EC-9560 DATA MINING

LAB 03

ATPUTHARAVI.R

2020/E/015

30 OCTOBER 2024

1. Load the combined dataset (train & test)

```
# Load the train and test datasets
df_train = pd.read_csv('train_dataset.csv')
df_test = pd.read_csv('test_dataset.csv')

# Add a column to indicate the source of the data
df_train['dataset'] = 'train'
df_test['dataset'] = 'test'

# Combine the two DataFrames
df_combined = pd.concat([df_train, df_test], ignore_index=True)

# Save the combined dataset to a new CSV file
df_combined.to_csv('combined_dataset.csv', index=False)

# Load the combined dataset
df = pd.read_csv('combined_dataset.csv')

# Display basic info
df.info()
df.head()
```

Output of this code:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 55692 entries, 0 to 55691
Data columns (total 24 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   age                  55692 non-null  int64
 1   height(cm)           55692 non-null  int64
 2   weight(kg)           55692 non-null  int64
 3   waist(cm)            55692 non-null  float64
 4   eyesight(left)       55692 non-null  float64
 5   eyesight(right)      55692 non-null  float64
 6   hearing(left)        55692 non-null  int64
 7   hearing(right)       55692 non-null  int64
 8   systolic             55692 non-null  int64
 9   relaxation           55692 non-null  int64
 10  fasting blood sugar  55692 non-null  int64
 11  Cholesterol          55692 non-null  int64
 12  triglyceride         55692 non-null  int64
 13  HDL                  55692 non-null  int64
 14  LDL                  55692 non-null  int64
 15  hemoglobin           55692 non-null  float64
 16  Urine protein        55692 non-null  int64
 17  serum creatinine     55692 non-null  float64
 18  AST                  55692 non-null  int64
 19  ALT                  55692 non-null  int64
 20  Gtp                  55692 non-null  int64
 21  dental caries        55692 non-null  int64
 22  smoking              38984 non-null  float64
 23  dataset              55692 non-null  object
dtypes: float64(6), int64(17), object(1)
memory usage: 10.2+ MB
```

	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	hearing(right)	systolic	relaxation	...	LDL	hemoglobin	Urine protein	serum creatinine	AST	ALT	Gtp	dental caries	smok
0	35	170	85	97.0	0.9	0.9	1	1	118	78	...	142	19.8	1	1.0	61	115	125	1	
1	20	175	110	110.0	0.7	0.9	1	1	119	79	...	114	15.9	1	1.1	19	25	30	1	
2	45	155	65	86.0	0.9	0.9	1	1	110	80	...	112	13.7	3	0.6	1090	1400	276	0	
3	45	165	80	94.0	0.8	0.7	1	1	158	88	...	91	16.9	1	0.9	32	36	36	0	
4	20	165	60	81.0	1.5	0.1	1	1	109	64	...	92	14.9	1	1.2	26	28	15	0	

5 rows × 24 columns

2. Checking null values

```
# Check for missing values
null_counts = df.isnull().sum()
print("Null values in each column:\n", null_counts[null_counts > 0])
```

Output:

```
Null values in each column:
 smoking    16708
dtype: int64
```

Here smoking column is targeting column. So, no need to remove null values.

3. Checking duplicate rows

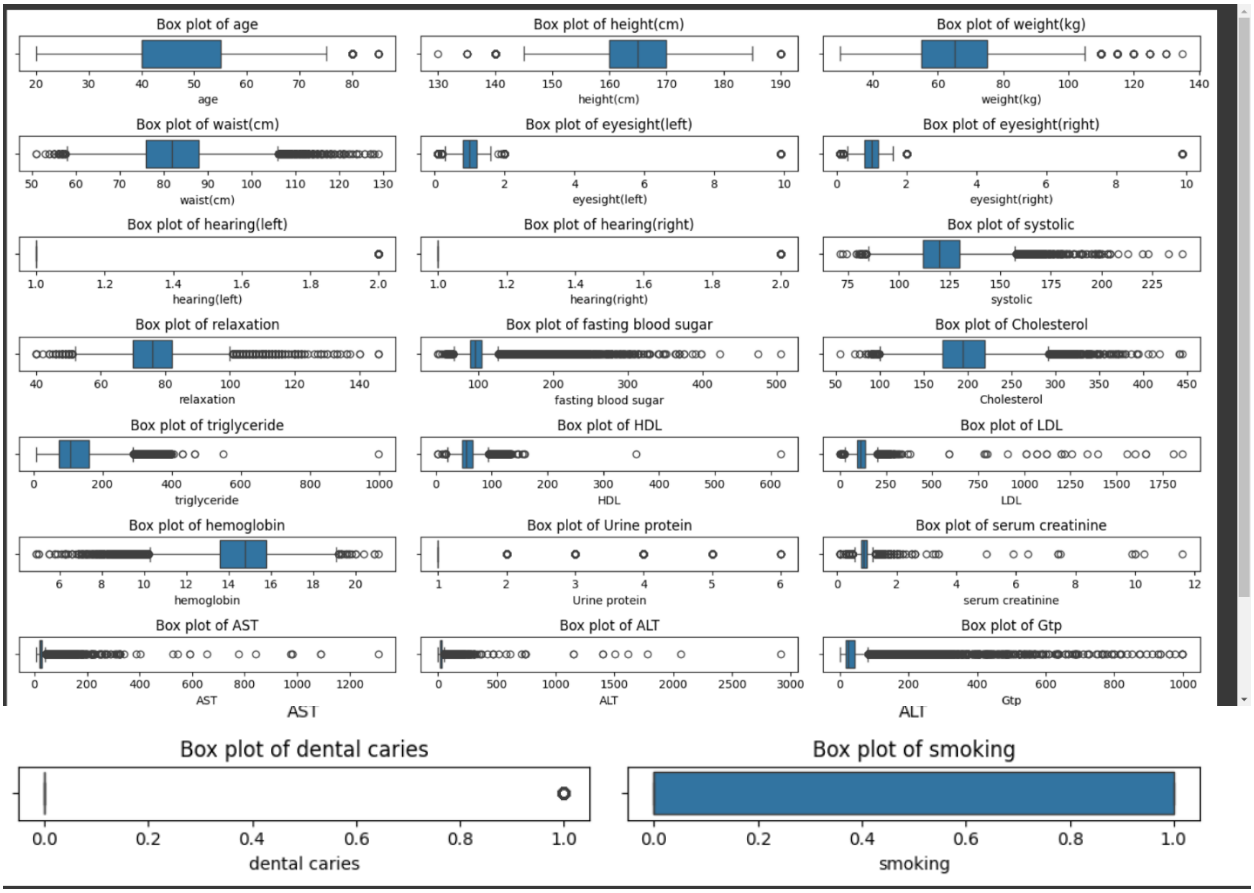
```
# Check for duplicate rows
duplicate_count = df.duplicated().sum()
print(f'Number of duplicate rows: {duplicate_count}')

# Remove duplicate rows
df_cleaned = df.drop_duplicates()

# Check the shape of the cleaned DataFrame
print(f'Original DataFrame shape: {df.shape}')
print(f'Cleaned DataFrame shape: {df_cleaned.shape}')
```

```
Number of duplicate rows: 6515
Original DataFrame shape: (55692, 24)
Cleaned DataFrame shape: (49177, 24)
```

4. Outlier visualization

```
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Select only numerical columns for outlier visualization
numeric_columns = df_cleaned.select_dtypes(include=[np.number]).columns

# Set up the plotting area
plt.figure(figsize=(15, 10))

# Generate box plots for each numerical feature
for i, col in enumerate(numeric_columns, 1):
    plt.subplot((len(numeric_columns) + 2) // 3, 3, i)
    sns.boxplot(data=df, x=col)
    plt.title(f'Box plot of {col}')

plt.tight_layout()
plt.show()
```

Output:



5. Removing outlier

```
# Identifying and removing outliers using the IQR method

# Select only numerical columns for outlier detection
numeric_columns = df_cleaned.select_dtypes(include=[np.number]).columns

