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 [1]:
```
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

# Load the Titanic dataset
df_titanic = sns.load_dataset('titanic')

# Selecting relevant features for simplicity
df_titanic = df_titanic[['survived', 'pclass', 'sex', 'age', 'sibsp', 'parch', 'fare', 'embarked']]

# Preprocessing
# Separate features and target
X = df_titanic.drop('survived', axis=1)
y = df_titanic['survived']

# Handling missing values and categorical variables
numeric_features = ['age', 'fare', 'sibsp', 'parch']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', MinMaxScaler())])

categorical_features = ['pclass', 'sex', 'embarked']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
```

```
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])

# ColumnTransformer for preprocessing
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])

# Preprocessing the dataset
X_preprocessed = preprocessor.fit_transform(X)

# Splitting the dataset (we'll use the train set to train the autoencoder)
X_train, X_test, y_train, y_test = train_test_split(X_preprocessed, y, test_size=0.2, random_state=42)

# Define the autoencoder architecture
input_dim = X_train.shape[1]
encoding_dim = 32

input_layer = Input(shape=(input_dim,))
encoded = Dense(encoding_dim, activation='relu')(input_layer)
decoded = Dense(input_dim, activation='sigmoid')(encoded)

autoencoder = Model(input_layer, decoded)
autoencoder.compile(optimizer='adam', loss='mean_squared_error')

# Train the autoencoder
autoencoder.fit(X_train, X_train, epochs=50, batch_size=256, shuffle=True, validation_split=0.2)

# Using the autoencoder for imputation on test set
X_test_imputed = autoencoder.predict(X_test)

# Note: Transforming imputed data back to original feature space is complex and requires reversing the preprocessing
# This is often not straightforward, especially for one-hot encoded features.
```

```
Epoch 1/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 5s 283ms/step - loss: 0.2474 - val_loss: 0.2440
Epoch 2/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 31ms/step - loss: 0.2425 - val_loss: 0.2388
Epoch 3/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.2364 - val_loss: 0.2337
Epoch 4/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - loss: 0.2321 - val_loss: 0.2288
Epoch 5/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.2271 - val_loss: 0.2240
Epoch 6/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - loss: 0.2228 - val_loss: 0.2193
Epoch 7/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 0.2173 - val_loss: 0.2146
Epoch 8/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.2131 - val_loss: 0.2100
Epoch 9/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step - loss: 0.2085 - val_loss: 0.2054
Epoch 10/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 0.2041 - val_loss: 0.2009
Epoch 11/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 0.2000 - val_loss: 0.1964
Epoch 12/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 45ms/step - loss: 0.1950 - val_loss: 0.1919
Epoch 13/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - loss: 0.1907 - val_loss: 0.1875
Epoch 14/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - loss: 0.1863 - val_loss: 0.1831
Epoch 15/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.1822 - val_loss: 0.1788
Epoch 16/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.1782 - val_loss: 0.1744
Epoch 17/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 0.1738 - val_loss: 0.1701
Epoch 18/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - loss: 0.1691 - val_loss: 0.1657
Epoch 19/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.1645 - val_loss: 0.1614
Epoch 20/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.1608 - val_loss: 0.1571
Epoch 21/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - loss: 0.1566 - val_loss: 0.1529
```

```
Epoch 22/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - loss: 0.1523 - val_loss: 0.1488
Epoch 23/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 0.1488 - val_loss: 0.1447
Epoch 24/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 0.1446 - val_loss: 0.1407
Epoch 25/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 0.1409 - val_loss: 0.1368
Epoch 26/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 0.1365 - val_loss: 0.1329
Epoch 27/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.1333 - val_loss: 0.1291
Epoch 28/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step - loss: 0.1293 - val_loss: 0.1254
Epoch 29/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 0.1252 - val_loss: 0.1217
Epoch 30/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 0.1215 - val_loss: 0.1181
Epoch 31/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 46ms/step - loss: 0.1184 - val_loss: 0.1147
Epoch 32/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 35ms/step - loss: 0.1155 - val_loss: 0.1113
Epoch 33/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 30ms/step - loss: 0.1116 - val_loss: 0.1081
Epoch 34/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.1086 - val_loss: 0.1049
Epoch 35/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step - loss: 0.1057 - val_loss: 0.1019
Epoch 36/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 44ms/step - loss: 0.1029 - val_loss: 0.0990
Epoch 37/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - loss: 0.1002 - val_loss: 0.0962
Epoch 38/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - loss: 0.0973 - val_loss: 0.0936
Epoch 39/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 0.0940 - val_loss: 0.0910
Epoch 40/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 43ms/step - loss: 0.0920 - val_loss: 0.0885
Epoch 41/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 49ms/step - loss: 0.0904 - val_loss: 0.0862
Epoch 42/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 0.0882 - val_loss: 0.0839
```

```
Epoch 43/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 37ms/step - loss: 0.0856 - val_loss: 0.0818
Epoch 44/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step - loss: 0.0835 - val_loss: 0.0797
Epoch 45/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 38ms/step - loss: 0.0813 - val_loss: 0.0777
Epoch 46/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 39ms/step - loss: 0.0791 - val_loss: 0.0758
Epoch 47/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 0.0771 - val_loss: 0.0740
Epoch 48/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 28ms/step - loss: 0.0759 - val_loss: 0.0722
Epoch 49/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 33ms/step - loss: 0.0735 - val_loss: 0.0705
Epoch 50/50
3/3 ━━━━━━━━━━━━━━━━━━━━ 0s 41ms/step - loss: 0.0723 - val_loss: 0.0688
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 21ms/step
```

In [ ]: