Appendix B: Dataset Catalogue

This appendix provides a comprehensive list of datasets relevant for neuroAI research, along with access information and example code for loading and working with these datasets.

B.1 Neuroscience Datasets

Electrophysiology Data

Allen Brain Observatory

- **Description**: Single-cell recordings from visual cortex of mice during visual stimulation
- **Contents**: Neural activity from ~75,000 neurons across multiple cortical areas and layers
- **Access**: https://observatory.brain-map.org/
- **Documentation**: https://allensdk.readthedocs.io/en/latest/brain_observatory.html

```python
# Example: Loading Allen Brain Observatory data
from allensdk.core.brain_observatory_cache import BrainObservatoryCache
import matplotlib.pyplot as plt

# Initialize the cache
boc = BrainObservatoryCache(manifest_file='boc_manifest.json')

# Get experiment containers for specific targeted structures and imaging depths
experiments = boc.get_experiment_containers(targeted_structures=['VISp'],
                                            imaging_depths=[175])

# Access the first experiment
container_id = experiments[0]['id']

# Get experiment with natural scenes stimulus
experiment_id = boc.get_ophys_experiments(
    experiment_container_ids=[container_id],
    stimuli=['natural_scenes'])[0]['id']

# Get neural data
data_set = boc.get_ophys_experiment_data(experiment_id)

# Get fluorescence traces for all cells
timestamps, traces = data_set.get_fluorescence_traces()
print(f"Recorded {traces.shape[0]} neurons for {len(timestamps)} timepoints")

# Plot example trace from first neuron
plt.figure(figsize=(12, 4))
plt.plot(timestamps, traces[0])
plt.xlabel('Time (s)')
plt.ylabel('Fluorescence (a.u.)')
plt.title('Example Neuron Activity')
plt.show()
```

Collaborative Research in Computational Neuroscience (CRCNS)

- **Description**: Diverse electrophysiology datasets from various labs and species
- **Contents**: Single-unit and multi-unit recordings, EEG, MEG, and more
- **Access**: https://crcns.org/
- **Documentation**: Varies by dataset

```python
# Example: Loading CRCNS data (HC1 - hippocampal recordings from rats)
import h5py
import numpy as np
import matplotlib.pyplot as plt

# Open the HDF5 file (you would need to download this)
with h5py.File('example_data.h5', 'r') as f:
    # List all groups
    print("Keys: %s" % list(f.keys()))

    # Get spike times for a specific cell
    spikes = np.array(f['spikes']['times'])

    # Plot raster
    plt.figure(figsize=(12, 4))
    plt.plot(spikes, np.ones_like(spikes), '|', markersize=10)
    plt.xlabel('Time (s)')
    plt.ylabel('Neuron')
    plt.title('Spike Raster')
    plt.grid(True)
    plt.show()
```

Neurodata Without Borders (NWB)

- **Description**: Standardized format for neurophysiology data
- **Contents**: Various datasets following the NWB format standard
- **Access**: https://www.nwb.org/example-datasets/
- **Documentation**: https://pynwb.readthedocs.io/

```python
# Example: Reading NWB formatted data
from pynwb import NWBHDF5IO
import matplotlib.pyplot as plt

# Open NWB file (you would need to download an example file)
with NWBHDF5IO('example.nwb', 'r') as io:
    nwbfile = io.read()

    # Print available acquisition data
    print(nwbfile.acquisition)

    # Access LFP data if available
    if 'LFP' in nwbfile.acquisition:
        lfp_data = nwbfile.acquisition['LFP'].data[:]
        lfp_timestamps = nwbfile.acquisition['LFP'].timestamps[:]

        # Plot LFP
        plt.figure(figsize=(12, 4))
        plt.plot(lfp_timestamps, lfp_data)
        plt.xlabel('Time (s)')
        plt.ylabel('Voltage (mV)')
        plt.title('LFP Recording')
        plt.show()
```

Neuroimaging Data

Human Connectome Project (HCP)

- **Description**: High-quality neuroimaging data from 1,200+ healthy young adults
- **Contents**: MRI (structural, functional, diffusion), MEG, behavioral, and genetic data
- **Access**: https://www.humanconnectome.org/study/hcp-young-adult
- **Documentation**: https://www.humanconnectome.org/study/hcp-young-adult/documentation

```python
# Example: Working with HCP data using nilearn
from nilearn import datasets, plotting
import matplotlib.pyplot as plt

# Download a preprocessed resting-state fMRI from HCP
# Note: You need to register and get credentials for actual HCP data
# This example uses a sample from nilearn, not the actual HCP dataset
hcp_dataset = datasets.fetch_development_fmri(n_subjects=1)

# Get the first subject's data
func_filename = hcp_dataset.func[0]

# Plot the mean image
mean_img = plotting.plot_epi(func_filename, title='Mean fMRI image')
plt.show()

# Plot the temporal variance
var_img = plotting.plot_stat_map(plotting.mean_img(func_filename),
                              title='Temporal variance')
plt.show()
```

