Perform data preprocessing tasks like normalization and encoding using Pandas

Table of Contents

Description

Data preprocessing is a crucial phase in the data analysis and machine learning pipeline. It involves cleaning, transforming, and organizing raw data into a structured format that is suitable for analysis or modeling. Proper preprocessing enhances data quality, reduces noise, and can significantly improve the performance of machine learning models. This lab guide focuses on using the Pandas library in Python to carry out essential preprocessing tasks, particularly normalization and encoding.

Problem Statement

Raw datasets often contain inconsistencies, missing values, and varying formats, making effective analysis and modeling challenging. This guide focuses on using Pandas in Python to preprocess data by handling missing values, normalizing numerical features, and encoding categorical variables. The objective is to transform the dataset into a clean, structured format that enhances data quality and improves model performance.

Prerequisites

Completion of all previous lab guides (up to Lab Guide-03) is required before proceeding with Lab Guide-04.

Software Requirements

- **Python**: Version 3.11.9
- **VSCode**: Visual Studio Code editor

Hardware Requirements

- Minimum 4GB RAM.
- At least 1GB of free disk space for storing datasets.

---

4. Setup Instructions

Setting Up a Python Environment

1. Install Python from official Python website.

- Locate a reliable version of Python 3, **"Download Python 3.11.9"**.
- Choose the correct link for your device from the options provided: either Windows installer (64-bit) or Windows installer (32-bit) and proceed to download the executable file.

2. Install required packages using pip:

```
pip install pandas matplotlib scikit-learn
```

```
PS C:\Users\Administrator\Desktop\AIML> pip install pandas numpy matplotlib seaborn scikit-learn
Collecting pandas
  Using cached pandas-2.2.3-cp311-cp311-win_amd64.whl.metadata (19 kB)
Collecting numpy
  Using cached numpy-2.1.2-cp311-cp311-win_amd64.whl.metadata (59 kB)
Collecting matplotlib
  Using cached matplotlib-3.9.2-cp311-cp311-win_amd64.whl.metadata (11 kB)
Collecting seaborn
  Using cached seaborn-0.13.2-py3-none-any.whl.metadata (5.4 kB)
Collecting scikit-learn
  Using cached scikit_learn-1.5.2-cp311-cp311-win_amd64.whl.metadata (13 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\program files\python311\lib\site-packages (from pandas) (2.9.0.post0
)
Requirement already satisfied: pytz>=2020.1 in c:\program files\python311\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in c:\program files\python311\lib\site-packages (from pandas) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\program files\python311\lib\site-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in c:\program files\python311\lib\site-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in c:\program files\python311\lib\site-packages (from matplotlib) (4.54.1)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\program files\python311\lib\site-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in c:\program files\python311\lib\site-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=8 in c:\program files\python311\lib\site-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\program files\python311\lib\site-packages (from matplotlib) (3.2.0)
Requirement already satisfied: scipy>=1.6.0 in c:\program files\python311\lib\site-packages (from scikit-learn) (1.14.1)
Requirement already satisfied: joblib>=1.2.0 in c:\program files\python311\lib\site-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\program files\python311\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: six>=1.5 in c:\program files\python311\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1
.16.0)
Using cached pandas-2.2.3-cp311-cp311-win_amd64.whl (11.6 MB)
Using cached numpy-2.1.2-cp311-cp311-win_amd64.whl (12.9 MB)
Using cached matplotlib-3.9.2-cp311-cp311-win_amd64.whl (7.8 MB)
Using cached seaborn-0.13.2-py3-none-any.whl (294 kB)
Using cached scikit_learn-1.5.2-cp311-cp311-win_amd64.whl (11.0 MB)
Installing collected packages: numpy, pandas, scikit-learn, matplotlib, seaborn
Successfully installed matplotlib-3.9.2 numpy-2.1.2 pandas-2.2.3 scikit-learn-1.5.2 seaborn-0.13.2
```

Key Concepts

Here’s a simplified explanation of the key concepts related to data preprocessing:

Normalization

Normalization scales numerical data to a specific range, usually between 0 and 1. This ensures that no single feature dominates the analysis due to larger values. For example, if you have data on house prices and square footage, normalizing helps algorithms treat both features equally. Common methods include Min-Max scaling, where each value is adjusted based on the minimum and maximum of the dataset.

Encoding

Encoding converts categorical variables (like color names or city names) into numerical values so that machine learning algorithms can process them. The two main methods are:

- **One-Hot Encoding**: This creates a new binary column for each category. For instance, if you have a “Color” feature with values like "Red," "Green," and "Blue," it will create three columns: one for each color, with a 1 or 0 indicating presence.
- **Label Encoding**: This assigns an integer to each category. For instance, "Red" could be 0, "Green" 1, and "Blue" 2. This method is simpler but can introduce unintended relationships if the categories do not have a natural order.

Handling Missing Values

Missing values can lead to inaccurate models. There are a few common approaches to deal with them:

- **Imputation**: Filling in missing values with statistical measures like the mean or median. For example, if you have missing age data, you might fill those gaps with the average age of the group.
- **Dropping**: If a feature has too many missing values, it might be better to remove it from the dataset entirely, or drop the rows with missing values if they are not significant.

Exploratory Data Analysis (EDA)

EDA is the process of examining datasets to summarize their main characteristics, often through visualizations. It helps identify patterns, relationships, and potential outliers, guiding further preprocessing steps. For instance, you might use histograms to see how data is distributed or scatter plots to check relationships between variables.

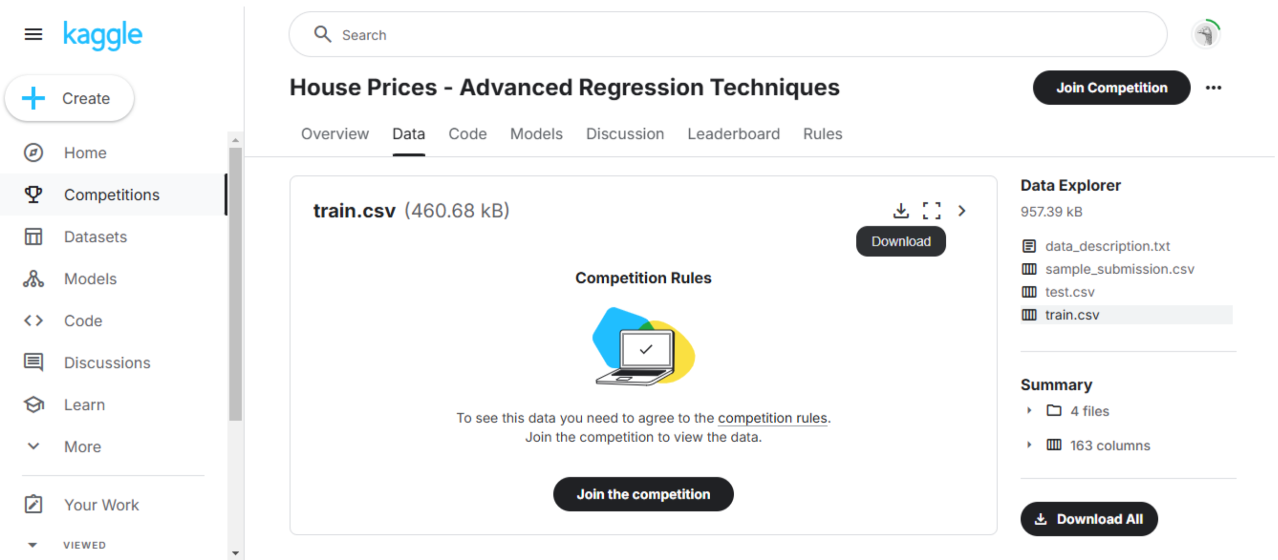
Data Preprocessing Steps

Loading the Dataset

- **Create a new python file**
  - Create a Python file named `data_preprocessing.py`.

Begin by loading the dataset into a Pandas DataFrame. You can download the dataset from Kaggle:

- **Downloading the Dataset**
  - Go to the **Kaggle website** and sign in to your account. If you don't have an account, create one.
  - Navigate to the **House Prices: Advanced Regression Techniques** competition page.



