Five importnat ways for Imputing Missing Values

You can impute missing values using machine learning models. This process is known as data imputation and is commonly used in data preprocessing to handle missing or incomplete data. There are several methods and models you can use, depending on the nature of your data and the missing values:

1:- Simple Imputation Techniques:

Mean/Median Imputation:

Replace missing values with the mean or median of the column. Suitable for numerical data.

Mode Imputation:

Replace missing values with the mode (most frequent value) of the column. Useful for categorical data.

2:- K-Nearest Neighbors (KNN)

This algorithm can be used to impute missing values based on the similarity of rows.

3:- Regression Imputation

Use a regression model to predict the missing values based on other variables in your dataset.

4:- Decision Trees and Random Forests:

These can handle missing values inherently. They can also be used to predict missing values based on the patterns learned from the other data.

5:- Advanced Techniques:

Multiple Imputation by Chained Equations (MICE):

This is a more sophisticated technique that models each variable with missing values as a function of other variables in a round-robin fashion.

Deep Learning Methods:

Neural networks, especially autoencoders, can be effective in imputing missing values in complex datasets.

6:- Time Series Specific Methods:

For time-series data, you might use techniques like interpolation, forward-fill, or backward-fill.

It's important to choose the right method based on the type of data, the pattern of missingness (e.g., at random, completely at random, or not at random), and the amount of missing data. Additionally, it's crucial to understand that imputation can introduce bias or affect the distribution of your data, so it should be done with caution and an understanding of the potential implications.

1:- 1. Simple Imputation Techniques

1.1. Mean/Median Imputation

Mean/median imputation replaces missing values with the mean or median of the column. This is a simple and effective method, but it has some limitations. For example, it reduces variance in the dataset, and it can lead to biased estimates if the missing values are not missing at random.

Let's see how to implement mean/median imputation in Python using the Titanic dataset.

1.1.1. Mean Imputation

First, let's import the necessary libraries and load the dataset:

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
%matplotlib inline

# Load the Titanic dataset
data = sns.load_dataset('titanic')
data.head()
```

Out[1]:

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

In [2]:

```python
# check the number of missing values in each column
data.isnull().sum().sort_values(ascending=False)
```

Out[2]:

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

We can see that the age column has 177 missing values. Let's replace these missing values with the mean of the column:

In [3]:
```
# impute missing values with mean
data['age'] = data['age'].fillna(data['age'].mean())

# check the number of missing values in each column
data.isnull().sum().sort_values(ascending=False)
```

Out[3]:
```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

We can see that the missing values in the age column have been replaced with the mean of the column.

1.1.2. Median Imputation

Let's load the dataset and replace the missing values in the age column with the median of the column:

In [4]:
```
df = sns.load_dataset('titanic')
df.isnull().sum().sort_values(ascending=False)
```

Out[4]:
```
deck           688
age            177
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

In [5]:
```
# impute missing values with median
df['age'] = df['age'].fillna(df['age'].median())

# check the number of missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[5]:
```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

1.2. Mode Imputation

Mode imputation replaces missing values with the mode (most frequent value) of the column. This is useful for imputing categorical columns, such as Embarked and embark_town in the Titanic dataset.

Let's see how to implement mode imputation in Python using the Titanic dataset.

In [6]:
```
# load the dataset
df = sns.load_dataset('titanic')

# check the number of missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[6]:
```
deck           688
age            177
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

In [7]:
```
# impute missing values with mode
df['embark_town'] = df['embark_town'].fillna(df['embark_town'].mode()[0])
df['embarked'] = df['embarked'].fillna(df['embarked'].mode()[0])

# check the number of missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[7]:
```
deck           688
age            177
survived         0
pclass           0
sex              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
```

We can see that the missing values in the embark_town column and embarked column have been replaced with the mode of the column.

2. K-Nearest Neighbors (KNN)

KNN is a machine learning algorithm that can be used for imputing missing values. It works by finding the most similar data points to the one with the missing value based on other available features. The missing value is then imputed with the mean or median of the most similar data points.