OpenNeuro

- **Description**: Open platform for sharing BIDS-compatible neuroimaging data
- **Contents**: fMRI, EEG, MEG, iEEG datasets from various studies
- **Access**: https://openneuro.org/
- **Documentation**: Varies by dataset

```python
# Example: Using openneuro-py to download data
# !pip install openneuro-py
import openneuro
import os

# Create a download instance
api = openneuro.OpenNeuroAPI()

# Specify dataset ID (ds002422 is an example, you can find many on the website)
dataset_id = 'ds002422'

# Set download directory
download_dir = './openneuro_data'
os.makedirs(download_dir, exist_ok=True)

# Download a specific dataset
# This downloads the dataset metadata
downloader = api.download(dataset_id, download_dir)

# List all files (doesn't download them yet)
files = [f for f in downloader.files if not f.endswith('/')]
print(f"Total files: {len(files)}")

# Download a specific file
# downloader.download([files[0]])  # Uncomment to actually download

# For working with the data, you would typically use BIDS tools
# !pip install pybids
from bids import BIDSLayout

# Create a layout to work with the BIDS dataset (only metadata)
layout = BIDSLayout(download_dir)

# Get all available sessions
sessions = layout.get_sessions()
print(f"Available sessions: {sessions}")

# Get functional MRI runs
runs = layout.get(datatype='func', extension='.nii.gz')
print(f"Number of fMRI runs: {len(runs)}")
```

NeuroVault

- **Description**: Repository for statistical maps of the human brain
- **Contents**: fMRI and PET statistical maps
- **Access**: https://neurovault.org/
- **Documentation**: NeuroVault/NeuroVault

```python
# Example: Using neurovault's API
# !pip install requests
import requests
import matplotlib.pyplot as plt
from nilearn import plotting
import nibabel as nib
import numpy as np
import os

# Get collection info (collection 1952 is an example)
base_url = 'https://neurovault.org/api/collections/1952/'
response = requests.get(base_url)
collection_data = response.json()

# Get images in the collection
images_url = base_url + 'images/'
response = requests.get(images_url)
images_data = response.json()['results']
print(f"Found {len(images_data)} images")

# Download and display the first image
if len(images_data) > 0:
    # Get download URL for the first image
    img_url = images_data[0]['file']
    img_name = os.path.basename(img_url)

    # Download the image
    img_response = requests.get(img_url)
    with open(img_name, 'wb') as f:
        f.write(img_response.content)

    # Load and display the image
    img = nib.load(img_name)
    plotting.plot_stat_map(img, title=images_data[0]['name'])
    plt.show()
```

Behavior and Cognition

International Brain Laboratory (IBL)

- **Description**: Standardized mouse behavioral and neural data
- **Contents**: Neural recordings during decision-making tasks
- **Access**: https://www.internationalbrainlab.com/public-resources
- **Documentation**: https://docs.internationalbrainlab.org/

```
# Example: Loading IBL data
# !pip install ONE-api
from one.api import ONE
import numpy as np
import matplotlib.pyplot as plt

# Initialize ONE
one = ONE()

# Search for sessions with available data types
selection = one.search(dataset_types=['spikes.times', 'trials'])
print(f"Found {len(selection)} sessions")

# Select the first session
eid = selection[0]

# List all available data for this session
datasets = one.list_datasets(eid)
print(f"Available datasets: {datasets}")

# Load spike data
spikes_times, spikes_clusters = one.load_dataset(eid, 'spikes.times'), one.load_data

# Load trial data
trials = one.load_object(eid, 'trials')

# Plot raster for a few neurons
unique_clusters = np.unique(spikes_clusters)
print(f"Total neurons: {len(unique_clusters)}")

# Plot raster for 5 clusters
plt.figure(figsize=(15, 10))
for i, cluster_id in enumerate(unique_clusters[:5]):
    cluster_spikes = spikes_times[spikes_clusters == cluster_id]
    plt.plot(cluster_spikes, np.ones_like(cluster_spikes) * i, '|', markersize=1)

plt.xlabel('Time (s)')
plt.ylabel('Neuron ID')
plt.title('Spike Raster')
plt.show()
```

