**Code understanding :**

This code imports libraries that are commonly used in data analysis, visualization, and machine learning. Here's a breakdown in simple terms:

1. **import numpy as np**
   * Imports the numpy library and gives it the nickname np.
   * Used for numerical calculations, working with arrays, and performing mathematical operations.
2. **import pandas as pd**
   * Imports the pandas library and gives it the nickname pd.
   * Useful for working with data in tables (like Excel spreadsheets), such as loading, cleaning, and analyzing data.
3. **import matplotlib.pyplot as plt**
   * Imports the matplotlib library's plotting module and gives it the nickname plt.
   * Used to create graphs and charts for data visualization.
4. **import seaborn as sns**
   * Imports the seaborn library and gives it the nickname sns.
   * Extends matplotlib by providing prettier and more advanced statistical plots.
5. **import calendar**
   * Imports Python’s built-in calendar module.
   * Used to work with dates and calendar-related tasks, like finding weekdays or month names.
6. **from pandas.api.types import CategoricalDtype**
   * Imports the CategoricalDtype class from pandas.
   * Allows you to define custom categories for your data, often useful for sorting and grouping.
7. **from sklearn.preprocessing import StandardScaler**
   * Imports the StandardScaler class from the sklearn library.
   * Used in machine learning to standardize data (make all features have a mean of 0 and standard deviation of 1), which helps some algorithms perform better.

**What is this code for?**

This code sets up tools for:

* Handling and analyzing data (numpy, pandas).
* Visualizing data (matplotlib, seaborn).
* Working with dates (calendar).
* Preparing data for machine learning (StandardScaler).

This code is related to loading and checking data files for a machine learning project. Here's a simple explanation:

**1. File Paths**

train\_data\_path = r"C:\Users\pc planet\Desktop\AI\House-Price-Prediction-master\House-Price-Prediction-master\ML\_Model\data\_set\train.csv"

test\_data\_path = r"C:\Users\pc planet\Desktop\AI\House-Price-Prediction-master\House-Price-Prediction-master\ML\_Model\data\_set\test.csv"

* These lines define the file paths for two datasets:
  + **train\_data\_path**: Path to the training dataset (used to train the model).
  + **test\_data\_path**: Path to the testing dataset (used to evaluate the model).
* The r before the string indicates a raw string, so backslashes (\) are treated as literal characters.

**2. Load Data into Pandas DataFrames**

df\_train = pd.read\_csv(train\_data\_path)

df\_test = pd.read\_csv(test\_data\_path)

* **pd.read\_csv()**: Loads the data from the CSV files into **dataframes** (df\_train and df\_test) using the Pandas library.
  + A **dataframe** is like a table where each column represents a variable (feature) and each row represents a record (data point).

**3. Check Data Shape**

print(df\_train.shape)

print(df\_test.shape)

**df\_train**: Refers to a pandas DataFrame.

**.shape**: Returns a tuple representing the dimensions of the DataFrame.

* The first value in the tuple is the number of rows.
* The second value is the number of columns.
* **df\_train.shape**: Prints the number of rows and columns in the training dataset.
  + Example: (1000, 10) means 1000 rows and 10 columns.
* **df\_test.shape**: Prints the number of rows and columns in the testing dataset.
  + This gives an idea of the dataset size and structure.

**Why is this done?**

1. **Define the file paths**: To know where the datasets are stored.
2. **Load the datasets**: To bring the data into memory for analysis or training.
3. **Check dataset shape**: To understand the size and structure of the data (e.g., how many samples and features are available).

This part of the code configures **Pandas display options** to ensure that when you view the data in the DataFrame, all rows and columns are shown in the output. Here's a breakdown:

**1. Display All Columns**

pd.set\_option('display.max\_columns', None)

* **What it does**: Configures Pandas to show **all columns** when you print or display a DataFrame.
* **Why it's needed**: By default, Pandas limits the number of columns displayed to prevent clutter. If the dataset has many columns, some might be hidden (replaced with ellipses ...), making it hard to analyze.
* **Example**:
  + **Without this option**:
  + col1 col2 col3 col4 ... col10
  + 1 2 3 4 ... 10
  + **With this option**:
  + col1 col2 col3 col4 col5 col6 col7 col8 col9 col10
  + 1 2 3 4 5 6 7 8 9 10

**2. Display All Rows**

pd.set\_option('display.max\_rows', None)

* **What it does**: Configures Pandas to show **all rows** of a DataFrame when printed.
* **Why it's needed**: By default, Pandas limits the number of rows displayed to avoid overwhelming output. If you want to see the entire dataset, you need this setting.
* **Example**:
  + **Without this option** (if dataset has many rows):
  + 1 2 3 4
  + 5 6 7 8
  + ... (remaining rows truncated)
  + **With this option**:
  + 1 2 3 4
  + 5 6 7 8
  + 9 10 11 12
  + 13 14 15 16

**Why Use These Options?**

1. To **analyze the complete data structure** without missing hidden rows or columns.
2. Helpful for **debugging** or **exploratory data analysis** (EDA), where you need a clear view of all your data.

