

EXPERIMENT 3

Aim: Experiment to preprocess dataset using different preprocessing techniques.

Theory:

1. Data Cleaning - Handling Missing Values

Missing values occur when data points are absent in a dataset. Handling missing values is essential to ensure data quality and improve model performance.

Techniques to Handle Missing Values

- **Removing Missing Values:**

If a dataset has a large number of missing values in a column, that column can be dropped if it doesn't add much value.

If certain rows have missing values and are not significant, they can be removed.

- **Replacing Missing Values (Imputation):**

Mean/Median/Mode Imputation: For numerical data, missing values can be replaced with the mean, median, or mode of the column.

Forward/Backward Fill: Uses previous or next values to fill missing data (common in time-series data).

Using Predictive Models: Missing values can be estimated using regression or machine learning models.

2. Data Cleaning - Removing Noisy Values & Outliers

Noisy data refers to random variations, errors, or irrelevant information in a dataset.

Binning Technique (For Noisy Data)

Binning is used to smooth data by dividing it into bins and replacing values with bin averages or boundaries.

Types of Binning:

- **Equal-Width Binning:** The range of values is divided into equal intervals.
- **Equal-Frequency Binning:** Each bin contains approximately the same number of data points.

Removing Outliers Using the Interquartile Range (IQR) Method

Outliers are extreme values that deviate significantly from other observations.

Steps:

- 1) Compute Q1 (25th percentile) and Q3 (75th percentile).
- 2) Compute IQR:
 $IQR = Q3 - Q1$
- 3) Define outlier boundaries:
Lower bound = $Q1 - 1.5 \times IQR$
Upper bound = $Q3 + 1.5 \times IQR$
- 4) Any value outside this range is considered an outlier.

Boxplot for Outlier Detection

A boxplot visually represents the spread of data, including quartiles and potential outliers. Outliers appear as individual points beyond the whiskers.

3. Data Transformation - Converting Data Types

Data transformation involves converting data into a more suitable format.

Numerical to Categorical

- Discretization: Converts continuous numerical values into categories.

Example: Converting age into categories like "child," "teen," "adult," and "senior."

Categorical to Numerical (One-Hot Encoding)

- Used for machine learning models that require numerical input.
- Each categorical value is converted into binary columns.

Example:

Color: ['Red', 'Blue', 'Green']

One-hot encoded:

Red	Blue	Green
1	0	0
0	1	0
0	0	1

4. Data Transformation - Normalization (Z-Score Standardization)

Normalization ensures that all numerical attributes have a similar scale to improve model efficiency.

Z-Score Normalization

Converts data to a standard scale with a mean of 0 and a standard deviation of 1.

Formula:

$$Z = (X - \mu) / \sigma$$

where:

X is the original value,

μ is the mean,

σ is the standard deviation.

Why Use Z-Score Normalization?

- Helps in algorithms that assume normally distributed data.
- Reduces bias caused by different data ranges.

5. Data Reduction - Reducing Rows (Attribute-Oriented Induction & Numerosity Reduction)

Data reduction improves efficiency by reducing data size while preserving essential information.

Attribute-Oriented Induction

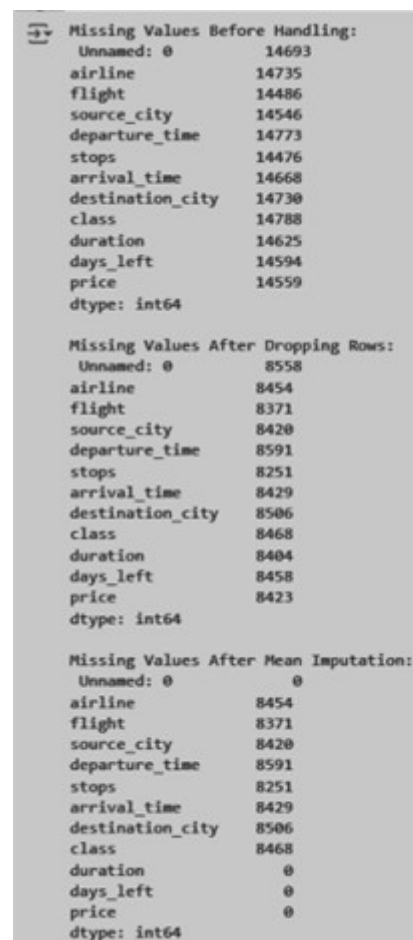
- Aggregates data by higher-level attributes.
- Example: Instead of storing individual transactions, aggregate data by category (e.g., monthly sales).