# Initialize an empty DataFrame to store rows without outliers
df_no_outliers = df_cleaned.copy()

for col in numeric_columns:
    # Calculate Q1 (25th percentile) and Q3 (75th percentile) for the column
    Q1 = df_no_outliers[col].quantile(0.25)
    Q3 = df_no_outliers[col].quantile(0.75)
    IQR = Q3 - Q1

    # Define outlier boundaries
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR

    # Remove outliers
    df_no_outliers = df_no_outliers[(df_no_outliers[col] >= lower_bound) & (df_no_outliers[col] <= upper_bound)]

print("Original dataset shape:", df_cleaned.shape)
print("Dataset shape after removing outliers:", df_no_outliers.shape)
```

```
Original dataset shape: (49177, 24)
Dataset shape after removing outliers: (14949, 24)
```

In this step, I choose IQR method for outlier removal. The Interquartile Range method is a statistical procedure for dealing with outliers in a given set of data. It is concerned with the division of a data set into four no unequal parts. More specifically, the IQR is the range within which the central 50% of the data lies and hence is a better measure of variability as it is not skewed by the extreme values as much as the range is.

Steps for Computing IQR and Finding the Outliers:

a. Derive the Quartiles:
    - Q1 (First Quartile): The 25th percentile of the data which means that 25% of the data is below this value.