Let's see how to implement KNN imputation in Python using the Titanic dataset.

In [8]:
```
# load the dataset
df = sns.load_dataset('titanic')

# check the number of missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[8]:
```
deck           688
age            177
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

In [9]:
```
# impute missing values with KNN imputer
from sklearn.impute import KNNImputer

# call the KNN class with number of neighbors = 4
imputer = KNNImputer(n_neighbors=4)

#impute missing values with KNN imputer
df['age'] = imputer.fit_transform(df[['age']])

# check the number of missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[9]:
```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

3. Regression Imputation

Regression imputation uses a regression model to predict the missing values based on other variables in the dataset. It works well for both categorical and numerical data.

Let's see how to implement regression imputation in Python using the Titanic dataset.

In [10]:
```python
# load the dataset
df = sns.load_dataset('titanic')

# check the number of missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[10]:
```
deck           688
age            177
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

In [11]:
```
df.head()
```

Out[11]:

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

In [12]:
```
# impute missing values with regression imputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# call the IterativeImputer class with max_iter = 10
imputer = IterativeImputer(max_iter=10)

#impute missing values with regression imputer
df['age'] = imputer.fit_transform(df[['age']])
```

```
# check the number of missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[12]:
```
deck           688
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
age              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

4. Random Forests for Imputing Missing Values

Random forests can handle missing values inherently. They can also be used to predict missing values based on the patterns learned from the other data.

Let's see how to implement random forests in Python using the Titanic dataset.

In [13]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error, mean_absolute_percentage_error
from sklearn.impute import SimpleImputer

# 1. Load the dataset
df = sns.load_dataset('titanic')
```

```
# check missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[13]:

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

We will remove the deck column from the dataset because it has too many missing values:

In [14]:

```
# remove deck column
df.drop('deck', axis=1, inplace=True)

# check missing values in each column
df.isnull().sum().sort_values(ascending=False)
```

Out[14]:
```
age            177
embarked         2
embark_town      2
survived         0
pclass           0
sex              0
sibsp            0
parch            0
fare             0
class            0
who              0
adult_male       0
alive            0
alone            0
dtype: int64
```

We will encode the data at this stage:

In [15]:
```
df.head()
```

Out[15]:

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

In [16]:
```
# encode the data using label encoding
from sklearn.preprocessing import LabelEncoder
# Columns to encode
columns_to_encode = ['sex', 'embarked', 'who', 'class', 'embark_town', 'alive']

# Dictionary to store LabelEncoders for each column
label_encoders = {}

# Loop to apply LabelEncoder to each column
for col in columns_to_encode:
```

```python
    # Create a new LabelEncoder for the column
    le = LabelEncoder()

    # Fit and transform the data, then inverse transform it
    df[col] = le.fit_transform(df[col])

    # Store the encoder in the dictionary
    label_encoders[col] = le

# Check the first few rows of the DataFrame
df.head()
```

Out[16]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone
0	0	3	1	22.0	1	0	7.2500	2	2	1	True	2	0	False
1	1	1	0	38.0	1	0	71.2833	0	0	2	False	0	1	False
2	1	3	0	26.0	0	0	7.9250	2	2	2	False	2	1	True
3	1	1	0	35.0	1	0	53.1000	2	0	2	False	2	1	False
4	0	3	1	35.0	0	0	8.0500	2	2	1	True	2	0	True

We have to first impute the missing values in the age column before we can use it to predict the missing values in the embarked and emark_town columns.

In [17]:
```python
# Split the dataset into two parts: one with missing values, one without
df_with_missing = df[df['age'].isna()]
# dropna removes all rows with missing values
df_without_missing = df.dropna()
```

Let's see the shape of the datasets with and without the missing values:

In [18]:
```python
print("The shape of the original dataset is: ", df.shape)
print("::::::::::::::::::::::::::::::::::::::::::")
print("The shape of the dataset with missing values removed is: ", df_without_missing.shape)
print("::::::::::::::::::::::::::::::::::::::::::")
print("The shape of the dataset with missing values is: ", df_with_missing.shape)
```

```
The shape of the original dataset is:  (891, 14)
::::::::::::::::::::::::::::::::::::::::::
The shape of the dataset with missing values removed is:  (714, 14)
::::::::::::::::::::::::::::::::::::::::::
The shape of the dataset with missing values is:  (177, 14)
```

In [19]:
```
df_with_missing.head()
```

Out[19]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone
5	0	3	1	NaN	0	0	8.4583	1	2	1	True	1	0	True
17	1	2	1	NaN	0	0	13.0000	2	1	1	True	2	1	True
19	1	3	0	NaN	0	0	7.2250	0	2	2	False	0	1	True
26	0	3	1	NaN	0	0	7.2250	0	2	1	True	0	0	True
28	1	3	0	NaN	0	0	7.8792	1	2	2	False	1	1	True

let's see the first five rows of the dataset without the missing values:

In [20]:
```
df_without_missing.head()
```

Out[20]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone
0	0	3	1	22.0	1	0	7.2500	2	2	1	True	2	0	False
1	1	1	0	38.0	1	0	71.2833	0	0	2	False	0	1	False
2	1	3	0	26.0	0	0	7.9250	2	2	2	False	2	1	True
3	1	1	0	35.0	1	0	53.1000	2	0	2	False	2	1	False
4	0	3	1	35.0	0	0	8.0500	2	2	1	True	2	0	True

Let's see the names of all the columns in the dataset:

In [21]:
```
# check the names of the columns
print(df.columns)
```

```
Index(['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare',
       'embarked', 'class', 'who', 'adult_male', 'embark_town', 'alive',
       'alone'],
      dtype='object')
```

In [22]:
```
# Regression Imputation

# split the data into X and y and we will only take the columns with no missing values
X = df_without_missing.drop(['age'], axis=1)
y = df_without_missing['age']

# split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

# Random Forest Imputation
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)

# evaluate the model
y_pred = rf_model.predict(X_test)
print("RMSE for Random Forest Imputation: ", np.sqrt(mean_squared_error(y_test, y_pred)))
print(":::::::::::::::::::::::::::::::::::::::::")
print("R2 Score for Random Forest Imputation: ", r2_score(y_test, y_pred))
print(":::::::::::::::::::::::::::::::::::::::::")
print("MAE for Random Forest Imputation: ", mean_absolute_error(y_test, y_pred))
print(":::::::::::::::::::::::::::::::::::::::::")
print("MAPE for Random Forest Imputation: ", mean_absolute_percentage_error(y_test, y_pred))
```

```
RMSE for Random Forest Imputation:  11.081260589808045
:::::::::::::::::::::::::::::::::::::::::
R2 Score for Random Forest Imputation:  0.33769388288226154
:::::::::::::::::::::::::::::::::::::::::
MAE for Random Forest Imputation:  8.666661815622195
:::::::::::::::::::::::::::::::::::::::::
MAPE for Random Forest Imputation:  0.40839466096086574
```

In [23]:
```
# check the number of missing values in each column
df_with_missing.isnull().sum().sort_values(ascending=False)
```

Out[23]:
```
age            177
survived         0
pclass           0
sex              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
```

In [24]:
```
# Predict missing values
y_pred = rf_model.predict(df_with_missing.drop(['age'], axis=1))
```

In [25]:
```
# remove warning
import warnings
warnings.filterwarnings('ignore')

# replace the missing values with the predicted values
df_with_missing['age'] = y_pred

# check the missing values
df_with_missing.isnull().sum().sort_values(ascending=False)
```

