Data Preprocessing

Data pre-processing is a crucial step in data science that involves transforming raw data into a format suitable for analysis.It improves the quality and structure of the dataset to ensure that models can make accurate predictions. Here is an extensive overview of the data preprocessing , starting with Data Cleaning , followed by other sub-tasks:

Data Cleaning

Data cleaning is the process of detecting and correcting (or removing) error , inconsistencies , and inaccuracies from datasets to improve data quality.It ensures that the data used for analysis is free of noise , incomplete values , and duplicate entries.

Why is Data Cleaning Important ?

Real-world data is often messy and can contain missing values , errors or outliers.Cleaning data ensures that your analysis or model is built on relible , consistent , and valid data , leading to more accurate and insightful results.

Sub-tasks of Data Cleaning :

1. Handling Missing Data Missing data is a common problem that occures when some observations in the dataset lack a certain value. Why : Algorithms cannot process missing values directly. Methods : Removal : Deleting rows or columns with missing data if they are few. Imputaiton : Replacing missing values with the sum , median , mode or more complex methods like interpolation.

In [1]:
```
import pandas as pd
df = pd.DataFrame({'Name' : ['Ram gopal' , 'Puri' , 'Sandeep' , None,'Ram gopal'],
                   'Age' : [45 , 40 , None , 32,45],
                   'Salary' : [50000 , None , 55000,48000,50000],
                  'City' : ['Aamravathi','Guntur','Vizag','MR Kunta','Aamravathi']})
#remove rows with missing data
df_clean = df.dropna() # When we are using dropna we have to mention inplace=True
# impute missing values with the mean
df['Age'].fillna(df['Age'].mean() , inplace = True)
df['Salary'].fillna(df['Salary'].mean() , inplace = True)
df
```

Out[1]:

	Name	Age	Salary	City
0	Ram gopal	45.0	50000.0	Aamravathi
1	Puri	40.0	50750.0	Guntur
2	Sandeep	40.5	55000.0	Vizag
3	None	32.0	48000.0	MR Kunta
4	Ram gopal	45.0	50000.0	Aamravathi

2. Removing Duplicates

Why : Duplicate data can distort the analysis. Real Example : Suppose you are analyzing customer data , and a customer's record appears multiple times. This can affect the accuracy of customer segmentation model.

In [2]:
```
df.drop_duplicates(inplace = True)
df
```

Out[2]:

	Name	Age	Salary	City
0	Ram gopal	45.0	50000.0	Aamravathi
1	Puri	40.0	50750.0	Guntur
2	Sandeep	40.5	55000.0	Vizag
3	None	32.0	48000.0	MR Kunta

3. Handling outliers

Why : Outliers can skew statistical models and provide misleading results. Methods : Removing :Detect and remove outliers based on the IQR(Interquartile Range) or Z-score. Transforming : Normalize the data to reduce the impact of outliers.

IQR

In [3]:
```
nu_df = df.select_dtypes(include = 'number')
```

In [4]:
```
Q1 = nu_df.quantile(0.25)
Q3 = nu_df.quantile(0.75)
IQR = Q3 - Q1
IQR
```

Out[4]:
```
Age          3.625
Salary    2312.500
dtype: float64
```

In [5]:
```
df.describe()
```

Out[5]:

	Age	Salary
count	4.000000	4.000000
mean	39.375000	50937.500000
std	5.406401	2946.572868
min	32.000000	48000.000000
25%	38.000000	49500.000000
50%	40.250000	50375.000000
75%	41.625000	51812.500000
max	45.000000	55000.000000

In [6]:
```
df =  df[~((nu_df < (Q1 - 1.5 * IQR)) |(nu_df > (Q3 + 1.5 * IQR))).any(axis=1)]
df.shape
```

Out[6]:
```
(3, 4)
```

Z-Score

In [7]:
```
#detecting and handling outliers using z-score
import warnings
warnings.filterwarnings('ignore')
from scipy import stats as st
import numpy as np
df['zscore'] = np.abs(st.zscore(df['Salary']))
df_no_outliers = df[df['zscore']<3] #removing outliers
```

In [8]:
```
df
```

Out[8]:

	Name	Age	Salary	City	zscore
0	Ram gopal	45.0	50000.0	Aamravathi	0.870563
1	Puri	40.0	50750.0	Guntur	0.529908
2	Sandeep	40.5	55000.0	Vizag	1.400471

Correct Data Types

Why : Incorrect data types (e.g., numeric values stored as strings) can prevent analysis or cause computational errors.

