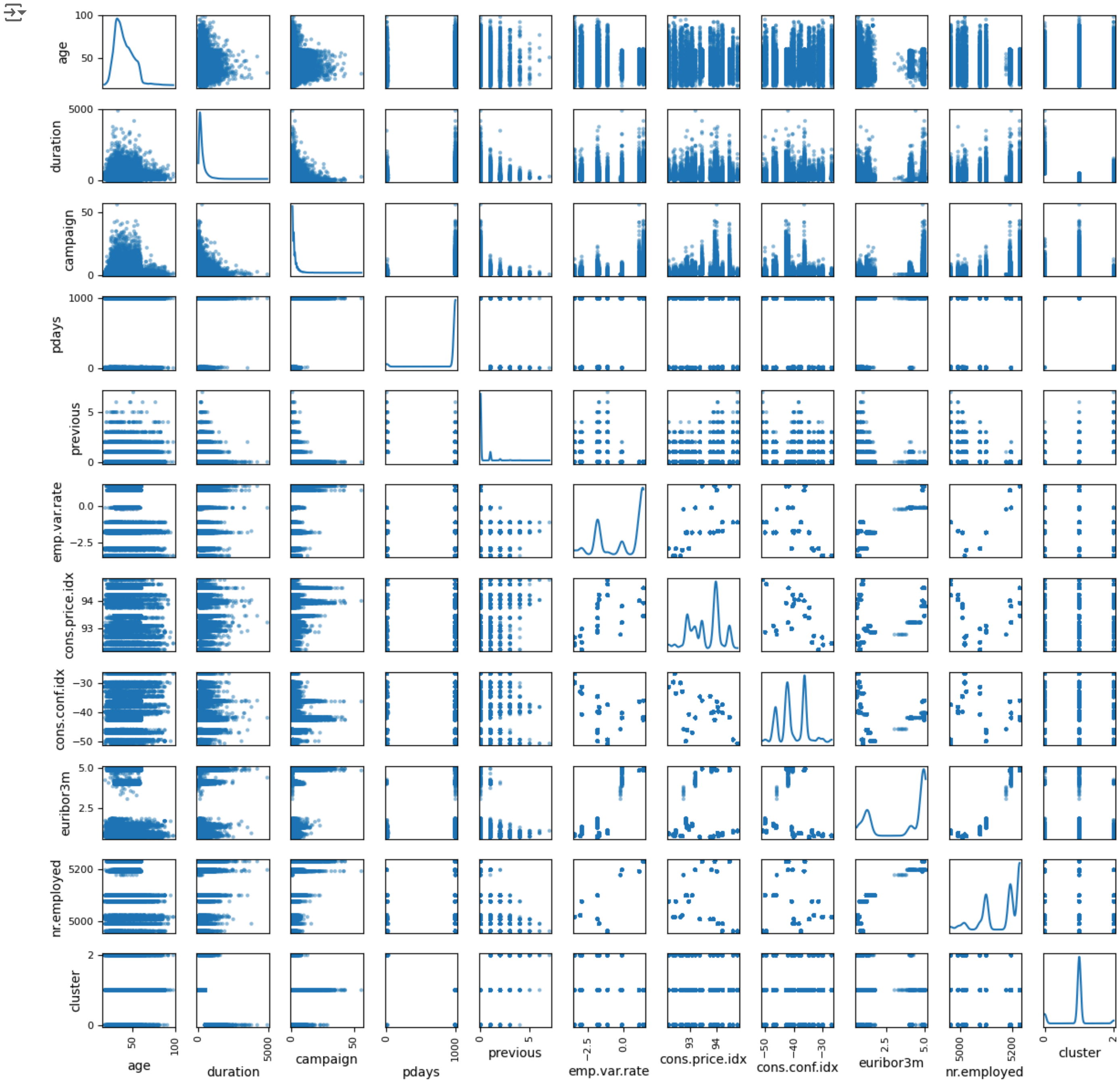
```
df_num = df[Number_cols]
features = df_num

# Perform K-means clustering
k = 3  # Number of clusters
kmeans = KMeans(n_clusters=k, random_state=42)
kmeans.fit(features)

# Get the cluster labels
labels = kmeans.labels_

# Add the cluster labels to the original dataset
df_num['cluster'] = labels

# View the cluster assignments
print(df_num[['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx',
            'cons.conf.idx', 'euribor3m', 'nr.employed', 'cluster']])
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of `n_init` will change fro
  warnings.warn(
       age  duration  campaign  pdays  previous  emp.var.rate  cons.price.idx  \
0       56       261         1    999         0           1.1          93.994
1       57       149         1    999         0           1.1          93.994
2       37       226         1    999         0           1.1          93.994
3       40       151         1    999         0           1.1          93.994
4       56       307         1    999         0           1.1          93.994
...    ...       ...       ...    ...       ...           ...             ...
41183   73       334         1    999         0          -1.1          94.767
41184   46       383         1    999         0          -1.1          94.767
41185   56       189         2    999         0          -1.1          94.767
41186   44       442         1    999         0          -1.1          94.767
41187   74       239         3    999         1          -1.1          94.767

       cons.conf.idx  euribor3m  nr.employed  cluster
0              -36.4      4.857       5191.0        1
1              -36.4      4.857       5191.0        1
2              -36.4      4.857       5191.0        1
3              -36.4      4.857       5191.0        1
4              -36.4      4.857       5191.0        1
...              ...        ...          ...      ...
41183          -50.8      1.028       4963.6        1
41184          -50.8      1.028       4963.6        1
41185          -50.8      1.028       4963.6        1
41186          -50.8      1.028       4963.6        1
41187          -50.8      1.028       4963.6        1

[41188 rows x 11 columns]
<ipython-input-30-03a5ca58dee0>:13: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
  df_num['cluster'] = labels
```

