Experiment – 3

Handling Missing Data: Explore different strategies for dealing with missing data, such as imputation techniques (mean, median, mode), or advanced methods like k-nearest neighbors imputation.

Date Set: *We will use the data in a csv file, namely* ***data.csv****, which is the data set on which we will implement procedure to handle missing data. This data set belongs to a retail company that collected some data from their customers, whether or not they purchased a certain product. Here, each of the rows correspond to different customers. For each of these customers the company gathered their country, their age, salary, and whether or not they purchased their product.*

Objective:

Explore strategies for handling missing data in a dataset.

Tasks:

1. Import the libraries.
2. Import the data set.
3. Identify missing values in the dataset.
4. Implement different imputation techniques (mean, median, mode).
5. Experiment with advanced imputation methods like k-nearest neighbors imputation.
6. Evaluate the impact of each imputation method on the dataset.

Python Program:

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Note:

A. **import numpy as np**:
   a. **numpy** is a powerful library for numerical computing in Python.
   b. **import numpy as np** imports the **numpy** library and gives it an alias **np** for convenience.
   c. The **np** alias is commonly used to reference functions and objects from the **numpy** library. For example, **np.array()** creates a NumPy array.

B. **import matplotlib.pyplot as plt**:
   a. **matplotlib** is a popular library for creating static, animated, and interactive visualizations in Python.
   b. **import matplotlib.pyplot as plt** imports the **pyplot** module from **matplotlib** and gives it an alias **plt** for convenience.

c. The **plt** alias is commonly used to create plots and charts. For example, **plt.plot()** is used to create line plots.

C. **import pandas as pd**:
   a. **pandas** is a data manipulation and analysis library for Python.
   b. **import pandas as pd** imports the **pandas** library and gives it an alias **pd** for convenience.
   c. The **pd** alias is commonly used to reference functions and objects from the **pandas** library. For example, **pd.DataFrame()** is used to create a DataFrame.

Importing the dataset

```
dataset = pd.read_csv(r'C:\Users\...\GHRCEM\Dataset\Data.csv')
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values
```

Note:
Here we have created two new entities: X is the matrix of features (also called independent variable), and Y is the dependent variable vector.

A. **X = dataset.iloc[:, :-1].values**:
   a. **dataset**: This presumably represents a pandas DataFrame containing your dataset.
   b. **iloc (i. e. integer-location based indexing)**: This is a method in pandas used for integer-location based indexing. It allows you to select data by row and column index.
   c. **[:, :-1]**: This selects all rows (**:**) and all columns except the last one (**:-1**). In other words, it selects all features (independent variables) of the dataset. Remember -1 is the index of the last column of pandas DataFrame.
   d. **.values**: This converts the selected data (features) into a NumPy array. Machine learning algorithms often work with NumPy arrays instead of pandas DataFrames.

   So, **X** would contain the feature values of your dataset.

B. **Y = dataset.iloc[:, -1].values**:
   a. **[:, -1]**: It selects all rows (**:**) and only the last column (**-1**). This is typically the target variable (dependent variable) in a machine learning problem.
   b. **.values**: Converts the selected column into a NumPy array.

   So, **Y** would contain the target variable values.

C. The **r** prefix before the string `(r'C:\Users\...\GHRCEM\Dataset\Data.csv')` makes it a raw string, so backslashes are treated as literal characters and won't cause the Unicode escape error.

```
print(dataset.values)
# prints the entire dataset
```

Output:

```
[['France' 44.0 72000.0 'No']
 ['Spain' 27.0 48000.0 'Yes']
 ['Germany' 30.0 54000.0 'No']
 ['Spain' 38.0 61000.0 'No']
 ['Germany' 40.0 nan 'Yes']
 ['France' 35.0 58000.0 'Yes']
```

```
 ['Spain' nan 52000.0 'No']
 ['France' 48.0 79000.0 'Yes']
 ['Germany' 50.0 83000.0 'No']
 ['France' 37.0 67000.0 'Yes']]
```

```
print(X)
```

Output:

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 nan]
 ['France' 35.0 58000.0]
 ['Spain' nan 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```

```
print(Y)
```

Output:

```
['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes']
```

Taking care of missing data

```
from sklearn.impute import SimpleImputer as si
imputer = si(missing_values = np.nan, strategy = 'mean')
imputer.fit(X[:, 1:3])
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

Note:
This code uses scikit-learn's **SimpleImputer** to fill in missing values in specific columns of the dataset with the mean value of each respective column. This is a common technique in data preprocessing when dealing with missing data before applying machine learning algorithms. Let's break down each line:

A. **from sklearn.impute import SimpleImputer as si**:
   a. Imports the **SimpleImputer** class from scikit-learn's **impute** module.
   b. The **as si** part gives the **SimpleImputer** class an alias **si** for convenience, making it shorter to reference in the code.

B. **imputer = si(missing_values = np.nan, strategy = 'mean')**:
   a. Creates an instance of the **SimpleImputer** class with specific parameters.
   b. **missing_values = np.nan**: Specifies that missing values in the dataset are represented by **np.nan** (NumPy's representation of missing or undefined values).
   c. **strategy = 'mean'**: Specifies the imputation strategy. In this case, it uses the mean value of the non-missing values in each column to fill in the missing values.

C. **imputer.fit(X[:, 1:3])**:
   a. Fits the imputer on the specified subset of the dataset.
   b. **X[:, 1:3]** selects all rows (**:**) and columns 1 to 2 (index 1 is the second column, and index 3 is exclusive), which typically means it's selecting specific features with missing values for imputation.

c. The **fit** method computes the mean value for each selected column.

D. **X[:, 1:3] = imputer.transform(X[:, 1:3])**:

a. Transforms the selected subset of the dataset by replacing missing values with the computed means.

b. **transform** method applies the imputation strategy to fill in missing values in the specified columns.

c. The updated values are then assigned back to the original dataset.

```
print(X)
```

Output:

```
[['France' 44.0 72000.0]
 ['Spain' 27.0 48000.0]
 ['Germany' 30.0 54000.0]
 ['Spain' 38.0 61000.0]
 ['Germany' 40.0 63777.77777777778]
 ['France' 35.0 58000.0]
 ['Spain' 38.77777777777778 52000.0]
 ['France' 48.0 79000.0]
 ['Germany' 50.0 83000.0]
 ['France' 37.0 67000.0]]
```