Data Preparation and Normalisation

Data Preparation and Scaling Techniques

---

1. Overview of the Code

This Python code performs data preprocessing steps on a dataset for a machine learning project. It includes loading a dataset, handling missing values, encoding categorical variables, splitting the data into training and testing sets, scaling numerical features, and visualizing data distributions before and after scaling.

2. Libraries Used

- **pandas**: For data manipulation and analysis.
- **numpy**: Provides support for numerical operations.
- **sklearn**: A machine learning library used for model selection, preprocessing, and scaling.
    - **train_test_split**: Splits the data into training and testing sets.
    - **StandardScaler**: Used for feature scaling (standardization).
    - **LabelEncoder**: Converts categorical data to numeric values.
- **seaborn**: For statistical data visualization.
- **matplotlib**: For creating plots and visualizing data.
- **style**: A module from matplotlib used to set the style of plots.

3. Step-by-Step Explanation of the Code

3.1 Load the Dataset

df = pd.read_csv('Dataset (ATS).csv')

The dataset is read from a CSV file using pandas. The dataset presumably contains customer information and features like gender, dependents, phone service, internet service, contract type, and a target variable (Churn).

3.2 Display Basic Information About the Dataset

```
print(df.info())
print(df.describe())
```

- **info()**: Displays the structure of the dataset, including column names, data types, and the number of non-null entries.
- **describe()**: Provides summary statistics for numerical columns (e.g., mean, standard deviation, min, and max values).

3.3 Data Cleaning: Handling Missing Values

```
print(df.isnull().sum())
df.fillna(method='ffill', inplace=True)
print(df.isnull().sum())
```

- **isnull().sum()**: Identifies missing values in each column.
- **fillna(method='ffill')**: Fills missing values using the forward fill technique, which propagates the last valid observation forward to fill gaps.
- **inplace=True**: Modifies the dataframe in place, meaning no new dataframe is created.

3.4 Encode Categorical Variables

```
categorical_cols = ['gender', 'Dependents', 'PhoneService', 'MultipleLines',
'InternetService', 'Contract', 'Churn']
for col in categorical_cols:
    le = LabelEncoder()
    df[col] = le.fit_transform(df[col])
```

- **Label Encoding**: Converts categorical variables into numeric values. For example, "Male" and "Female" in the gender column might be encoded as 0 and 1, respectively.
- This technique is necessary because machine learning models often work better with numeric data than categorical data.

3.5 Split the Data Into Training and Testing Sets

```
X = df.drop('Churn', axis=1)
y = df['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **Feature and Target Separation**: X contains all the features (independent variables), and y contains the target variable (Churn), which we aim to predict.
- **train_test_split()**: Splits the dataset into training (80%) and testing (20%) sets. This ensures that the model is trained on one portion of the data and evaluated on another to avoid overfitting.

4. Emphasis on Scaling Techniques

4.1 Standard Scaling

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- **Scaling**: In this step, numerical features are normalized using **StandardScaler**, a method that ensures that the data has **mean of 0** and a **standard deviation of 1.**
- **Why Scaling is Important**:
    - Some machine learning algorithms (like SVM, K-Means, and neural networks) perform better when the data is scaled because they are sensitive to the magnitude of features.
    - Scaling prevents features with large values from dominating those with smaller values.

4.1.1 How StandardScaler Works

- **StandardScaler** transforms the data by:

$$Z = \frac{(X - \mu)}{\sigma}$$

where:

- **X** is the feature,
- **μ** is the mean of the feature values in the training data,
- **σ** is the standard deviation.

- The scaler first **fits** the X_train data (calculates the mean and standard deviation), and then applies the transformation (fit_transform).
- For X_test, the scaler uses the same mean and standard deviation values from X_train (using transform), ensuring consistent scaling between training and testing sets.

4.2 Visualization of Distributions Before and After Scaling

```
style.use('seaborn-whitegrid')
plt.figure(figsize=(14, 6))

plt.subplot(1, 2, 1)
sns.histplot(df['MonthlyCharges'], kde=True, color='skyblue')
plt.title('MonthlyCharges Distribution Before Scaling')
plt.xlabel('MonthlyCharges')

plt.subplot(1, 2, 2)
sns.histplot(X_train_scaled[:, X.columns.get_loc('MonthlyCharges')], kde=True,
color='salmon')
plt.title('MonthlyCharges Distribution After Scaling')
plt.xlabel('MonthlyCharges (scaled)')

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

- This section visualizes the distribution of one specific feature (MonthlyCharges) before and after scaling.
- **Before Scaling**: The distribution of MonthlyCharges is shown as it appears in the original dataset. The values may have a wide range.
- **After Scaling**: The same feature (MonthlyCharges) is plotted after scaling. As expected, the distribution is now centered around 0, with values standardized around the mean of 0 and standard deviation of 1.

5. Summary

This code snippet covers several key steps in preparing a dataset for machine learning, with a particular focus on scaling techniques. The following preprocessing steps are performed:

- **Data Loading**: Reads the dataset into a DataFrame using pandas.
- **Data Cleaning**: Handles missing values by forward filling.
- **Label Encoding**: Converts categorical variables to numeric form using LabelEncoder.
- **Train-Test Split**: Splits the dataset into training and testing sets.
- **Scaling**: Uses StandardScaler to normalize numerical features, ensuring that all features have a mean of 0 and a standard deviation of 1.

The **scaling** step is essential for many machine learning algorithms because it ensures that features are on the same scale, improving performance and stability of the models. By visualizing data before and after scaling, you can clearly see the effect of normalization on feature distributions.