    - Q3 (Third Quartile): The 75th percentile of the data which means that 75% of the data is below this value.
b. Compute the IQR:

$$IQR = Q3 - Q1$$

c. Determine Outlier Boundaries:
    - Calculate the lower and upper bounds for outliers:

$$Lower\ Bound = Q1 - 1.5 \times IQR$$
$$Upper\ Bound = Q3 + 1.5 \times IQR$$

d. Identify Outliers:
    - An outlier refers to any point of data that is less than the lower limit or greater than the upper limit.

Advantages of IQR Approach:

- Resilience: IQR has lesser effects of outliers than any other range which is why it is a better approach in outlier detection.
- Ease of use: It is easy to comprehend the concept and incorporate it.
- Non-parametric: There is no need of normal distribution of data so it can fit to various other sources of data.

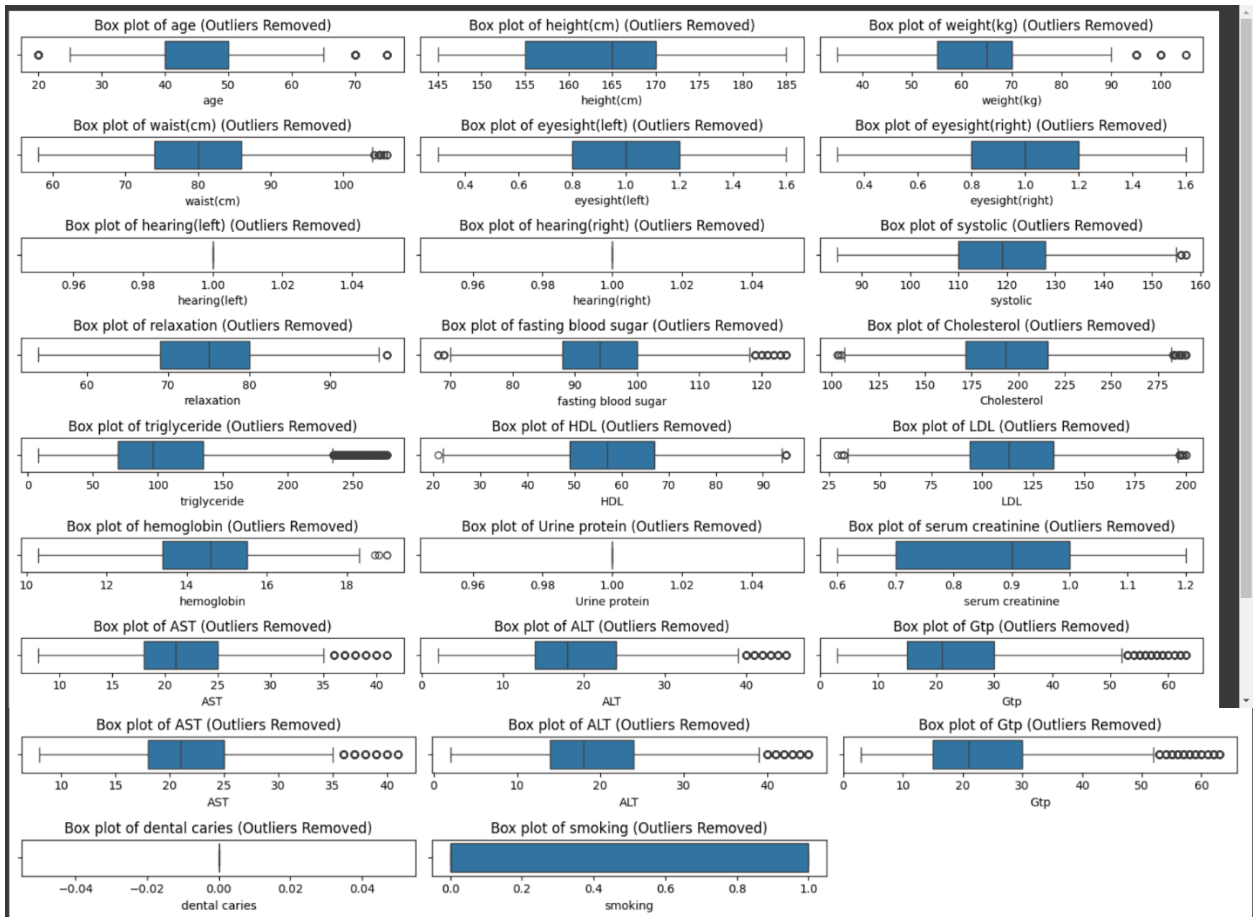
Drawbacks:

- Sample Size Dependency: For small size dataset few data points may affect the quartiles which results to non-reliable conclusions.
- Outlier Definition: IQR times one and a half is a bit of a cut off who's value maybe varies with the context of the analysis.

The IQR approach is usually carried out in the course of data cleaning during exploratory data analysis before machine learning algorithms can be applied.

6. After remove outlier visualization

