In [5]:
```python
import pandas as pd

# file path
file_path = r"C:\Users\gabed\Downloads\test.csv"

# Load the dataset
df = pd.read_csv(file_path)

# Check for missing values in each column
missing_values = df.isnull().sum()
print(missing_values)
```

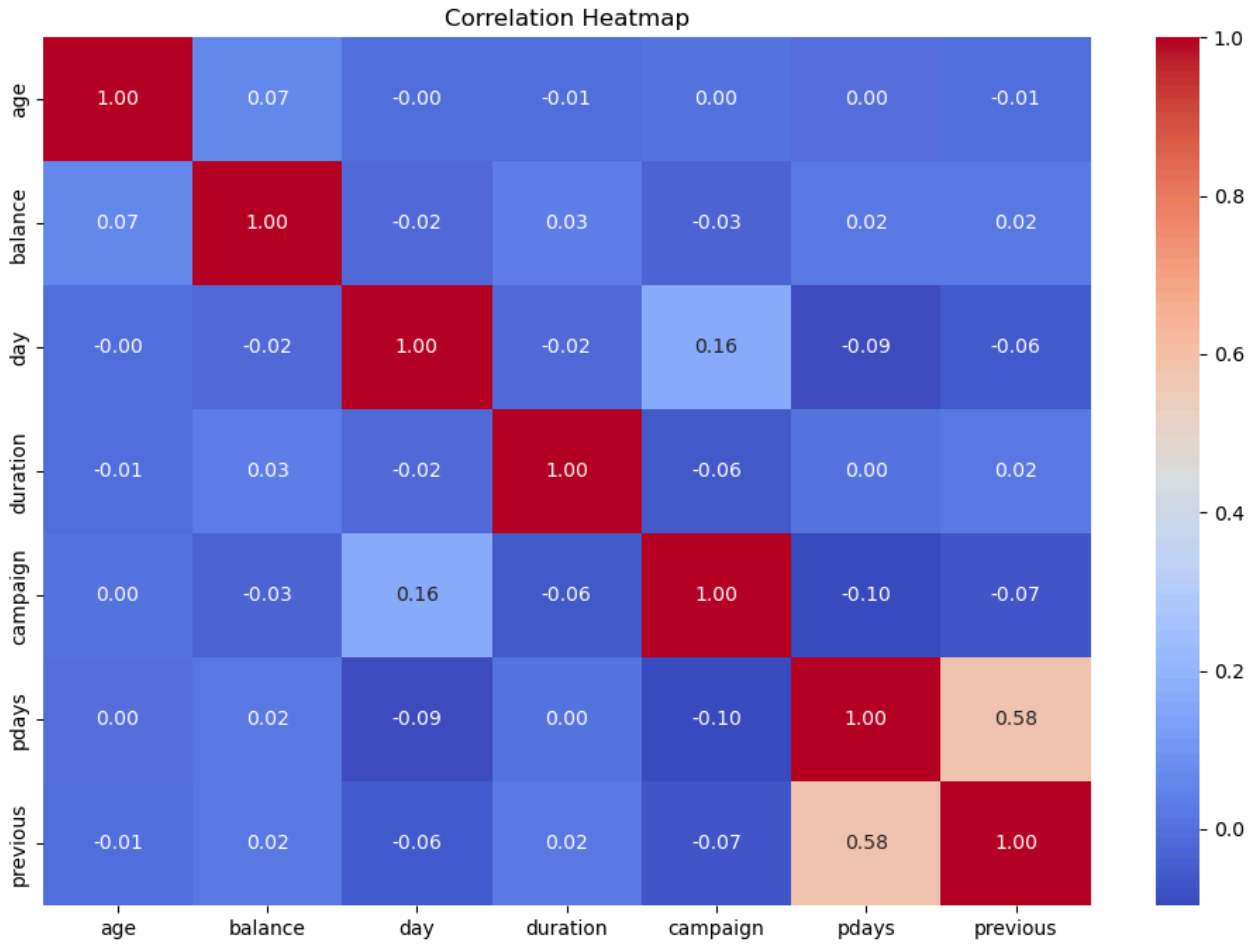
```
age          0
job          0
marital      0
education    0
default      0
balance      0
housing      0
loan         0
contact      0
day          0
month        0
duration     0
campaign     0
pdays        0
previous     0
poutcome     0
y            0
dtype: int64
```

