Experiment – 4

Data Cleaning: Identify and clean inconsistencies, errors, and outliers in the dataset. Use techniques like outlier detection and removal, or data normalization to improve data quality.

Date Set: *The dataset consists of individuals with various attributes, and the goal may be to predict a binary target variable. Here's a breakdown of the columns:*

- ***id:*** *Unique identifier for each individual.*
- ***age:*** *Age of the individual.*
- ***income:*** *Income of the individual.*
- ***target:*** *Binary target variable, where 1 typically represents a positive outcome or belonging to a certain category, and 0 represents a negative outcome.*

Objective:

Identify and clean inconsistencies, errors, and outliers in the dataset.

Tasks:

1. Identify and visualize outliers in the dataset.
2. Choose an appropriate method to handle outliers (e.g., removal or transformation).
3. Implement data cleaning techniques to address inconsistencies and errors.
4. Assess the impact of data cleaning on the dataset.

Data cleaning, also known as data cleansing or data scrubbing, is the process of identifying and correcting errors, inconsistencies, and inaccuracies in datasets. The goal of data cleaning is to improve the quality of the data by removing or correcting any errors or inconsistencies that might affect its accuracy and reliability. This process is a crucial step in the overall data preparation and analysis pipeline.

Data can be collected from various sources, and during the collection and integration process, errors may be introduced. Common issues include missing values, duplicate records, inaccuracies, outliers, and inconsistencies in formatting or coding. Data cleaning aims to address these issues to ensure that the data is accurate, reliable, and suitable for analysis.

Key steps in the data cleaning process may include:

1. **Handling Missing Data:** Identifying and addressing missing values by either removing them, imputing values, or using other appropriate techniques.
2. **Removing Duplicates:** Identifying and eliminating duplicate records to avoid redundancy and ensure data integrity.
3. **Correcting Inconsistencies:** Addressing inconsistencies in data formats, units of measurement, and coding schemes to maintain uniformity.
4. **Handling Outliers:** Identifying and dealing with outliers that may skew the analysis or results.
5. **Standardizing Data:** Ensuring that data follows a consistent format and standardizing variables to facilitate analysis.
6. **Validating Data:** Verifying data against predefined rules or criteria to identify and correct any discrepancies.
7. **Handling Typos and Errors:** Correcting typographical errors, misspellings, and other data entry mistakes.

Python Program:

Importing the libraries

```
import pandas as pd
```

A. **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

```
# Load the dataset, Replace with the actual file path
df = pd.read_csv(r'C:\Users\KUNAL KUNDU\...\experiment_4.csv')

# Display the initial dataset summary
print("Initial dataset summary:")
print(df.describe())
```

Note:

- **print(df.describe())**: The **describe()** function generates descriptive statistics of the DataFrame, including measures of central tendency, dispersion, and shape of the distribution. This provides a quick overview of the dataset, including the count, mean, std (standard deviation), min, 25%, 50%, 75%, and max values for each column.
- The values 25%, 50%, and 75% represent the percentiles of the data distribution. Here's what each of these percentiles means:
  - 25% (Q1 - First Quartile): This value represents the data point below which 25% of the data falls. It is also known as the first quartile.
  - 50% (Q2 - Second Quartile/Median): This is the median or the middle value of the dataset. It separates the lower 50% from the upper 50%.
  - 75% (Q3 - Third Quartile): This value represents the data point below which 75% of the data falls. It is also known as the third quartile.

Output:

```
Initial dataset summary:
              id        age         income    target
count  40.000000  40.000000      39.000000  40.00000
mean   20.500000  33.775000   82256.410256   0.85000
std    11.690452   6.232617   20615.168086   0.36162
min     1.000000  22.000000   45000.000000   0.00000
25%    10.750000  29.000000   70000.000000   1.00000
50%    20.500000  34.000000   82000.000000   1.00000
75%    30.250000  38.250000   93500.000000   1.00000
max    40.000000  45.000000  150000.000000   1.00000
```

Taking care of missing data

```
# Identify and handle missing values
df.dropna(inplace=True)

# Display the dataset summary after handling missing values
print("\nDataset summary after handling missing values:")
print(df.describe())
```

Note:

Here, we use the **dropna()** function to remove rows with missing values. Handling missing values is crucial for maintaining the quality of the dataset.

Output:

```
Dataset summary after handling missing values:
              id        age         income     target
count  39.000000  39.000000      39.000000  39.000000
mean   20.923077  33.615385   82256.410256   0.846154
std    11.528876   6.230719   20615.168086   0.365518
min     1.000000  22.000000   45000.000000   0.000000
25%    11.500000  29.000000   70000.000000   1.000000
50%    21.000000  34.000000   82000.000000   1.000000
75%    30.500000  37.500000   93500.000000   1.000000
max    40.000000  45.000000  150000.000000   1.000000
```

Outlier Detection using Isolation Forest

```
from sklearn.ensemble import IsolationForest

# Create an Isolation Forest model
outlier_detector = IsolationForest(contamination=0.05)

# Fit the model and identify outliers
df['is_outlier']                                                      =
outlier_detector.fit_predict(df.drop(columns=['target']))

# Display the count of outliers
print("\nNumber of identified outliers:", df[df['is_outlier']
== -1].shape[0])
```

Note:

1. **Importing IsolationForest:**
   from sklearn.ensemble import IsolationForest: This line imports the **IsolationForest** class from scikit-learn. **IsolationForest** is a machine learning algorithm used for outlier detection.
2. **Creating an Isolation Forest model:**
   outlier_detector = IsolationForest(contamination=0.05): This line creates an instance of the **IsolationForest** class. The **contamination** parameter is set to 0.05, which means that approximately 5% of the data is expected to be outliers. You can adjust this parameter based on your specific use case.