  - Click on the "Data" tab and download the `train.csv` file (the dataset used for training).
  - Move the downloaded `train.csv` file into your project directory.

```python
import pandas as pd

# Load the dataset
data = pd.read_csv('./train.csv')
```

Exploratory Data Analysis (EDA)

Perform exploratory data analysis to understand the dataset.

```python
# Display basic information about the dataset
print(data.info())
print(data.describe())

# Check for missing values
missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values[missing_values > 0])
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python data_preprocessing.py
```

Output

```
PS C:\Users\Administrator\Desktop\AIML> python decision_tree.py
DecisionTreeRegressor(random_state=42)
```

```
PS C:\Users\Administrator\Desktop\AIML> python data_preprocessing.py
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             1460 non-null   int64
 1   MSSubClass     1460 non-null   int64
 2   MSZoning       1460 non-null   object
 3   LotFrontage    1201 non-null   float64
 4   LotArea        1460 non-null   int64
 5   Street         1460 non-null   object
 6   Alley          91 non-null     object
 7   LotShape       1460 non-null   object
 8   LandContour    1460 non-null   object
 9   Utilities      1460 non-null   object
 10  LotConfig      1460 non-null   object
 11  LandSlope      1460 non-null   object
 12  Neighborhood   1460 non-null   object
 13  Condition1     1460 non-null   object
 14  Condition2     1460 non-null   object
 15  BldgType       1460 non-null   object
 16  HouseStyle     1460 non-null   object
 17  OverallQual    1460 non-null   int64
 18  OverallCond    1460 non-null   int64
 19  YearBuilt      1460 non-null   int64
 20  YearRemodAdd   1460 non-null   int64
 21  RoofStyle      1460 non-null   object
 22  RoofMatl       1460 non-null   object
 23  Exterior1st    1460 non-null   object
 24  Exterior2nd    1460 non-null   object
 25  MasVnrType     588 non-null    object
 26  MasVnrArea     1452 non-null   float64
 27  ExterQual      1460 non-null   object
```

Handling Missing Values

Address missing values appropriately, as they can significantly impact model performance.

- **Methods to handle missing values:**
  - **Mean/Median Imputation:** Replace missing values with the mean or median of the column.
  - **Mode Imputation:** For categorical variables, use the mode.
  - **Dropping Missing Values:** If the percentage of missing data is significant, consider dropping the column or row.

```
# Fill missing values (example: mean imputation for LotFrontage)
data['LotFrontage'].fillna(data['LotFrontage'].mean(), inplace=True)

# For categorical features, fill with mode
data['MasVnrType'].fillna(data['MasVnrType'].mode()[0], inplace=True)
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python data_preprocessing.py
```

Output

```
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
None
                Id   MSSubClass  LotFrontage        LotArea  ...      MiscVal       MoSold       YrSold      SalePrice
count  1460.000000  1460.000000  1201.000000    1460.000000  ...  1460.000000  1460.000000  1460.000000    1460.000000
mean    730.500000    56.897260    70.049958   10516.828082  ...    43.489041     6.321918  2007.815753  180921.195890
std     421.610009    42.300571    24.284752    9981.264932  ...   496.123024     2.703626     1.328095   79442.502883
min       1.000000    20.000000    21.000000    1300.000000  ...     0.000000     1.000000  2006.000000   34900.000000
25%     365.750000    20.000000    59.000000    7553.500000  ...     0.000000     5.000000  2007.000000  129975.000000
50%     730.500000    50.000000    69.000000    9478.500000  ...     0.000000     6.000000  2008.000000  163000.000000
75%    1095.250000    70.000000    80.000000   11601.500000  ...     0.000000     8.000000  2009.000000  214000.000000
max    1460.000000   190.000000   313.000000  215245.000000  ... 15500.000000    12.000000  2010.000000  755000.000000

[8 rows x 38 columns]
Missing Values:
 LotFrontage      259
Alley           1369
MasVnrType       872
MasVnrArea         8
BsmtQual          37
BsmtCond          37
BsmtExposure      38
BsmtFinType1      37
BsmtFinType2      38
Electrical         1
FireplaceQu      690
GarageType        81
GarageYrBlt       81
GarageFinish      81
GarageQual        81
GarageCond        81
PoolQC          1453
Fence           1179
MiscFeature     1406
```