In [9]:
```
df['Age'] = pd.to_numeric(df['Age'] ,errors = 'coerce') #coerce invalid type to NaN
df
```

Out[9]:

	Name	Age	Salary	City	zscore
0	Ram gopal	45.0	50000.0	Aamravathi	0.870563
1	Puri	40.0	50750.0	Guntur	0.529908
2	Sandeep	40.5	55000.0	Vizag	1.400471

Addressing Inconsistences

Why : Inconsistent data (e.g., different date formats or inconsistent text capitalization)can lead to incorrect grouping or analysis.
Real Example : A dataset where "Andhra" and "andhra" considered different locations.

In [10]:
```
df['City'] = df['City'].str.lower()
df
```

Out[10]:

	Name	Age	Salary	City	zscore
0	Ram gopal	45.0	50000.0	aamravathi	0.870563
1	Puri	40.0	50750.0	guntur	0.529908
2	Sandeep	40.5	55000.0	vizag	1.400471

Data Integration

What is Data Intergration? Data integration involves combining from multiple sources into unified dataset.It is essential when you're working from different databases or systems

Why is Data Integration Important? In real-world projects , data often comes from various sources , such as transactional systems, customer relationship management(CRM) tools, and online platforms.Integrating these dataset ensures you can analyse all the data comprehensively.

Methods :

1. Merging Datasets : Combining two or more datasets based on a common columns (e.g., customer_id).
2. Concatenating Datasets :Stacking datsets on top of one other(i.e.,adding rows)
3. Joining Tables : Combining tables using different join types (inner,outer,left,right).

In [11]:
```
df_director = pd.DataFrame({'D_id' : [1,2,3],
                            'Name' : ['Ram Gopal','Puri','Sandeep']})
df_remu = pd.DataFrame({'D_id' : [1,2,3],
                        'Remuneration' : [500000 , 400000 , 450000 ],
                      'Movies' : [20,25,3]})
df_merged = pd.merge(df_director , df_remu , on ='D_id')
df_merged
```

Out[11]:

	D_id	Name	Remuneration	Movies
0	1	Ram Gopal	500000	20
1	2	Puri	400000	25
2	3	Sandeep	450000	3

Data Transformation

What is Data Transformation?

Data transformation is the process of converting data into a format that is more appropriate for analysis.It involves scaling , encoding categorical data , and feature engineering.

Why is Data Transformation Important?

Raw data often needs to be normalized or encoded into format suitable for machine learning algorithms , which work best with numeric and scaled data.

Sub-Tasks of Data Transforamtion :

1. Normalization and Scaling :

```
                Why : Some algorithms (like distance-based models) are sensitive to the scale
of features.
                Methods :
        Min-Max Scaling : Rescales data to a range [0,1].
        Standradization : Rescales data so that it has a mean of 0
                          and a standard deviation of 1.
```

In [12]:
```
from sklearn.preprocessing import MinMaxScaler , StandardScaler
scaler = MinMaxScaler()
df_scaled = scaler.fit_transform(df[['Age' , 'Salary']])
df_scaled
```

Out[12]:
```
array([[1.  , 0.  ],
       [0.  , 0.15],
       [0.1 , 1.  ]])
```

2. Encoding Categorical Data

```
Why : Machine learning models cannot process non-numeric data.
```

```
Methods :
     Label Encoding : Converts categories to numeric labels.
     One-Hot Encoding : Creates binary columns for each category.
```

In [13]:
```
df_encode = pd.get_dummies(df, columns = ['City'])
df_encode
```

Out[13]:

	Name	Age	Salary	zscore	City_aamravathi	City_guntur	City_vizag
0	Ram gopal	45.0	50000.0	0.870563	True	False	False
1	Puri	40.0	50750.0	0.529908	False	True	False
2	Sandeep	40.5	55000.0	1.400471	False	False	True

3. Feature Engineering

```
Why : Creating new features based on existing ones can improves models performance.
Real Example : Creating a total_spent feature from quantity and price
```

In [14]:
```
df_merged['Total_gain'] = df_merged['Remuneration']* df_merged['Movies']
df_merged
```

Out[14]:

	D_id	Name	Remuneration	Movies	Total_gain
0	1	Ram Gopal	500000	20	10000000
1	2	Puri	400000	25	10000000
2	3	Sandeep	450000	3	1350000

Data Reduction :

What is Data Reduction :

Data reduction technique aim to reduce the amount of data without losing significant information.This improves the efficiency of the analysis.

Why is Data Reduction Important?

Handling large datasets can be computationally expensive, so reducing the dataset size helps in faster and more efficient processing.