3. **Fitting the model and identifying outliers:**
   df['is_outlier'] = outlier_detector.fit_predict(df.drop(columns=['target'])): This line fits the Isolation Forest model to the dataset, excluding the 'target' column. The **fit_predict** method returns a binary array where -1 indicates an outlier, and 1 indicates an inlier. The result is stored in a new column called 'is_outlier' in the DataFrame.
4. **Displaying the count of outliers:**
   print("\nNumber of identified outliers:", df[df['is_outlier'] == -1].shape[0]): This line prints the count of identified outliers in the dataset. It does this by filtering the DataFrame to include only rows where 'is_outlier' is -1 (indicating an outlier) and then checking the number of rows using **shape[0]**. **shape[0]**: This part retrieves the number of rows in the filtered DataFrame, which is the count of identified outliers. The **shape** attribute returns a tuple representing the dimensions of the DataFrame, and **[0]** retrieves the number of rows.

Output:
```
Number of identified outliers: 2
```

Remove Outliers

```
# Remove outliers
df_cleaned          =          df[df['is_outlier']          ==
1].drop(columns=['is_outlier'])

# Display the dataset summary after removing outliers
print("\nDataset summary after removing outliers:")
print(df_cleaned.describe())
```

Note:
1. **Filtering Outliers:**
   - **df['is_outlier'] == 1**: This creates a boolean mask where **True** corresponds to rows identified as inliers (not outliers) by the Isolation Forest model.
   - **df[df['is_outlier'] == 1]**: This applies the boolean mask to the DataFrame, keeping only the rows where 'is_outlier' is equal to 1. This effectively removes the rows identified as outliers.
2. **Dropping the 'is_outlier' Column:**
   - **.drop(columns=['is_outlier'])**: This part removes the 'is_outlier' column from the DataFrame, as it's no longer needed after identifying and removing outliers.
3. **Assigning to df_cleaned:**
   - **df_cleaned = ...**: The result of the above filtering and column dropping operations is assigned to the variable **df_cleaned**. This new DataFrame contains only the rows that were not identified as outliers.

Output:
```
Dataset summary after removing outliers:
              id        age         income     target
count  37.000000  37.000000      37.000000  37.000000
mean   21.783784  33.000000   81027.027027   0.864865
std    11.190609   5.778312   17487.275511   0.346583
```

```
min      1.000000  22.000000   45000.000000   0.000000
25%     13.000000  29.000000   70000.000000   1.000000
50%     22.000000  33.000000   82000.000000   1.000000
75%     31.000000  37.000000   92000.000000   1.000000
max     40.000000  44.000000  120000.000000   1.000000
```

Data Normalization (Min-Max Normalization)

```
# Data normalization (Min-Max normalization)
df_normalized = (df_cleaned - df_cleaned.min()) / (df_cleaned.max() - df_cleaned.min())

# Display the normalized dataset summary
print("\nNormalized dataset summary:")
print(df_normalized.describe())
```

Note:

1. **Normalization:** Normalization is a data preprocessing technique used to scale the values of different features to a standard range, making them more comparable. It is particularly useful when working with machine learning algorithms that are sensitive to the scale of input features, such as gradient-based optimization algorithms.

   The most common type of normalization is Min-Max normalization, which scales the values of a feature to a range between 0 and 1. The formula for Min-Max normalization is:
   Normalized Value = (Original Value−Min Value) / (Max Value−Min Value)
2. **Min-Max Normalization:**
   - **(df_cleaned - df_cleaned.min()) / (df_cleaned.max() - df_cleaned.min())**: This expression performs Min-Max normalization on the cleaned dataset (**df_cleaned**). This scales the values in each column between 0 and 1.
3. **Assigning to df_normalized:**
   - **df_normalized = ...**: The result of the normalization operation is assigned to the variable **df_normalized**. This new DataFrame contains the normalized values.

Output:

```
Normalized dataset summary:
              id        age     income     target
count  37.000000  37.000000  37.000000  37.000000
mean    0.532918   0.500000   0.480360   0.864865
std     0.286939   0.262651   0.233164   0.346583
min     0.000000   0.000000   0.000000   0.000000
25%     0.307692   0.318182   0.333333   1.000000
50%     0.538462   0.500000   0.493333   1.000000
75%     0.769231   0.681818   0.626667   1.000000
max     1.000000   1.000000   1.000000   1.000000
```

Save the Cleaned and Normalized Dataset

```
# Save the cleaned and normalized dataset to a new file
```

```
df_normalized.to_csv('cleaned_normalized_dataset.csv',
index=False)
print("\nCleaned and normalized dataset saved to
'cleaned_normalized_dataset.csv'")
```

Note:

df_normalized.to_csv('cleaned_normalized_dataset.csv', index=False):

- **df_normalized**: This is the DataFrame containing the cleaned and normalized data.
- **.to_csv('cleaned_normalized_dataset.csv', index=False)**: This method is used to export the DataFrame to a CSV (Comma-Separated Values) file named 'cleaned_normalized_dataset.csv'. The **index=False** parameter ensures that the index column is not included in the CSV file.

Output:

```
Cleaned and normalized dataset saved to 'cleaned_normalized_da
taset.csv'
```