In [8]:
```python
# Detect and remove outliers based on IQR
def remove_outliers(df, column):
    Q1 = df[column].quantile(0.25)
    Q3 = df[column].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    print(f"\nFor '{column}':")
    print(f"Q1 (25th percentile): {Q1}")
    print(f"Q3 (75th percentile): {Q3}")
    print(f"IQR (Interquartile Range): {IQR}")
    print(f"Lower Bound: {lower_bound}")
    print(f"Upper Bound: {upper_bound}")

    # Count how many rows are being removed as outliers
    original_size = df.shape[0]
    df_cleaned = df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
    cleaned_size = df_cleaned.shape[0]

    print(f"Original dataset size: {original_size}")
    print(f"Cleaned dataset size (after removing outliers from '{column}'): {cleaned_size}")
    print(f"Number of outliers removed: {original_size - cleaned_size}\n")

    return df_cleaned

# Apply the function to a column, e.g., 'balance'
df_no_outliers = remove_outliers(df, 'balance')
```

```
For 'balance':
Q1 (25th percentile): 69.0
Q3 (75th percentile): 1480.0
IQR (Interquartile Range): 1411.0
Lower Bound: -2047.5
Upper Bound: 3596.5
Original dataset size: 4521
Cleaned dataset size (after removing outliers from 'balance'): 4015
Number of outliers removed: 506
```

In [9]:
```python
import seaborn as sns
import matplotlib.pyplot as plt

# Compute the correlation matrix
correlation_matrix = df_no_outliers.corr()

# Display the correlation matrix
print("Correlation Matrix:")
print(correlation_matrix)

# Visualize the correlation matrix with a heatmap
plt.figure(figsize=(12, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Heatmap')
plt.show()
```

```
C:\Users\gabed\anaconda3\lib\site-packages\scipy\__init__.py:155: UserWarning: A NumPy version >=1.18.5 and <1.25.0 is required for this version of SciPy (detected version 1.26.4
  warnings.warn(f"A NumPy version >={np_minversion} and <{np_maxversion}"
Correlation Matrix:
                age   balance       day  duration  campaign     pdays  previous
age        1.000000  0.065997 -0.003070 -0.005567  0.003595  0.000424 -0.005573
balance    0.065997  1.000000 -0.015009  0.032559 -0.034587  0.017007  0.020037
day       -0.003070 -0.015009  1.000000 -0.018668  0.161880 -0.089217 -0.057986
duration  -0.005567  0.032559 -0.018668  1.000000 -0.058618  0.004972  0.016623
campaign   0.003595 -0.034587  0.161880 -0.058618  1.000000 -0.096797 -0.067987
pdays      0.000424  0.017007 -0.089217  0.004972 -0.096797  1.000000  0.584273
previous  -0.005573  0.020037 -0.057986  0.016623 -0.067987  0.584273  1.000000
```



In [42]:

```
# Identifying features to drop
threshold = 0.8
correlated_features = set()

# Calculate the correlation matrix
correlation_matrix = df_no_outliers.corr()

# Identify correlated features
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            colname = correlation_matrix.columns[i]
            correlated_features.add(colname)

# Create a copy of the original DataFrame to avoid SettingWithCopyWarning
df_cleaned = df_no_outliers.copy()

# Drop correlated features from the copy
df_cleaned.drop(columns=correlated_features, inplace=True)

# Print the dropped correlated features
print("Dropped correlated features:")
print(correlated_features)

# Summary of the cleaned dataset
print("Cleaned dataset:")
print(df_cleaned.info())
```