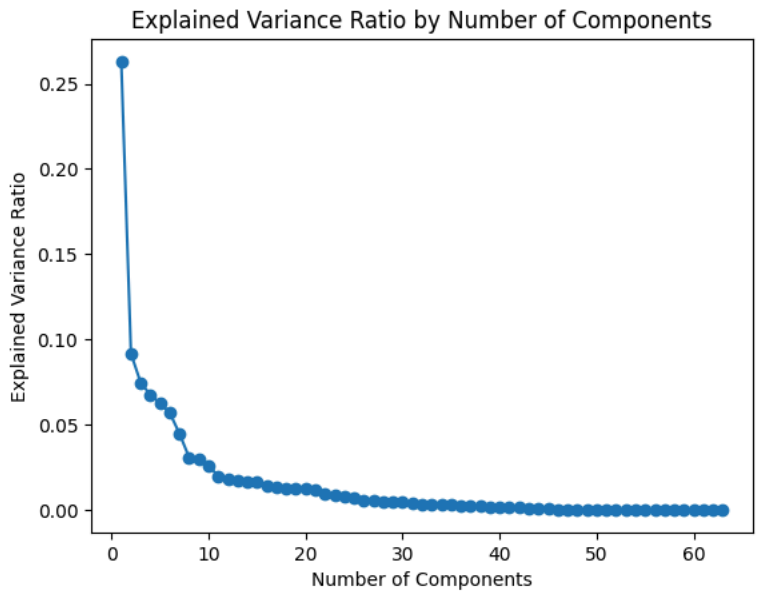
```
# Define cluster labels and colors
cluster_labels = ['Cluster 0', 'Cluster 1', 'Cluster 2']  # Replace with your cluster labels
cluster_colors = ['red', 'blue', 'green']  # Replace with desired colors for each cluster

# Visualize the clusters
for i in range(k):  # k is the number of clusters
    plt.scatter(df_num[df_num['cluster'] == i]['age'], df_num[df_num['cluster'] == i]['duration'], color=cluster_colors[i], label=clust

# Add legends
plt.legend()

# Set plot labels and title
plt.xlabel('Age')
plt.ylabel('Duration')
plt.title('K-means Clustering')

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

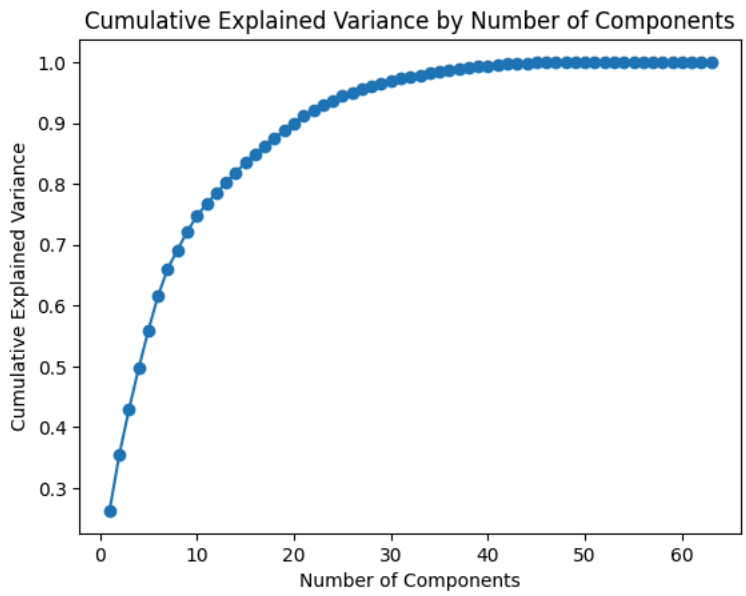



```
# Create a scatter plot matrix
scatter_matrix(df_num, figsize=(12, 12), diagonal='kde')

# Adjust the plot layout
plt.tight_layout()

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

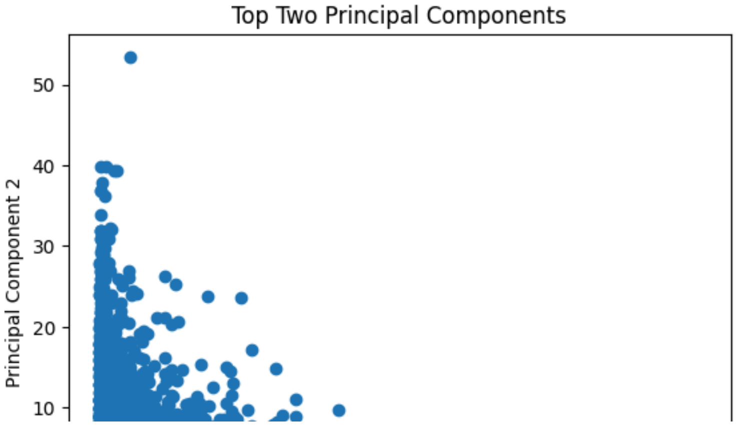

age
duration
campaign
pdays
previous
emp.var.rate
cons.price.idx
cons.conf.idx
euribor3m
nr.employed
cluster

```
pca = PCA()

# Fit PCA on the numeric data
pca.fit(df_pre)

# Get the explained variance ratio
explained_variance_ratio = pca.explained_variance_ratio_

# Calculate the cumulative explained variance
cumulative_variance = np.cumsum(explained_variance_ratio)

# Plot the explained variance ratio
plt.plot(range(1, len(explained_variance_ratio) + 1), explained_variance_ratio, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Explained Variance Ratio by Number of Components')
plt.show()

# Plot the cumulative explained variance
plt.plot(range(1, len(cumulative_variance) + 1), cumulative_variance, marker='o')
plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.title('Cumulative Explained Variance by Number of Components')
plt.show()
```





```
pca = PCA(n_components=6)

# Fit and transform the data using PCA
principal_components = pca.fit_transform(df_num)

# Create a scatter plot of the projected data
plt.scatter(principal_components[:, 0], principal_components[:, 5])
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('Top Two Principal Components')
plt.show()
```




```
# Select numerical columns
numerical_cols = ['age', 'duration', 'campaign', 'pdays', 'previous', 'emp.var.rate', 'cons.price.idx',
                  'cons.conf.idx', 'euribor3m', 'nr.employed']

# Create a subset of the dataframe with numerical columns
df_numerical = df[numerical_cols]

# Create scatterplot matrix
sns.pairplot(df_numerical)
```

```
<seaborn.axisgrid.PairGrid at 0x7f56a8461cc0>
```