Out[25]:
```
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
```

In [26]:
```python
# concatenate the two dataframes
df_complete = pd.concat([df_with_missing, df_without_missing], axis=0)
# print the shape of the complete dataframe
print("The shape of the complete dataframe is: ", df_complete.shape)

#check the first 5 rows of the complete dataframe
df_complete.head()
```

```
The shape of the complete dataframe is:  (891, 14)
```

Out[26]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alone
5	0	3	1	32.976583	0	0	8.4583	1	2	1	True	1	0	True
17	1	2	1	35.642218	0	0	13.0000	2	1	1	True	2	1	True
19	1	3	0	18.347000	0	0	7.2250	0	2	2	False	0	1	True
26	0	3	1	35.571486	0	0	7.2250	0	2	1	True	0	0	True
28	1	3	0	20.651429	0	0	7.8792	1	2	2	False	1	1	True

In [27]:
```python
for col in columns_to_encode:
    # Retrieve the corresponding LabelEncoder for the column
    le = label_encoders[col]

    # Inverse transform the data
```

```
    df_complete[col] = le.inverse_transform(df[col])

# check the first 5 rows of the complete dataframe
df_complete.head()
```

Out[27]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	embark_town	alive	alon
5	0	3	male	32.976583	0	0	8.4583	S	Third	man	True	Southampton	no	Tru
17	1	2	female	35.642218	0	0	13.0000	C	First	woman	True	Cherbourg	yes	Tru
19	1	3	female	18.347000	0	0	7.2250	S	Third	woman	False	Southampton	yes	Tru
26	0	3	female	35.571486	0	0	7.2250	S	First	woman	True	Southampton	yes	Tru
28	1	3	male	20.651429	0	0	7.8792	S	Third	man	False	Southampton	no	Tru

In [28]:

```
# print the shape of the complete dataframe
print("The shape of the complete dataframe is: ", df_complete.shape)
```

```
The shape of the complete dataframe is:  (891, 14)
```

In [29]:

```
# check the number of missing values in each column
df_complete.isnull().sum().sort_values(ascending=False)
```

Out[29]:

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

5. Advanced Techniques

5.1. Multiple Imputation by Chained Equations (MICE)

Multiple Imputation by Chained Equations (MICE) is a more sophisticated technique that models each variable with missing values as a function of other variables in a round-robin fashion. It works well for both categorical and numerical data.

To demonstrate Multiple Imputation by Chained Equations (MICE) in Python, we can use the IterativeImputer class from the sklearn.impute module. MICE is a sophisticated method of imputation that models each feature with missing values as a function of other features, and it uses that estimate for imputation. It does this in a round-robin fashion: each feature is modeled in turn. The MICE algorithm is implemented in the IterativeImputer class.

Let's see how to implement MICE in Python using the Titanic dataset.

In [30]:
```
# imoprt libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# laod the dataset
df = sns.load_dataset('titanic')
df.head()
```

Out[30]:

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

In [31]:

```
# check the missing values
df.isnull().sum().sort_values(ascending=False)
```

Out[31]:

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

In [32]:

```
df.head()
```

Out[32]:

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

In [33]:

```
from sklearn.preprocessing import LabelEncoder

# create a LabelEncoder object using LabelEncoder() in for loop for categorical columns
# Columns to encode
columns_to_encode = ['sex', 'embarked', 'who', 'deck', 'class', 'embark_town', 'alive']

# Dictionary to store LabelEncoders for each column
label_encoders = {}

# Loop to apply LabelEncoder to each column for encoding
for col in columns_to_encode:
    # Create a new LabelEncoder for the column
    le = LabelEncoder()
    # Fit and transform the data
    df[col] = le.fit_transform(df[col])
    # Store the encoder in the dictionary
    label_encoders[col] = le
df.head()
```

Out[33]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	1	22.0	1	0	7.2500	2	2	1	True	7	2	0	False
1	1	1	0	38.0	1	0	71.2833	0	0	2	False	2	0	1	False
2	1	3	0	26.0	0	0	7.9250	2	2	2	False	7	2	1	True
3	1	1	0	35.0	1	0	53.1000	2	0	2	False	2	2	1	False
4	0	3	1	35.0	0	0	8.0500	2	2	1	True	7	2	0	True

In [34]:

```
# impute the missing values with IterativeImputer
# call the IterativeImputer class with max_iter = 10
imputer = IterativeImputer(max_iter=10)

#impute missing values using IterativeImputer in a for loop for age, embark_town,embarked columns and deck

# Columns to impute
columns_to_impute = ['age', 'embark_town', 'embarked', 'deck']

# Loop to impute each column
for col in columns_to_impute:
    df[col] = imputer.fit_transform(df[[col]])
# check the missing values
df.isnull().sum().sort_values(ascending=False)
```

Out[34]:
```
survived       0
pclass         0
sex            0
age            0
sibsp          0
parch          0
fare           0
embarked       0
class          0
who            0
adult_male     0
deck           0
embark_town    0
alive          0
alone          0
dtype: int64
```

In [35]:
```
df.head()
```

Out[35]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	1	22.0	1	0	7.2500	2.0	2	1	True	7.0	2.0	0	False
1	1	1	0	38.0	1	0	71.2833	0.0	0	2	False	2.0	0.0	1	False
2	1	3	0	26.0	0	0	7.9250	2.0	2	2	False	7.0	2.0	1	True
3	1	1	0	35.0	1	0	53.1000	2.0	0	2	False	2.0	2.0	1	False
4	0	3	1	35.0	0	0	8.0500	2.0	2	1	True	7.0	2.0	0	True

In [36]:
```
# Inverse transform for encoded columns
for col in columns_to_encode:
    # Retrieve the corresponding LabelEncoder for the column
    le = label_encoders[col]
    # Inverse transform the data and convert to integer type
    df[col] = le.inverse_transform(df[col].astype(int))

df.head()
```

Out[36]:

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

5.2. Deep Learning Methods

Neural networks, especially autoencoders, can be effective in imputing missing values in complex datasets. Deep learning methods, particularly neural networks like autoencoders, offer a powerful approach for imputing missing values in complex datasets. These methods are especially useful when the data has intricate, non-linear relationships that traditional statistical methods might not capture effectively.

Understanding Autoencoders for Imputation:

1:- What is an Autoencoder ?

An autoencoder is a type of neural network that is trained to copy its input to its output.

It has a hidden layer that describes a code used to represent the input.

The network may be viewed as consisting of two parts: an encoder function, which compresses the input into a latent-space representation, and a decoder

function, which reconstructs the input from the latent space.

2:- How Autoencoders Work for Imputation:

The key idea is to train the autoencoder to ignore the noise (missing values) in the input data.

During training, inputs with missing values are presented, and the network learns to predict the missing values in a way that minimizes reconstruction error for known parts of the data.

This results in the network learning a robust representation of the data, enabling it to make reasonable guesses about missing values.

3:- Advantages of Using Autoencoders:

Handling Complex Patterns:

They can capture non-linear relationships in the data, which is particularly useful for complex datasets.

Scalability:

They can handle large-scale datasets efficiently.

Flexibility:

They can be adapted to different types of data (e.g., images, text, time-series).

4:- Implementation Considerations:

Data Preprocessing:

Data should be normalized or standardized before feeding it into an autoencoder.

Network Architecture:

The choice of architecture (number of layers, type of layers, etc.) depends on the complexity of the data.

Training Process:

It might involve techniques like dropout or noise addition to improve the model's ability to handle missing data.

5:- Example Use-Cases:

Image Data:

Filling in missing pixels or reconstructing corrupted images.

Time-Series Data:

Imputing missing values in sequences like stock prices or weather data.

Tabular Data:

Handling missing entries in datasets used for machine learning.

Implementation Example:

Here's a simplified example of how you might set up an autoencoder for imputation in Python using TensorFlow and Keras: (Check the next notebook)

In [ ]:

```

```