Visualizing Missing Values

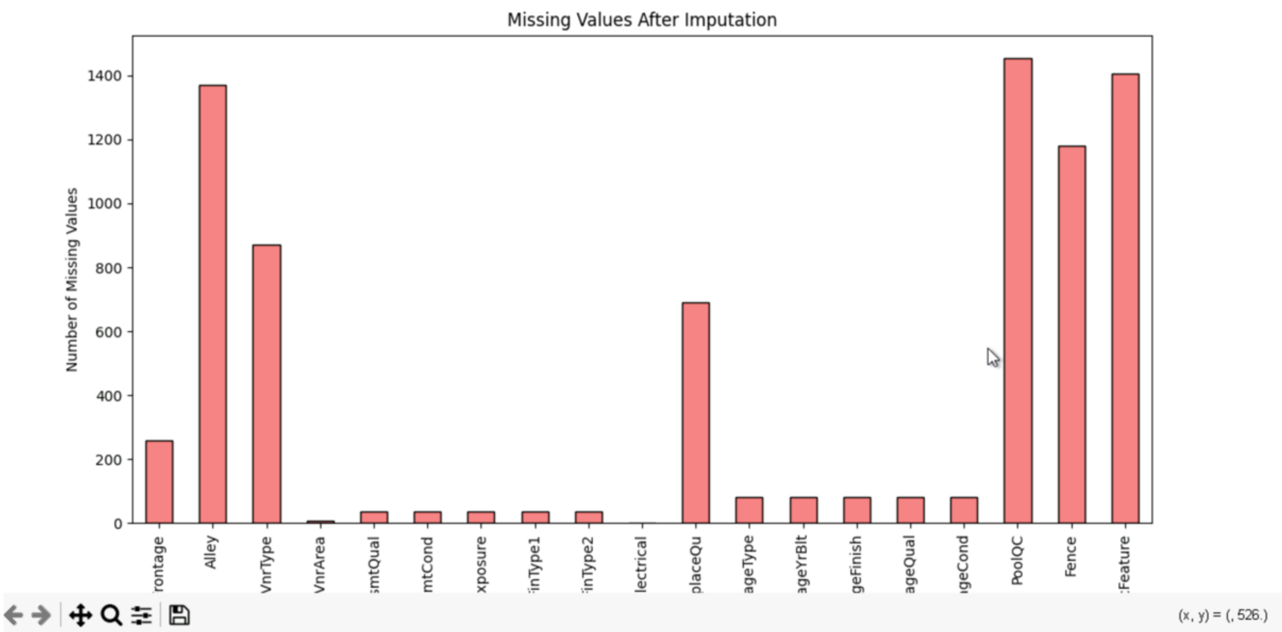
```python
import matplotlib.pyplot as plt

# Create a bar plot for missing values before handling
plt.figure(figsize=(10, 6))
missing_values[missing_values > 0].plot(kind='bar', color='skyblue',
edgecolor='black')
plt.title('Missing Values by Feature')
plt.xlabel('Features')
plt.ylabel('Number of Missing Values')
plt.show()

# Recheck missing values after imputation
missing_values_after = data.isnull().sum()
plt.figure(figsize=(10, 6))
missing_values_after[missing_values_after > 0].plot(kind='bar',
color='lightcoral', edgecolor='black')
plt.title('Missing Values After Imputation')
plt.xlabel('Features')
plt.ylabel('Number of Missing Values')
plt.show()
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python data_preprocessing.py
```

Output



Normalization

Normalize numerical features to a common scale.

```
from sklearn.preprocessing import MinMaxScaler

# Normalize numerical features
scaler = MinMaxScaler()
data[['LotFrontage']] = scaler.fit_transform(data[['LotFrontage']])
```

Visualizing Normalized Features:

```
# Plotting before and after normalization
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.hist(data['LotFrontage'], bins=30, color='skyblue', edgecolor='black')
plt.title('LotFrontage Distribution (Before Normalization)')
plt.xlabel('LotFrontage')
plt.ylabel('Frequency')
```