**Best Practices**

* These options are good for small datasets but may slow down the output or make it unreadable for large datasets.
* You can reset them to default later using:
* pd.reset\_option('display.max\_columns')
* pd.reset\_option('display.max\_rows')

**Explanation of the Code**

This part of the code combines the training and testing datasets and prints the shape of the resulting DataFrame. Let’s break it down:

**1. Combine the Datasets**

df = pd.concat((df\_train, df\_test))

* **pd.concat()**: Concatenates (joins) two or more DataFrames along a specified axis.
  + By default, it combines them **vertically** (one below the other) because the default axis is 0.
* **df\_train and df\_test**: These are the training and testing datasets.
* The result (df):
  + All rows from df\_train are placed on top.
  + All rows from df\_test are added below.
  + The number of columns in both datasets must match for this to work.

**2. Temporary Copy of the DataFrame**

temp\_df = df

* Creates a copy of the combined DataFrame (df) and assigns it to temp\_df.
* This can be useful if you want to experiment with the data (temp\_df) while keeping the original combined data (df) intact.

**3. Print the Shape of the Combined DataFrame**

print(df.shape)

* **df.shape**: Displays the number of rows and columns in the combined DataFrame.
  + The result is a tuple: (rows, columns).
* Example:
  + If df\_train has 1000 rows and 10 columns, and df\_test has 500 rows and the same 10 columns:
    - The combined df will have 1500 rows and 10 columns.
    - Output: (1500, 10).

**Why Combine the Datasets?**

1. **Preprocessing Together**:
   * Often, you want to apply the same preprocessing (like scaling, filling missing values, etc.) to both training and testing data. Combining them ensures consistent transformations.
2. **Feature Engineering**:
   * If you are creating new features, you might need to consider the entire dataset (training + testing).
3. **Temporary Combination**:
   * After preprocessing, you can split the data back into training and testing datasets if needed.

**Example Scenario**

Imagine a machine learning task where:

* Training data has known labels (e.g., house prices).
* Testing data has missing labels (prices you want to predict). Combining them simplifies the workflow for handling missing values, standardizing features, or encoding categorical data.

The **df.info()** method in Pandas provides a summary of a DataFrame, giving you important details about its structure and content. Here's a detailed explanation:

**What df.info() Does**

When you call df.info() on a DataFrame, it shows:

1. **Index and Range**:
   * Displays the total number of rows (e.g., RangeIndex: 1000 entries).
   * Shows the range of indices (e.g., 0 to 999).
2. **Column Names and Data Types**:
   * Lists all column names in the DataFrame.
   * Displays the data type for each column (e.g., int64, float64, object).
3. **Non-Null Counts**:
   * For each column, it shows how many non-null (not missing) values it has.
   * This helps you identify columns with missing data.
4. **Memory Usage**:
   * Provides the approximate memory usage of the DataFrame in bytes or megabytes.

The **df.describe()** function in Pandas provides a summary of the **statistical properties** of the numerical (and optionally categorical) columns in a DataFrame. Here's a detailed explanation:

**What df.describe() Does**

* By default, it generates descriptive statistics for **numerical columns** in the DataFrame.
* It summarizes key statistics like count, mean, standard deviation, minimum, maximum, and percentiles for each column.

**Default Output for Numerical Data**

If you call df.describe() on a DataFrame, it returns a table like this:

| **Statistic** | **Column1** | **Column2** | **Column3** |
| --- | --- | --- | --- |
| **count** | 100.0 | 100.0 | 100.0 |
| **mean** | 50.2 | 35.7 | 78.9 |
| **std** | 10.5 | 8.9 | 12.3 |
| **min** | 10.0 | 20.0 | 60.0 |
| **25%** (Q1) | 40.0 | 30.0 | 70.0 |
| **50%** (Median/Q2) | 50.0 | 35.0 | 80.0 |
| **75%** (Q3) | 60.0 | 40.0 | 90.0 |
| **max** | 80.0 | 50.0 | 100.0 |

**Explanation of Each Row:**

1. **count**: Number of non-missing values in the column.
2. **mean**: Average value of the column.
3. **std**: Standard deviation (how much values vary from the mean).
4. **min**: Minimum value in the column.
5. **25% (Q1)**: The 25th percentile (value below which 25% of the data falls).
6. **50% (Median/Q2)**: The 50th percentile (value below which 50% of the data falls).
7. **75% (Q3)**: The 75th percentile (value below which 75% of the data falls).
8. **max**: Maximum value in the column.

**Descriptive Statistics for Categorical Data**

If you want to include **categorical** (non-numeric) columns in the summary, use:

df.describe(include='object')

This generates:

| **Statistic** | **Column1 (Categorical)** |
| --- | --- |
| **count** | 100 |
| **unique** | 10 |
| **top** | "CategoryA" |
| **freq** | 20 |

**Explanation of Each Row:**

1. **count**: Number of non-missing values.
2. **unique**: Number of unique categories.
3. **top**: The most frequent category.
4. **freq**: Frequency of the most common category.

**Describe for All Data Types**

To describe both numerical and categorical columns together, use:

df.describe(include='all')

**Why Use df.describe()?**

1. **Quick Data Overview**: Summarize the central tendencies, spread, and range of your data.
2. **Identify Missing Data**: Columns with fewer counts than the total rows have missing values.
3. **Spot Anomalies**: Check for extreme values or unexpected distributions.
4. **Understand Data Distribution**: Percentiles give a sense of how the data is spread.