Numerosity Reduction

- Parametric Methods: Approximate data using models (e.g., regression, clustering).
- Non-Parametric Methods: Reduce data through sampling, histogram, or clustering.

Steps:**1) Data Cleaning - removing missing values(demonstrate removing and replacing Null values)****Code:**

```
print("Missing Values Before Handling:\n", df.isnull().sum())
if df.isnull().sum().sum() <= 2:
    df = df.dropna().reset_index(drop=True)
else:
    df = df.dropna(thresh=df.shape[1] - 1).reset_index(drop=True)
print("\nMissing Values After Dropping Rows:\n", df.isnull().sum())
df.fillna(df.mean(numeric_only=True), inplace=True)
print("\nMissing Values After Mean Imputation:\n", df.isnull().sum())
df.fillna(df.median(numeric_only=True), inplace=True)
print("\nMissing Values After Median Imputation:\n", df.isnull().sum())
for col in df.columns:
    if df[col].dtype == "object":
        df[col].fillna(df[col].mode()[0], inplace=True)
print("\nMissing Values After Mode Imputation:\n", df.isnull().sum())
print("\nOverall Missing Values Left in Dataset:", df.isnull().sum().sum())
df.to_csv("Processed_Dataset.csv", index=False)
```



```
Missing Values Before Handling:
Unnamed: 0      14693
airline         14735
flight          14486
source_city     14546
departure_time  14773
stops           14476
arrival_time    14668
destination_city 14730
class           14788
duration        14625
days_left      14594
price           14559
dtype: int64

Missing Values After Dropping Rows:
Unnamed: 0      8558
airline         8454
flight          8371
source_city     8420
departure_time  8591
stops           8251
arrival_time    8429
destination_city 8506
class           8468
duration        8404
days_left      8458
price           8423
dtype: int64

Missing Values After Mean Imputation:
Unnamed: 0      0
airline         8454
flight          8371
source_city     8420
departure_time  8591
stops           8251
arrival_time    8429
destination_city 8506
class           8468
duration         0
days_left       0
price            0
dtype: int64
```

```

Missing Values After Median Imputation:
Unnamed: 0      0
airline      8454
flight      8371
source_city  8420
departure_time 8591
stops      8251
arrival_time 8429
destination_city 8506
class      8468
duration      0
days_left    0
price        0
dtype: int64

Missing Values After Mode Imputation:
Unnamed: 0      0
airline      0
flight      0
source_city  0
departure_time 0
stops      0
arrival_time 0
destination_city 0
class      0
duration      0
days_left    0
price        0
dtype: int64

✔ Overall Missing Values Left in Dataset: 0

```

The code handles missing values in a dataset using multiple techniques. It first checks the number of missing values in each column and prints the count. If the total number of missing values in the dataset is 2 or fewer, it removes all rows with missing values; otherwise, it removes only those rows where all but one column have missing values ($\text{thresh} = \text{df.shape}[1] - 1$). After dropping rows, it imputes missing values using different strategies: numerical columns are first filled with their mean values and then with their median values. For categorical (object-type) columns, missing values are replaced with the most frequent value (mode). Finally, the script prints the remaining missing values and saves the cleaned dataset as "Processed_Dataset.csv". This approach ensures that missing data is handled efficiently while preserving as much useful information as possible.