```
# Visualize the dataset after outlier removal
plt.figure(figsize=(15, 10))

# Generate box plots for each numerical feature after outlier removal
for i, col in enumerate(numeric_columns, 1):
    plt.subplot((len(numeric_columns) + 2) // 3, 3, i)
    sns.boxplot(data=df_no_outliers, x=col)
    plt.title(f'Box plot of {col} (Outliers Removed)')

plt.tight_layout()
plt.show()
```



7. Standardize numerical columns

```
from sklearn.preprocessing import StandardScaler

# Standardize numerical columns
scaler = StandardScaler()
numeric_cols = df_no_outliers.select_dtypes(include=[np.number]).columns
df_no_outliers[numeric_cols] = scaler.fit_transform(df_no_outliers[numeric_cols])
```

```
df_no_outliers.head(10)
```

Output:

	age	height(cm)	weight(kg)	waist(cm)	eyesight(left)	eyesight(right)	hearing(left)	hearing(right)	systolic	relaxation	...	LDL	hemoglobin	Urine protein	serum creatinine	AST	AL
9	0.123138	-1.005403	-0.761314	-0.247974	-1.054869	-0.024305	0.0	0.0	-0.417852	0.776571	...	-0.259724	-0.970333	0.0	-1.659746	-0.037337	-0.62981
11	0.556877	-1.005403	-1.198301	-0.971107	-1.731734	-0.702813	0.0	0.0	-0.582582	-1.232677	...	-1.535169	-1.397880	0.0	0.177148	-0.037337	-1.00265
14	-0.310601	-1.551881	-2.072275	-2.176329	-0.039573	1.671963	0.0	0.0	-1.488595	-0.287148	...	1.754135	-0.756560	0.0	0.789446	0.147480	-0.50553
19	0.990616	1.180507	0.549646	0.595681	0.637291	0.654202	0.0	0.0	-0.747312	-0.523531	...	-1.400912	0.312306	0.0	-0.435150	-0.406970	-0.50553
20	0.123138	1.180507	0.549646	-0.006929	0.637291	0.654202	0.0	0.0	0.899986	0.658380	...	1.149977	0.241048	0.0	1.401744	0.886746	0.98581
21	-1.611819	0.634030	0.549646	0.354637	0.637291	1.671963	0.0	0.0	-0.088393	0.421998	...	-0.628932	1.309915	0.0	1.401744	-0.591786	-1.00265
22	-0.310601	0.087552	-0.324327	-0.440809	-0.378005	-1.042066	0.0	0.0	-1.570960	-1.469059	...	-1.702991	-1.967942	0.0	-0.435150	-0.222153	-1.12693
26	0.990616	-1.005403	-1.198301	-0.609540	-1.054869	-2.059827	0.0	0.0	0.488162	-0.523531	...	1.619878	-2.039200	0.0	-1.047448	0.332296	-0.00842
29	2.291834	-0.458926	-1.198301	-0.127452	-0.716437	-1.720573	0.0	0.0	0.076337	-0.523531	...	1.015720	0.241048	0.0	-1.659746	-1.515868	-1.12693
31	0.990616	-0.458926	-0.761314	-0.127452	-0.039573	-0.024305	0.0	0.0	-0.253122	0.067425	...	-0.461110	-0.400271	0.0	0.177148	2.734910	0.61297

10 rows × 24 columns

Standardization of numerical columns is one of the important steps in the preprocessing of a lot of machine learning applications. There are several explanations as to why standardization is necessary:

I. To Rescale Different Features into One Scale:
   - Magnitude Differences: Scales of features may vary widely (for example, age would be in years compared to income which would be in thousands or millions). Standardization makes certain that all dimensions contribute the same to the distance metrics which the algorithms use.
   - Avoidance of Bias: Range of features may be causing some standardization which if performed will allow the larger values to dominate the learning process, which will result in bias.

II. Enhancement of Model Accuracy
   - Convergence Speed: For optimization algorithms, such as gradient descent, standardization may provide quicker convergence. This is particularly true for computational procedures in which the order of input features is essential, such as:
     - ✓ Support Vector Machine (SVM): Since SVM is based on distances, the use of raw data will misplace the optimal hyperplane.
     - ✓ K-Means Clustering: Distance computed based algorithms like K-Means, the different scales would adversely affect the output.

III. Rationales for Many Algorithms:
   - There are algorithms that work on the principle that the data is distributed normally (examples are Linear Regression, Logistic Regression). Standardization may assist in these regards.
   - PCA is another example of an algorithm that performs poorly if data is not standardized of Principal Component subspaces since the algorithm processes the data and finds the subspaces or components Proportionally with the Data Variance.

IV. Regularization:
   - Regularization approaches (such as Lasso and Ridge regression) impose cost functions proportional to the size of the coefficients. The model might overfit and prioritize some features per their scales instead of their useful contributions if the features are not normalized.

When Introducing Standardization is Relevant:

- The standardization must be used with the features that have different units or scales and still, the algorithm that will be applied is scale sensitive. Differently scaled data contain different measurement features and this is more common, especially with distribution sensitive applications.

Summary:

- Adjusting the scales of numeric features to a common range is a must process in data preprocessing phase for many machine learning techniques as all the features present tend to have similar scale, which enhances the model performance and interpretation.