52 Suhail Shaikh D17C

ADS Experiment -2

Aim: Apply data cleaning techniques (e.g. Data Imputation).

Theory:

Data Cleaning

Data cleaning, also known as data cleansing or data scrubbing, is the process of identifying and correcting or removing errors, inaccuracies, and inconsistencies in datasets. This process is crucial in maintaining the quality and reliability of data, which is especially important in data analysis and machine learning where the output quality is determined by the input data quality.

Data cleaning involves various techniques and methods, including:

- **Data Auditing**: Involves statistical analysis of the data to identify outliers or anomalies in the data.
- **Data Discrepancy Detection**: Identifies data inconsistencies using predefined rules and algorithms.
- **Data Transformation**: Converts data from one format or structure into another.
- **Data Validation**: Uses a set of validation rules to accept or reject data.

Data Imputation

Data imputation is a key technique in data cleaning that deals with missing data in the dataset. Missing data can occur due to various reasons such as errors in data collection, failures in measurement systems, or data privacy protocols. Ignoring missing data or deleting the rows with missing data can lead to biased or incorrect results.

Various data imputation methods are used to estimate the missing values based on other data in the dataset, including:

- **Mean/Median/Mode Imputation**: Replaces missing data with the mean, median, or mode of the available data. This method is simple to implement but can lead to an underestimation of variances or correlations when the amount of missing data is significant.
- **Fixed Value Imputation**: Replaces missing data with a fixed value. The fixed value, which could be a standard default value like zero, is determined based on the context and nature of the data.
- **Last Observation Carried Forward/Next Observation Carried Backward (LOCF/NOCB) Imputation**: Replaces missing data with the last observed value before

the missing data point or the next observed value after the missing data point. This method is often used in time series data.

- **Linear Interpolation**: Estimates missing values in a sequence of numbers by drawing a line between two adjacent points and picking the value at the position of the missing data.
- **Regression Imputation**: Uses a regression model to predict missing values based on other variables.
- **K-Nearest Neighbors (KNN) Imputation**: Replaces missing data with the mean or median of the k-nearest non-missing values. The “closeness” of values is typically determined using a distance metric such as Euclidean distance.
- **Multiple Imputation**: Generates multiple imputations for missing data, creating several different complete datasets. Each of these datasets is then analyzed separately, and the results are pooled to create a single estimate and confidence interval.

Each of these methods has its strengths and weaknesses, and the choice of method depends on the nature of the data and the specific requirements of the analysis.

Conclusion:
In summary, both data cleaning and data imputation are critical components of the data preprocessing pipeline, aiming to ensure the reliability, completeness, and accuracy of datasets used for analyses and modeling. The theories underlying these processes emphasize the systematic identification and correction of errors, as well as the thoughtful handling of missing values to maintain the integrity of the data

```
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer
```

```
df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ADS2/food_coded.csv")
```

```
df.head()
```

	GPA	Gender	breakfast	calories_chicken	calories_day	calories_scone	coffee
0	2.4	2	1	430	NaN	315.0	1
1	3.654	1	1	610	3.0	420.0	2
2	3.3	1	1	720	4.0	420.0	2
3	3.2	1	1	430	3.0	420.0	2
4	3.5	1	1	720	2.0	420.0	2

5 rows × 61 columns

```
df.columns
```

```
Index(['GPA', 'Gender', 'breakfast', 'calories_chicken', 'calories_day',
       'calories_scone', 'coffee', 'comfort_food', 'comfort_food_reasons',
       'comfort_food_reasons_coded', 'cook', 'comfort_food_reasons_coded.1',
       'cuisine', 'diet_current', 'diet_current_coded', 'drink',
       'eating_changes', 'eating_changes_coded', 'eating_changes_coded1',
       'eating_out', 'employment', 'ethnic_food', 'exercise',
       'father_education', 'father_profession', 'fav_cuisine',
       'fav_cuisine_coded', 'fav_food', 'food_childhood', 'fries', 'fruit_day',
       'grade_level', 'greek_food', 'healthy_feeling', 'healthy_meal',
       'ideal_diet', 'ideal_diet_coded', 'income', 'indian_food',
       'italian_food', 'life_rewarding', 'marital_status',
       'meals_dinner_friend', 'mother_education', 'mother_profession',
       'nutritional_check', 'on_off_campus', 'parents_cook', 'pay_meal_out',
       'persian_food', 'self_perception_weight', 'soup', 'sports', 'thai_food',
       'tortilla_calories', 'turkey_calories', 'type_sports', 'veggies_day',
       'vitamins', 'waffle_calories', 'weight'],
      dtype='object')
```