Methods

1. Dimensionality Reduction : Reducing the number of features using methods like Principle Component Analysis(PCA).
2. Aggregation : Summerizating the data (e.g., calculating total or averages) to reduce the dataset's granularity.

In [15]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 2)
df_reduced = pca.fit_transform(df[['Age' , 'Salary']])
df_reduced
```

Out[15]:
```
array([[-1.91666812e+03, -2.10673242e+00],
       [-1.16666547e+03,  2.47851013e+00],
       [ 3.08333360e+03, -3.71777704e-01]])
```

Data Discretization

What is Data Discretization?

Data discretization involves transforming continuous data into discrete intrevals or categories.

Why is Data Discretization Important?

Some algorithms work better with discrete values.Discretization can also make the results easier to interpret by grouping continuous data into ranges.

In [16]:
```
bins = [0,18,35,60,100]
labels = ['Child' , 'Young Adult' , 'Adult' , 'Senior']
df['Age_Group'] = pd.cut(df['Age'] , bins=bins , labels = labels)
df
```

Out[16]:

	Name	Age	Salary	City	zscore	Age_Group
0	Ram gopal	45.0	50000.0	aamravathi	0.870563	Adult
1	Puri	40.0	50750.0	guntur	0.529908	Adult
2	Sandeep	40.5	55000.0	vizag	1.400471	Adult

Real-Time Examples of Data Preprocessing

Imagine you are working with a dataset of customer transactions for a retail company.The dataset includes customer details , product information , and purchase history.You aim to predict customer churn(whether a customer will stop purchaging).

1.Data Cleaning :

Handle missing values in the Age and salary columns. Remove duplicate entries where the same transaction is recorded twice. Detect and remove outliers in the oreder_value column.

2.Data Integration :

Merge customer demographic data with transaction data Combine external datsets , such as customer feedback surveys.

3.Data Transformation :

Scale the order_value and customer_tenure columns to ensure they are on a similar scale. Encode the customer_type (regular,new,VIP) using one-hot encoding.

4.Data Reduction :

Use PCA to reduce the dimensionality of features like customer_activity and purchase_history.

5.Data Discretization :

Group the customer_tenure into bins such as new_customer , medium_tenure and long_tenure.

This comprehensive preprocessing will prepare the data for machine learning models ensure that it is clean, concictent and well-structured for analysis.

Let's use a real-world dataset to apply all the preprocessing techniques mentioned above. For this, I'll use the famous "Titanic" dataset, which contains information about the passengers on the Titanic, such as age, sex, class, fare, etc., and whether they survived or not. This dataset is available in the Seaborn library.

we'll perform the following steps:

1. Data Cleaning
2. Data integration
3. Data Transformation
4. Data Reduction
5. Data Discretization

Step 1: Loading the Titanic Dataset

In [17]:

```
import pandas as pd
import seaborn as sns
# Load the Titanic dataset from seaborn
df = sns.load_dataset('titanic')
#Display the first few rows os the dataset
df.head()
```

Out[17]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

The dataset includes the following columns:

1. survived: 1 if the passenger survived, 0 otherwise.
2. pclass: Passenger class (1st, 2nd, 3rd).
3. sex: Gender.

4. age: Age in years.
5. sibsp: Number of siblings/spouses aboard.
6. parch: Number of parents/children aboard.
7. fare: Passenger fare.
8. embarked: Port of embarkation (C = Cherbourg; Q = Queenstown; S = Southampton).
9. deck: Deck level.
10. embark_town: Town of embarkation.

Step 2: Data Cleaning

Handling Missing Data

In [18]:

```
#checking for missing values
print(df.isnull().sum())
#Filling missing 'Age' values with the mean
df['age'].fillna(df['age'].mean() , inplace = True)
#Dropping columns with too many missing values (like'deck')
df.drop(columns=['deck'] , inplace = True)
#Dropping rows with missing 'embarked'
df.dropna(subset=['embarked'] , inplace = True)
#Verify the cleaned data
df.isnull().sum()
```

```
survived         0
pclass           0
sex              0
age            177
sibsp            0
parch            0
fare             0
embarked         2
class            0
who              0
adult_male       0
deck           688
embark_town      2
alive            0
alone            0
dtype: int64
```

Out[18]:

```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
embark_town    0
alive          0
alone          0
dtype: int64
```

Handling Ouliers

We will use the IQR method to detect and remove outliers in the fare column.

In [19]:

```
#Detecting outliers using IQR in 'fare'
Q1 = df['fare'].quantile(0.25)
Q3 = df['fare'].quantile(0.75)
IQR = Q3-Q1
IQR
```

Out[19]:

```
23.1042
```

In [20]:

```
#Removing outliers
df = df[~(df['fare']<(Q1 - 1.5 * IQR)) | (df['fare']> (Q3 +1.5 * IQR))]
#Checking if outliers are removed
print(df.shape)
```

```
(889, 14)
```

Step 3 : Data Integration

For this dataset, we already have a unified table, so there's no need for merging or concatenating additional datasets. However, if we had more data sources, we would use techniques like merging or joining.

Step 4 : Data Transformation

4.1 Scaling Numeric Data

We will scale the fare and age columns using Min-Max scaling.

In [21]:
```
from sklearn.preprocessing import MinMaxScaler
#initilize MinMaxScaler
scaler = MinMaxScaler()
#Scaling 'fare' and 'age'
df[['age' , 'fare']] = scaler.fit_transform(df[['age' , 'fare']])
#Verify the scaled columns
print(df[['age' , 'fare']].head)
```

```
<bound method NDFrame.head of           age      fare
0    0.271174  0.014151
1    0.472229  0.139136
2    0.321438  0.015469
3    0.434531  0.103644
4    0.434531  0.015713
..        ...       ...
886  0.334004  0.025374
887  0.233476  0.058556
888  0.367921  0.045771
889  0.321438  0.058556
890  0.396833  0.015127

[889 rows x 2 columns]>
```

4.2 Encoding Categorical Data

We will encode the categorical columns(sex,embrked,class) using one-hot encoding

In [22]:
```
#one-hot encoding for categorical columns
df = pd.get_dummies(df , columns=['sex', 'embarked' , 'class'], drop_first= True)
df.head()
```

Out[22]:

	survived	pclass	age	sibsp	parch	fare	who	adult_male	embark_town	alive	alone	sex_male	embarked_Q	en
0	0	3	0.271174	1	0	0.014151	man	True	Southampton	no	False	True	False	
1	1	1	0.472229	1	0	0.139136	woman	False	Cherbourg	yes	False	False	False	
2	1	3	0.321438	0	0	0.015469	woman	False	Southampton	yes	True	False	False	
3	1	1	0.434531	1	0	0.103644	woman	False	Southampton	yes	False	False	False	
4	0	3	0.434531	0	0	0.015713	man	True	Southampton	no	True	True	False	

4.3 Feature Engineering

Let's create a new feature called family_size by combining sibs and parch to represent the total number of family members abord

In [24]:
```
#Creating 'family_size' feature
df['family_size'] = df['sibsp'] + df['parch']
#verify the new feature
print(df[['sibsp' , 'parch' , 'family_size']].head(5))
```

```
   sibsp  parch  family_size
0      1      0            1
1      1      0            1
2      0      0            0
3      1      0            1
4      0      0            0
```

Step 5 : Data Reduction

We will use Principle Component Analysis(PCA) to reduce the dimensionality of the dataset.Before doing that , let's remove the target varible 'survived' and non-numeric columns.

In [25]:
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
#Removing non-numberic columns and the target 'survied'
X = df.drop(columns=['survived' , 'who' , 'adult_male' , 'embark_town',
                    'alive','alone'])
#performaing PCA to reduce dimensionality
pca = PCA(n_components = 2)
X_reduced = pca.fit_transform(X)
#Dispaly the reduced dataset
print(X_reduced[:5])
```

```
[[ 0.22596941 -0.91841279]
 [ 0.1587736   1.38873859]
 [-1.03592723 -0.81457902]
 [ 0.17743301  1.36719003]
 [-1.08681302 -0.90966636]]
```

Step 6 : Data Discretization

We will discretize the 'age' column into categories such as Child , Young Adult, Adult and Senior.

In [26]:

```
#Binnig 'Age' into categories
bins = [0,18,35,60,100]
labels = ['Child' , 'Young Adult' , 'Adult' , 'Senior']
df['age_group'] = pd.cut(df['age'] , bins = bins , labels = labels)
#verify the new 'age_group' column
df[['age','age_group']].head()
```

Out[26]:

	age	age_group
0	0.271174	Child
1	0.472229	Child
2	0.321438	Child
3	0.434531	Child
4	0.434531	Child

Summary

1. Data Cleaning : we handled missing values, removed duplicates, and dealt with outliers.
2. Data Integration : We worked with a single dataset , but integration is important when multiple datsets are involved.
3. Data Transformation : We scaled numeric data, encoded categorical varible, and engineered new features like family_size.
4. Data Reduction : We applied PCA to reduce the dimensionality of the dataset.
5. Data Discretization : We binned the 'age' column into categories for better interpretability.

This comprehensive preprocessing workflow ensures that the data is clean, well-structured, and ready for analysis or model building