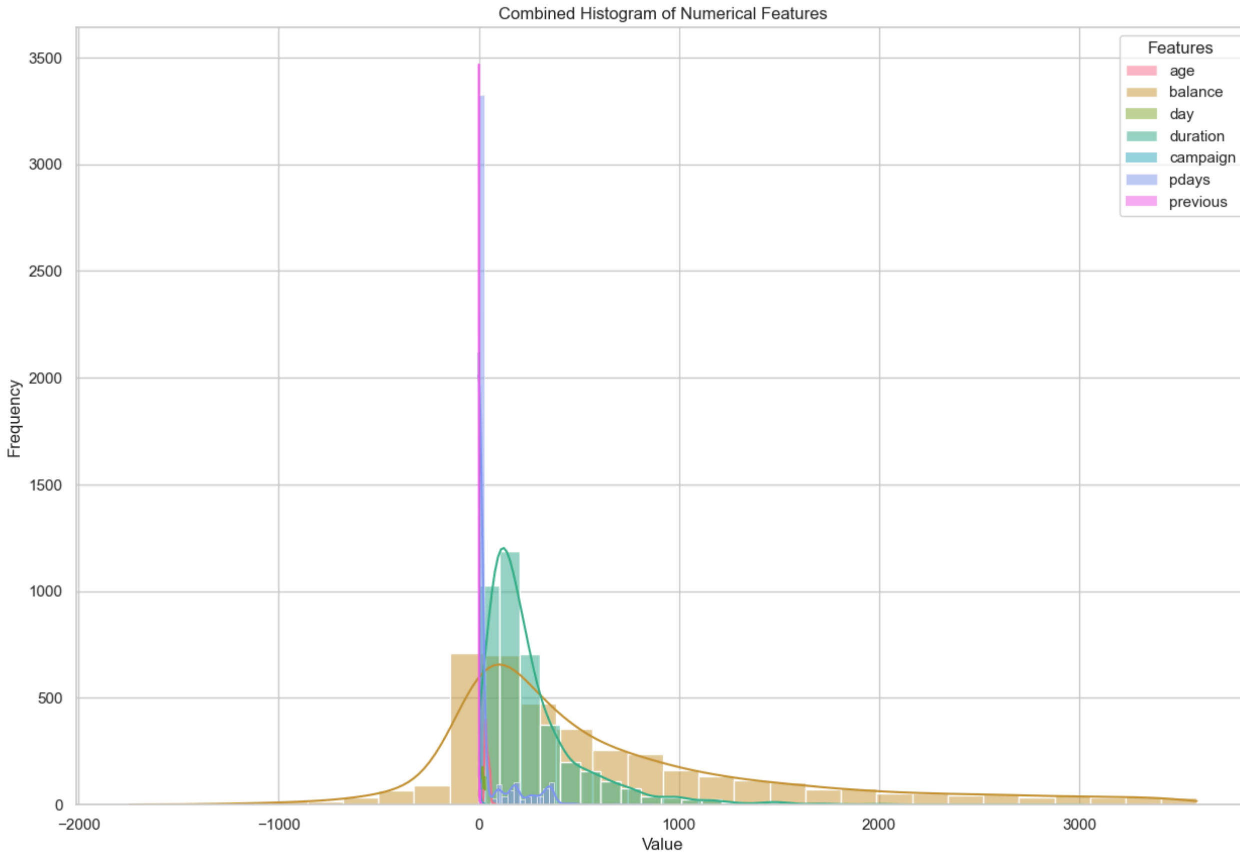
```
Dropped correlated features:
set()
Cleaned dataset:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4015 entries, 0 to 4520
Data columns (total 17 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   age        4015 non-null   int64 
 1   job        4015 non-null   object
 2   marital    4015 non-null   object
 3   education  4015 non-null   object
 4   default    4015 non-null   object
 5   balance    4015 non-null   int64 
 6   housing    4015 non-null   object
 7   loan       4015 non-null   object
 8   contact    4015 non-null   object
 9   day        4015 non-null   int64 
 10  month      4015 non-null   object
 11  duration   4015 non-null   int64 
 12  campaign   4015 non-null   int64 
 13  pdays      4015 non-null   int64 
 14  previous   4015 non-null   int64 
 15  poutcome   4015 non-null   object
 16  y          4015 non-null   object
dtypes: int64(7), object(10)
memory usage: 564.6+ KB
None
```