Step 1: Removing All Irrelevant Observation

```
df=df.drop(['father_education',"father_profession","marital_status","mother_education","comfort_food_reasons",
            "comfort_food_reasons_coded","comfort_food_reasons_coded.1",
            'diet_current_coded', 'eating_changes_coded', 'eating_changes_coded1','eating_changes_coded',
            'eating_changes_coded1','employment','fav_cuisine','fav_cuisine_coded', 'fav_food',
            'food_childhood', 'on_off_campus', 'parents_cook', 'pay_meal_out','self_perception_weight'], axis=1)
```

```
col = df.columns
print(col)
print(len(col))
```

```
Index(['GPA', 'Gender', 'breakfast', 'calories_chicken', 'calories_day',
       'calories_scone', 'coffee', 'comfort_food', 'cook', 'cuisine',
       'diet_current', 'drink', 'eating_changes', 'eating_out', 'ethnic_food',
       'exercise', 'fries', 'fruit_day', 'grade_level', 'greek_food',
       'healthy_feeling', 'healthy_meal', 'ideal_diet', 'ideal_diet_coded',
```

```
       'income', 'indian_food', 'italian_food', 'life_rewarding',
       'meals_dinner_friend', 'mother_profession', 'nutritional_check',
       'persian_food', 'soup', 'sports', 'thai_food', 'tortilla_calories',
       'turkey_calories', 'type_sports', 'veggies_day', 'vitamins',
       'waffle_calories', 'weight'],
      dtype='object')
42
```

step 2: Fixing Structural Error by keeping only numbers and float

```
number_col = []  # Initialize an empty list
for x in col:
    val = df[x][0]
    try:
        if isinstance(val, np.int64) or isinstance(val, np.float64):
            number_col.append(x)  # Corrected: use .append(x) to add to the list
    except:
        continue
print(number_col)
```

```
['Gender', 'breakfast', 'calories_chicken', 'calories_day', 'calories_scone', 'coffee', 'cook', 'cuisine', 'drink'
```

```
for x in col:
  if x not in number_col:
    df=df.drop(x,axis=1)
```

```
df.columns
len(df.columns)
```

```
32
```

Step 3: Calculating number of missing values in each column .

+ Code + Text

```
null_values=df.isnull().sum()
print(null_values)
```

```
Gender                0
breakfast             0
calories_chicken      0
calories_day         19
calories_scone        1
coffee                0
cook                  3
cuisine              17
drink                 2
eating_out            0
ethnic_food           0
exercise             13
fries                 0
fruit_day             0
grade_level           0
greek_food            0
healthy_feeling       0
ideal_diet_coded      0
income                1
indian_food           0
italian_food          0
life_rewarding        1
nutritional_check     0
persian_food          1
soup                  1
sports                2
thai_food             0
tortilla_calories     1
turkey_calories       0
veggies_day           0
vitamins              0
waffle_calories       0
dtype: int64
```

```
col=df.columns
col_to_clean=[]
for x in col:
  if null_values[x] > 0:
    col_to_clean.append(x)

print(col_to_clean)
```

```
['calories_day', 'calories_scone', 'cook', 'cuisine', 'drink', 'exercise', 'income', 'life_rewarding', 'persian_fo
```

Step 4: Handle missing data(Data Imputation)

a) Mean and Median

```
# For Mean
df_mean=df.copy()
for x in col_to_clean:
  mean_value=df[x].mean()
  df_mean[x].fillna(mean_value,inplace=True)

# For Median
df_median = df.copy()
for x in col_to_clean:
    median_value = df[x].median()
    df_median[x].fillna(median_value, inplace=True)
```

b) Minimum and Maximum Value

```
# For Maximum Value
df_max = df.copy()
for x in col_to_clean:
    max_value = df[x].max()
    df_max[x].fillna(max_value, inplace=True)

# For Minimum Value
df_min = df.copy()
for x in col_to_clean:
    min_value = df[x].min()
    df_min[x].fillna(min_value, inplace=True)
```

c) Most Frequent Value

```
# For Most Frequent Value
df_mode = df.copy()
for x in col_to_clean:
    mode_value = df[x].mode()[0]
    df_mode[x].fillna(mode_value, inplace=True)
```

d) Fixed Value

```
# For Fixed Value, let's say the fixed value is 0
df_fixed = df.copy()
for x in col_to_clean:
    df_fixed[x].fillna(0, inplace=True)
```

e) Next Value

```
# For Next Value
df_next = df.copy()
for x in col_to_clean:
    df_next[x].fillna(method='bfill', inplace=True)
```

f) K Nearest Neighbour

```
# Create the imputer
knn_imputer = KNNImputer(n_neighbors=3)
# Apply the imputer to the DataFrame
df_knn = df.copy()
df_knn[col_to_clean] = knn_imputer.fit_transform(df[col_to_clean])
```

g) Missing Value Prediction

```
df_mvp = df.copy()
for col in col_to_clean:
    # Split the data into training and test sets
    train = df_mvp[df_mvp[col].notna()]
    test = df_mvp[df_mvp[col].isna()]
    # Check if there are missing values to impute
    if test.shape[0] == 0:
        continue
    # Define the target and features
    y_train = train[col]
    X_train = train.drop(col, axis=1)
    X_test = test.drop(col, axis=1)
    # Create an imputer for filling missing values in the features
    imputer = SimpleImputer(strategy='mean')
    # Fill missing values in the features
    X_train = imputer.fit_transform(X_train)
    X_test = imputer.transform(X_test)
    # Train the model
    model = LinearRegression()
    model.fit(X_train, y_train)
    # Predict the missing values
    df_mvp.loc[df_mvp[col].isna(), col] = model.predict(X_test)
```

Creating Function to check if there is any null values in these data frames. It would return true if there is null values else false.

```
def check_null(df):
    return df.isnull().any().any()
```

```
data_frames=[df_mean,df_median,df_max,df_min,df_fixed,df_knn,df_mvp]
result=[]
for d in data_frames:
  result.append(check_null(d))
result
```

```
[False, False, False, False, False, False, False]
```

Since we are getting false for every data frame . Hence, Data imputation is done successfully.