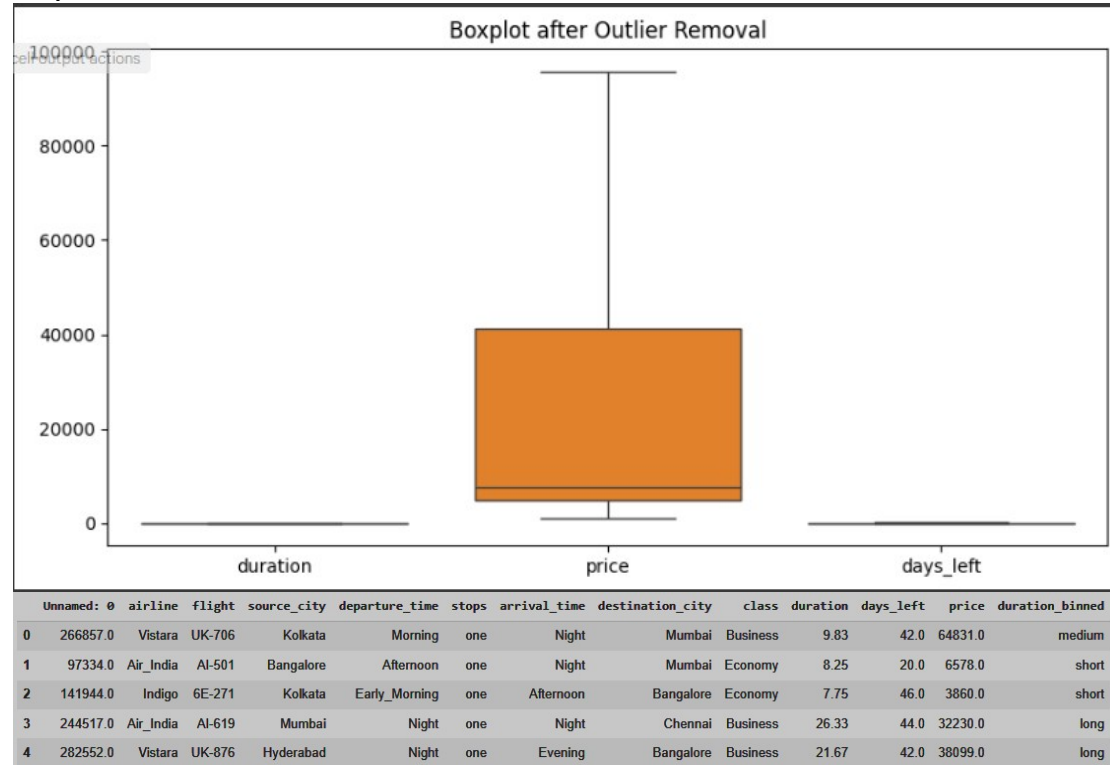
2) Data Cleaning - removing noisy values(Binning technique), removing outliers- Interquartile Range Method,Boxplot

Code:

```

df['duration_binned'] = pd.qcut(df['duration'], q=3, labels=['short', 'medium', 'long'])
def remove_outliers_iqr(data, col):
    Q1 = data[col].quantile(0.25)
    Q3 = data[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    return data[(data[col] >= lower_bound) & (data[col] <= upper_bound)]
outlier_cols = ['duration', 'price', 'days_left']
for col in outlier_cols:
    df = remove_outliers_iqr(df, col)
plt.figure(figsize=(10, 5))
sns.boxplot(data=df[outlier_cols])
plt.title("Boxplot after Outlier Removal")
plt.show()
df.to_csv('cleaned_no_outliers.csv', index=False)
df.head()

```

Output:

The code performs data cleaning by handling outliers and binning numerical values. It first bins the "duration" column into three categories—short, medium, and long—using the `qcut` function, which divides data into quantile-based bins. Then, it defines a function `remove_outliers_iqr()` that applies the Interquartile Range (IQR) method to detect and remove outliers. This function calculates the first (Q1) and third quartile (Q3), determines the IQR ($Q3 - Q1$), and removes values outside 1.5 times the IQR from the dataset. The function is applied to the "duration," "price," and "days_left" columns to remove outliers. After cleaning, a boxplot is generated using Seaborn to visualize the data distribution after outlier removal. Finally, the cleaned dataset is saved as "cleaned_no_outliers.csv", and the first few rows are displayed using `df.head()`.

Data Smoothing:**Code:**

```
def mean_smoothing(data, col):
    return data.groupby(f'{col}_binned')[col].transform('mean')
def median_smoothing(data, col):
    return data.groupby(f'{col}_binned')[col].transform('median')
def min_max_smoothing(data, col):
    return data.groupby(f'{col}_binned')[col].transform(lambda x: (x.min() + x.max()) / 2)
df['duration_mean_smooth'] = mean_smoothing(df, 'duration')
df['duration_median_smooth'] = median_smoothing(df, 'duration')
df['duration_minmax_smooth'] = min_max_smoothing(df, 'duration')
print("Original Duration Data (Before Smoothing):")
print(df[['duration', 'duration_binned']].head())
print("\nSmoothed Duration Data:")
print(df[['duration_mean_smooth', 'duration_median_smooth', 'duration_minmax_smooth']].head())
```

Output:

Original Duration Data (Before Smoothing):		
	duration	duration_binned
0	9.83	medium
1	8.25	short
2	7.75	short
3	26.33	long
4	21.67	long

Smoothed Duration Data:			
	duration_mean_smooth	duration_median_smooth	duration_minmax_smooth
0	11.323642	11.50	11.250
1	4.973680	5.50	4.625
2	4.973680	5.50	4.625
3	20.022477	19.42	21.750
4	20.022477	19.42	21.750

Smoothing is a crucial preprocessing step that reduces noise and ensures data consistency before analysis. By applying mean, median, and min-max smoothing to the duration column after binning (short, medium, long), we minimize fluctuations that could impact machine learning models. For example, durations like 8.25 and 7.75 in the "short" category are adjusted to more consistent values (4.97 mean, 5.50 median, 4.63 min-max), reducing variability while preserving trends.

Each method affects data differently—mean smoothing balances values, median smoothing is robust to outliers, and min-max smoothing considers the range. By stabilizing the dataset, smoothing enhances transformations like scaling and encoding, ultimately improving model performance and interpretability.

3) Data Transformation - converting numerical attributes to categorical and vice versa/ one hot encoding

Code:

```
df['days_left_binned'] = pd.qcut(df['days_left'], q=3, labels=['short', 'medium', 'long'])
categorical_cols = ['airline', 'source_city', 'destination_city', 'class', 'departure_time', 'arrival_time', 'stops']
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
scaler = StandardScaler()
numerical_cols = ['duration', 'days_left', 'price']
df_encoded[numerical_cols] = scaler.fit_transform(df_encoded[numerical_cols])
df_encoded.to_csv('transformed_data.csv', index=False)
print("Data Transformation Completed")
df_encoded.head()
```

Output:

<

The code into a numerical format. The "days_left" column is categorized into three groups—short, medium, and long—using `qcut()`, which divides the data into equal-sized quantiles. Then, categorical columns such as "airline", "source_city", "destination_city", "class", "departure_time", "arrival_time", and "stops" are transformed into numerical format using one-hot encoding via `pd.get_dummies()`. This process creates binary columns for each category while dropping the first category to avoid multicollinearity (`drop_first=True`). These transformations ensure that categorical variables are properly represented for machine learning models.