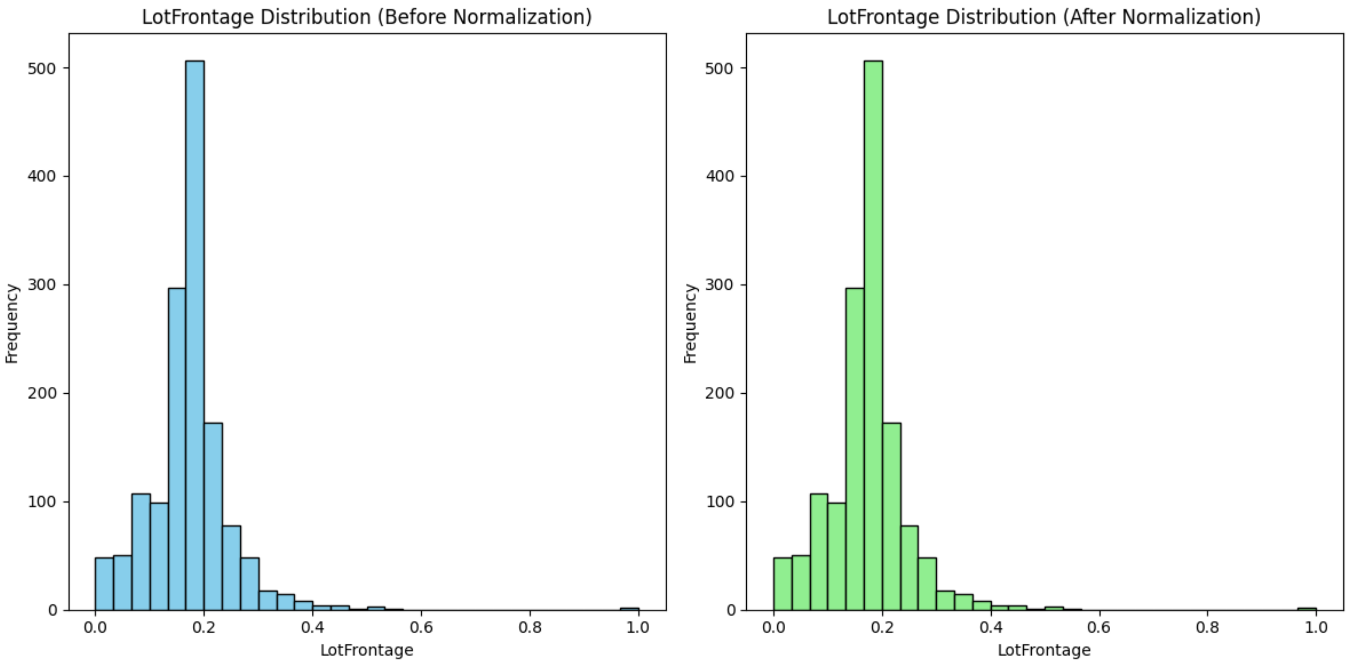
```
plt.subplot(1, 2, 2)
plt.hist(data['LotFrontage'], bins=30, color='lightgreen', edgecolor='black')
plt.title('LotFrontage Distribution (After Normalization)')
plt.xlabel('LotFrontage')
plt.ylabel('Frequency')

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

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python data_preprocessing.py
```

Output



Encoding Categorical Variables

Transform categorical variables into a numerical format using encoding techniques.

```
# Encoding categorical variables
data = pd.get_dummies(data, columns=['Neighborhood'], drop_first=True)
```

Visualizing Encoded Features:

```
# Check the column names
print("Columns in the dataset:")
print(data.columns)

# Strip whitespace from headers (if needed)
data.columns = data.columns.str.strip()

# Check if 'Neighborhood' column exists
if 'Neighborhood' in data.columns:
    # Visualizing the distribution of the 'Neighborhood' feature
    plt.figure(figsize=(10, 6))
    data['Neighborhood'].value_counts().plot(kind='bar', color='salmon',
edgecolor='black')
    plt.title('Distribution of Neighborhoods')
    plt.xlabel('Neighborhood')
    plt.ylabel('Count')
    plt.xticks(rotation=45)
    plt.show()
else:
    print("The 'Neighborhood' column does not exist in the dataset. Available
columns are:")
    print(data.columns)
```

Run the Python file

- Use the command below in your terminal to run the Python file:

```
python data_preprocessing.py
```

Output

```
PS C:\Users\Administrator\Desktop\AIML> python data_preprocessing.py
Columns in the dataset:
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities',
       ...
       'Neighborhood_NoRidge', 'Neighborhood_NridgHt', 'Neighborhood_OldTown',
       'Neighborhood_SWISU', 'Neighborhood_Sawyer', 'Neighborhood_SawyerW',
       'Neighborhood_Somerst', 'Neighborhood_StoneBr', 'Neighborhood_Timber',
       'Neighborhood_Veenker'],
      dtype='object', length=104)
The 'Neighborhood' column does not exist in the dataset. Available columns are:
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities',
       ...
       'Neighborhood_NoRidge', 'Neighborhood_NridgHt', 'Neighborhood_OldTown',
       'Neighborhood_SWISU', 'Neighborhood_Sawyer', 'Neighborhood_SawyerW',
       'Neighborhood_Somerst', 'Neighborhood_StoneBr', 'Neighborhood_Timber',
       'Neighborhood_Veenker'],
      dtype='object', length=104)
```

References

- Download the dataset: Kaggle House Prices: Advanced Regression Techniques
- Data Preprocessing in Python