In [12]:
```python
# Statistical summary of the cleaned dataset
print(df_no_outliers.describe())
```

```
               age      balance         day     duration     campaign  \
count  4015.000000  4015.000000  4015.00000  4015.000000  4015.000000   
mean     40.898630   645.734496    15.86401   264.787298     2.788543   
std      10.387602   869.797049     8.29080   264.609544     3.128038   
min      19.000000 -1746.000000     1.00000     4.000000     1.000000   
25%      33.000000    44.000000     8.00000   103.000000     1.000000   
50%      39.000000   340.000000    16.00000   184.000000     2.000000   
75%      48.000000   978.500000    21.50000   329.000000     3.000000   
max      87.000000  3587.000000    31.00000  3025.000000    50.000000   

             pdays     previous  
count  4015.000000  4015.000000  
mean     39.593275     0.521793  
std     100.717249     1.668149  
min      -1.000000     0.000000  
25%      -1.000000     0.000000  
50%      -1.000000     0.000000  
75%      -1.000000     0.000000  
max     871.000000    25.000000  
```

In [15]:
```python
# Set the aesthetic style of the plots
sns.set(style="whitegrid")

# Define a color palette with distinct colors for each variable
colors = sns.color_palette("husl", len(numerical_columns))  # Using husl palette for distinct col

# Create a combined histogram for all numerical features
plt.figure(figsize=(15, 10))

# Loop through numerical columns and plot their histograms on the same axis
for i, column in enumerate(numerical_columns):
    sns.histplot(df_no_outliers[column], bins=30, kde=True, color=colors[i], label=column, alpha=

# Adding labels and title
plt.title('Combined Histogram of Numerical Features')
```

```python
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.legend(title='Features')
plt.show()
```



In [16]:

```python
from sklearn.preprocessing import MinMaxScaler

# Initialize the MinMaxScaler
min_max_scaler = MinMaxScaler()

# Apply Min-Max scaling to the numerical columns
df_scaled_minmax = df_no_outliers.copy()  # Create a copy of the DataFrame to avoid modifying the
df_scaled_minmax[numerical_columns] = min_max_scaler.fit_transform(df_no_outliers[numerical_colum

# Display the scaled DataFrame
print(df_scaled_minmax.head())
```

```
        age          job  marital  education default   balance housing loan  \
0  0.161765   unemployed  married    primary      no  0.662479      no   no
2  0.235294   management   single   tertiary      no  0.580536     yes   no
3  0.161765   management  married   tertiary      no  0.604163     yes  yes
4  0.588235  blue-collar  married  secondary      no  0.327395     yes   no
5  0.235294   management   single   tertiary      no  0.467467      no   no

    contact       day month  duration  campaign     pdays  previous poutcome  \
0  cellular  0.600000   oct  0.024826  0.000000  0.000000      0.00  unknown
2  cellular  0.500000   apr  0.059914  0.000000  0.379587      0.04  failure
3   unknown  0.066667   jun  0.064548  0.061224  0.000000      0.00  unknown
4   unknown  0.133333   may  0.073486  0.000000  0.000000      0.00  unknown
5  cellular  0.733333   feb  0.045349  0.020408  0.202982      0.12  failure

    y
0  no
2  no
3  no
4  no
5  no
```

In [19]:

```python
# One-Hot Encoding: This technique is used for categorical variables with no ordinal relationship
# It creates binary columns for each category, allowing the model to treat them independently.
df_encoded = pd.get_dummies(df, columns=['marital', 'education', 'contact'], drop_first=True)

# Label Encoding: This technique is used for ordinal categorical variables where the order matter
# It converts each category into a unique integer. For example, 'Low' = 1, 'Medium' = 2, 'High' =
# (Assuming 'education' is an ordinal variable; if needed, adjust the column name and categories.
from sklearn.preprocessing import LabelEncoder

# Example of label encoding (if 'education' was an ordinal variable)
# label_encoder = LabelEncoder()
# df['education'] = label_encoder.fit_transform(df['education'])
```

In [27]:

```python
import numpy as np


# Assuming df_encoded is your DataFrame
# Display original DataFrame information
print("Original DataFrame info:")
print(df_encoded.info())

# Prepare features by dropping the target column
features = df_encoded.drop(columns=['y'], errors='ignore')  # Use errors='ignore' to avoid KeyErr

# Check what columns remain after dropping the target column
print("\nFeatures after dropping target column:")
print(features.head())

# Check which columns are numerical
numeric_features = features.select_dtypes(include=[np.number])

# Print the shape of numeric_features
print("\nNumeric features shape:", numeric_features.shape)

# Check if there are any NaN values and print info
if numeric_features.isnull().values.any():
    print("Numeric features contain NaN values.")
else:
    if numeric_features.shape[1] == 0:
        print("No numeric features available.")
    else:
        print("\nNumeric features are available without NaN values.")

# Optionally, check the original DataFrame structure
print("\nOriginal DataFrame info (again):")
```

```
print(df_encoded.info())

# Check the first few rows to identify issues
print("\nOriginal DataFrame head:")
print(df_encoded.head())
```

```
Original DataFrame info:
<class 'pandas.core.frame.DataFrame'>
Int64Index: 0 entries
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  0 non-null      int64
 1   job                  0 non-null      float64
 2   default              0 non-null      float64
 3   balance              0 non-null      int64
 4   housing              0 non-null      float64
 5   loan                 0 non-null      float64
 6   day                  0 non-null      int64
 7   month                0 non-null      float64
 8   duration             0 non-null      int64
 9   campaign             0 non-null      int64
 10  pdays                0 non-null      int64
 11  previous             0 non-null      int64
 12  poutcome             0 non-null      float64
 13  y                    0 non-null      float64
 14  marital_married      0 non-null      uint8
 15  marital_single       0 non-null      uint8
 16  education_secondary  0 non-null      uint8
 17  education_tertiary   0 non-null      uint8
 18  education_unknown    0 non-null      uint8
 19  contact_telephone    0 non-null      uint8
 20  contact_unknown      0 non-null      uint8
dtypes: float64(7), int64(7), uint8(7)
memory usage: 0.0 bytes
None

Features after dropping target column:
Empty DataFrame
Columns: [age, job, default, balance, housing, loan, day, month, duration, campaign, pdays, previ
ous, poutcome, marital_married, marital_single, education_secondary, education_tertiary, educatio
n_unknown, contact_telephone, contact_unknown]
Index: []

Numeric features shape: (0, 20)

Numeric features are available without NaN values.

Original DataFrame info (again):
<class 'pandas.core.frame.DataFrame'>
Int64Index: 0 entries
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  0 non-null      int64
 1   job                  0 non-null      float64
 2   default              0 non-null      float64
 3   balance              0 non-null      int64
 4   housing              0 non-null      float64
 5   loan                 0 non-null      float64
 6   day                  0 non-null      int64
 7   month                0 non-null      float64
 8   duration             0 non-null      int64
 9   campaign             0 non-null      int64
 10  pdays                0 non-null      int64
 11  previous             0 non-null      int64
 12  poutcome             0 non-null      float64
 13  y                    0 non-null      float64
 14  marital_married      0 non-null      uint8
 15  marital_single       0 non-null      uint8
 16  education_secondary  0 non-null      uint8
 17  education_tertiary   0 non-null      uint8
 18  education_unknown    0 non-null      uint8
```

```
 19  contact_telephone    0 non-null      uint8
 20  contact_unknown      0 non-null      uint8
dtypes: float64(7), int64(7), uint8(7)
memory usage: 0.0 bytes
None

Original DataFrame head:
Empty DataFrame
Columns: [age, job, default, balance, housing, loan, day, month, duration, campaign, pdays, previ
ous, poutcome, y, marital_married, marital_single, education_secondary, education_tertiary, educa
tion_unknown, contact_telephone, contact_unknown]
Index: []

[0 rows x 21 columns]
```