4) Data Transformation - data normalization(Z- score transformation)

Code:

```
df['days_left_binned'] = pd.qcut(df['days_left'], q=3, labels=['short', 'medium', 'long'])
categorical_cols = ['airline', 'source_city', 'destination_city', 'class', 'departure_time', 'arrival_time', 'stops']
df_encoded = pd.get_dummies(df, columns=categorical_cols, drop_first=True)
scaler = StandardScaler()
numerical_cols = ['duration', 'days_left', 'price']
df_encoded[numerical_cols] = scaler.fit_transform(df_encoded[numerical_cols])
df_encoded.to_csv('transformed_data.csv', index=False)
print("Data Transformation Completed")
df_encoded.head()
```

Output:

Data Transformation Completed

	Unnamed: 0	flight	duration	days_left	price	duration_binned	days_left_binned	airline_Air_India	airline_GO_FIRST	airline_Indigo	...	departure_time_Late_Night	departure_time_Morning	departure_time_Night
0	266857.0	UK-706	-0.322703	1.195474	1.971692	medium	long	False	False	False	...	False	True	False
1	97334.0	AI-501	-0.553636	-0.455088	-0.640918	short	medium	True	False	False	...	False	False	False
2	141944.0	6E-271	-0.626717	1.495576	-0.762018	short	long	False	False	True	...	False	False	False
3	244517.0	AI-619	2.088945	1.345525	0.509558	long	long	True	False	False	...	False	False	True
4	282552.0	UK-876	1.407837	1.195474	0.772779	long	long	False	False	False	...	False	False	True

5 rows × 35 columns

To standardize numerical attributes, the script applies Z-score normalization using `StandardScaler()`. This transformation ensures that all numerical values—"duration," "days_left," and "price"—are rescaled to have a mean of 0 and a standard deviation of 1. Standardization is crucial for models that rely on distance-based calculations, such as regression, clustering, and neural networks, as it prevents features with larger scales from dominating others. Finally, the transformed dataset is saved as "transformed_data.csv", and the first few rows are displayed to confirm the changes.

5)Data Reduction - reducing the number of rows by attribute-oriented induction or numerosity reduction

Code:

```
df_encoded = pd.read_csv('transformed_data.csv')
numeric_cols = df_encoded.select_dtypes(include=['number']).drop(columns=['days_left'], errors='ignore')
df_reduced = df_encoded[['days_left']].join(numeric_cols).groupby('days_left').mean().reset_index()

df_reduced.to_csv('reduced_data.csv', index=False)
print("Data Reduction Completed")
df_reduced.head()
```

Output:

Data Reduction Completed

	days_left	Unnamed: 0	duration	price
0	-1.880574	98172.606295	0.253458	0.008474
1	-1.805549	149899.453762	0.217380	0.380897
2	-1.730523	147427.915016	0.254747	0.335667
3	-1.655497	149481.970257	0.205601	0.209353
4	-1.580472	152701.884901	0.092970	0.233429

This code applies data reduction using attribute-oriented induction, where detailed data is aggregated to a higher-level summary. It first loads the transformed dataset (transformed_data.csv) and selects all numerical columns while excluding "days_left" from this selection. Then, it performs grouping based on "days_left" and calculates the mean of all numerical attributes within each group. This reduces the dataset's size by replacing multiple individual records with aggregated values, making it easier to analyze trends while preserving essential information. The reduced dataset is saved as "reduced_data.csv", and the first few rows are displayed to confirm the transformation.

Conclusion:

The data preprocessing steps implemented in this workflow ensure that the dataset is clean, structured, and optimized for analysis or machine learning models. Data cleaning was performed by handling missing values and outliers, improving data reliability. Data transformation techniques, including binning, one-hot encoding, and Z-score normalization, were applied to convert categorical and numerical attributes into a suitable format for modeling. Finally, data reduction through attribute-oriented induction helped summarize the dataset, reducing its size while preserving key information. These preprocessing steps enhance data quality, improve model performance, and ensure efficient processing for further analysis.