```
!pip install pydub
import pandas as pd  # Import for handling DataFrames
import json  # Import for reading JSON files
import xml.etree.ElementTree as ET  # Import for parsing XML files
from PIL import Image  # Import from PIL (Pillow) for handling image files
from pydub import AudioSegment  # Import from pydub for handling audio files
import cv2  # Import for handling video files
import concurrent.futures  # Import for parallel execution

def load_data(file_path):

    file_extension = file_path.split('.')[-1].lower()  # Determine the file extension from the file path

    try:
        # Load CSV files into a DataFrame
        if file_extension == 'csv':
            return pd.read_csv(file_path)
        # Load Excel files (both .xls and .xlsx) into a DataFrame,
        elif file_extension in ['xls', 'xlsx']:
            return pd.read_excel(file_path)
        # Load JSON files into a Python object
        elif file_extension == 'json':
            with open(file_path, 'r') as f:
                return json.load(f)
        # Parse XML files and return the root of the XML tree
        elif file_extension == 'xml':
            tree = ET.parse(file_path)
            return tree.getroot()
        # Load HDF5 files into a DataFrame
        elif file_extension in ['h5', 'hdf5']:
            return pd.read_hdf(file_path)
        # Load Feather files into a DataFrame
        elif file_extension == 'feather':
            return pd.read_feather(file_path)
        # Read text files as plain text
        elif file_extension == 'txt':
            with open(file_path, 'r') as f:
                return f.read()
        # Open image files (JPG, JPEG) using PIL
        elif file_extension in ['jpg', 'jpeg']:
            return Image.open(file_path)
        # Load audio files (MP3, WAV) using pydub
        elif file_extension in ['mp3', 'wav']:
            return AudioSegment.from_file(file_path)
        # Open video files (MP4, AVI, MKV) using cv2
        elif file_extension in ['mp4', 'avi', 'mkv']:
            return cv2.VideoCapture(file_path)
        else:
            # Raise an error if the file format is not supported
            raise ValueError(f"Unsupported file format: {file_extension}")
    except Exception as e:
        # Print error message if an exception occurs during file loading
        print(f"Error loading {file_path}: {e}")
        return None

def read_multiple_files(file_paths):
    # List to store loaded data from each file
    loaded_data = []

    # Use ThreadPoolExecutor to load files concurrently
    with concurrent.futures.ThreadPoolExecutor() as executor:
        # Submit load_data tasks for each file path
        future_to_file = {executor.submit(load_data, file_path): file_path for file_path in file_paths}
        # Process each future as it completes
        for future in concurrent.futures.as_completed(future_to_file):
            file_path = future_to_file[future]
            try:
                # Retrieve the result of the future
                data = future.result()
                # Add the data to loaded_data if it is not None
                if data is not None:
                    loaded_data.append(data)
            except Exception as e:
                # Print error message if an exception occurs during future execution
                print(f"Error loading {file_path}: {e}")

    return loaded_data

def merge_files_side_by_side(file_paths):
    # Read multiple files and get a list of DataFrames
    dfs = read_multiple_files(file_paths)
```

```
    # Filter out only DataFrames from the list
    dfs = [df for df in dfs if isinstance(df, pd.DataFrame)]

    # Handle the case with zero or one DataFrame
    if len(dfs) == 0:
        return None  # or raise an error, depending on desired behavior
    elif len(dfs) == 1:
        return dfs[0]

    # Concatenate DataFrames side-by-side (i.e., column-wise)
    merged_df = pd.concat(dfs, axis=1)

    return merged_df

def get_file_paths_from_input():
    # Prompt user for input and split by comma
    file_paths = input("Enter file paths separated by commas: ").split(',')
    # Strip any extra whitespace from each file path
    file_paths = [file_path.strip() for file_path in file_paths]
    return file_paths

# Example usage
file_paths = get_file_paths_from_input()  # Get the list of file paths from user input
print(type(file_paths))

if len(file_paths) > 1:
  if_yes = input("Do you want to merge the files side-by-side? (yes/no): ")
  if if_yes == "yes":
    merged_df = merge_files_side_by_side(file_paths)  # Merge the files side-by-side
  if if_yes == "no":
    pass

merged_df.head()
```

```
Collecting pydub
  Downloading pydub-0.25.1-py2.py3-none-any.whl.metadata (1.4 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub
Successfully installed pydub-0.25.1
Enter file paths separated by commas:
```

Start coding or generate with AI.