In [32]:
```
from sklearn.model_selection import train_test_split

# Load the dataset
file_path = r"C:\Users\gabed\Downloads\test.csv"
df = pd.read_csv(file_path)

# Assuming 'y' is the target column and all others are features
# Replace 'y' with the actual name of your target variable if it's different
X = df.drop(columns=['y'])  # Feature set
y = df['y']                 # Target variable

# Split into training and test sets (80% training, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Further split the training set into training and validation sets (80% training, 20% validation)
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=4

# Print the shapes of the datasets
print("Training set shape:", X_train.shape)
print("Validation set shape:", X_val.shape)
print("Test set shape:", X_test.shape)
```

```
Training set shape: (2892, 16)
Validation set shape: (724, 16)
Test set shape: (905, 16)
```

In [43]:
```
import warnings
from sklearn.exceptions import ConvergenceWarning
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score

# Suppress the ConvergenceWarning
warnings.filterwarnings("ignore", category=ConvergenceWarning)

# Load the dataset
file_path = r"C:\Users\gabed\Downloads\test.csv"  # Update this path as necessary
df = pd.read_csv(file_path)

# Assume df is your DataFrame containing the entire dataset
X = df.drop(columns='y')  # Features
y = df['y']               # Target variable

# Split into training (60%) and temporary set (40%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)

# Split temporary set into validation (20%) and test (20%)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

```python
# Identify categorical and numerical columns
categorical_cols = X.select_dtypes(include=['object']).columns.tolist()
numerical_cols = X.select_dtypes(include=['int64', 'float64']).columns.tolist()

# Create a column transformer to apply One-Hot Encoding to categorical columns
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_cols),  # One-Hot encode categorical features
        ('num', 'passthrough', numerical_cols)        # Pass through numerical features
    ])

# Create a pipeline that first transforms the data and then applies logistic regression
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(max_iter=2000))  # Increased max_iter for better convergenc
])

# Fit the model using the training set
model.fit(X_train, y_train)

# Predict on the validation set
y_val_pred = model.predict(X_val)

# Evaluate the model on the validation set
print("Validation Accuracy:", accuracy_score(y_val, y_val_pred))
print(classification_report(y_val, y_val_pred))

# Predict on the test set
y_test_pred = model.predict(X_test)

# Evaluate the model on the test set
print("Test Accuracy:", accuracy_score(y_test, y_test_pred))
print(classification_report(y_test, y_test_pred))
```

```
Validation Accuracy: 0.9037610619469026
              precision    recall  f1-score   support

          no       0.92      0.97      0.95       806
         yes       0.60      0.33      0.42        98

    accuracy                           0.90       904
   macro avg       0.76      0.65      0.69       904
weighted avg       0.89      0.90      0.89       904

Test Accuracy: 0.9093922651933701
              precision    recall  f1-score   support

          no       0.93      0.97      0.95       814
         yes       0.58      0.34      0.43        91

    accuracy                           0.91       905
   macro avg       0.76      0.66      0.69       905
weighted avg       0.89      0.91      0.90       905
```