Brain Imaging Data Structure (BIDS) Examples

- **Description**: Standardized datasets following BIDS format
- **Contents**: Various types of brain imaging and behavioral data
- **Access**: bids-standard/bids-examples
- **Documentation**: https://bids.neuroimaging.io/

```python
# Example: Working with BIDS datasets
# !pip install pybids
from bids import BIDSLayout
import os

# Path to the BIDS dataset
bids_path = './bids_dataset'  # You would need to download or have a BIDS dataset

# Check if path exists
if os.path.exists(bids_path):
    # Initialize layout
    layout = BIDSLayout(bids_path)

    # Get participant information
    subjects = layout.get_subjects()
    print(f"Subjects: {subjects}")

    # Get available modalities
    modalities = layout.get_modalities()
    print(f"Modalities: {modalities}")

    # Get all T1w images
    t1w_images = layout.get(suffix='T1w', extension='.nii.gz')
    print(f"Found {len(t1w_images)} T1w images")

    # Get task fMRI data
    func_runs = layout.get(datatype='func', suffix='bold')
    print(f"Found {len(func_runs)} functional runs")

    # Access metadata for a specific file
    if func_runs:
        metadata = layout.get_metadata(func_runs[0].path)
        print("\nSample metadata:")
        for key, value in list(metadata.items())[:5]:
            print(f"{key}: {value}")
else:
    print(f"BIDS dataset not found at {bids_path}")
```

B.2 AI Benchmark Datasets

Computer Vision

ImageNet

- **Description**: Large-scale image classification benchmark

- **Contents**: ~14 million images across 20,000+ categories
- **Access**: https://www.image-net.org/
- **Documentation**: https://www.image-net.org/challenges/LSVRC/

```
# Example: Using PyTorch's ImageNet loader
import torch
from torchvision import datasets, transforms
import matplotlib.pyplot as plt
import numpy as np

# Define transformations
transform = transforms.Compose([
    transforms.Resize(256),
    transforms.CenterCrop(224),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

# Load ImageNet validation set (you need to have this downloaded)
# Replace 'path/to/imagenet/val' with your actual path
try:
    val_dataset = datasets.ImageNet('path/to/imagenet/val', split='val', transform=t
    val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=4, shuffle=True

    # Get a batch of images
    images, labels = next(iter(val_loader))

    # Display images
    def imshow(img):
        # Unnormalize
        inv_normalize = transforms.Normalize(
            mean=[-0.485/0.229, -0.456/0.224, -0.406/0.225],
            std=[1/0.229, 1/0.224, 1/0.225]
        )
        img = inv_normalize(img)
        npimg = img.numpy()
        plt.imshow(np.transpose(npimg, (1, 2, 0)))

    plt.figure(figsize=(12, 6))
    imshow(torchvision.utils.make_grid(images))
    plt.title('ImageNet Samples')
    plt.show()
except Exception as e:
    print(f"Error loading ImageNet: {e}")
    print("Note: You need to download the ImageNet dataset separately due to its siz
```

CIFAR-10/100

- **Description**: Small-scale image classification datasets
- **Contents**: 60,000 32x32 color images in 10 or 100 classes
- **Access**: https://www.cs.toronto.edu/~kriz/cifar.html
- **Documentation**: Built into most deep learning frameworks

```python
# Example: Using CIFAR-10 with PyTorch
import torch
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np

# Define transformations
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
])

# Load CIFAR-10 (downloaded automatically if not present)
train_dataset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                             download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=4,
                                           shuffle=True, num_workers=2)

test_dataset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                            download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=4,
                                          shuffle=False, num_workers=2)

# Class labels
classes = ('plane', 'car', 'bird', 'cat', 'deer',
           'dog', 'frog', 'horse', 'ship', 'truck')

# Function to display images
def imshow(img):
    img = img / 2 + 0.5  # Unnormalize
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))

# Get some random training images
dataiter = iter(train_loader)
images, labels = next(dataiter)

# Show images
plt.figure(figsize=(10, 5))
imshow(torchvision.utils.make_grid(images))
plt.title(' '.join('%5s' % classes[labels[j]] for j in range(4)))
plt.show()
```

MS COCO

- **Description**: Dataset for object detection, segmentation, and captioning
- **Contents**: 330,000+ images with 80 object categories

- **Access**: https://cocodataset.org/
- **Documentation**: cocodataset/cocoapi

```python
# Example: Working with COCO dataset
# !pip install pycocotools
from pycocotools.coco import COCO
import numpy as np
import skimage.io as io
import matplotlib.pyplot as plt
import requests
from io import BytesIO

# Initialize COCO API for instance annotations
dataType = 'val2017'
annFile = f'./annotations/instances_{dataType}.json'  # You need to download this

try:
    coco = COCO(annFile)

    # Get categories and supercategories
    cats = coco.loadCats(coco.getCatIds())
    cat_names = [cat['name'] for cat in cats]
    print(f"COCO categories: {cat_names}")

    # Get all images containing specific categories
    catIds = coco.getCatIds(catNms=['person', 'dog', 'skateboard'])
    imgIds = coco.getImgIds(catIds=catIds)
    print(f"Found {len(imgIds)} images with specified categories")

    # Load and display an image
    if imgIds:
        img = coco.loadImgs(imgIds[np.random.randint(0, len(imgIds))])[0]

        # Use URL to load image
        I = io.imread(img['coco_url'])
        plt.figure(figsize=(12, 9))
        plt.imshow(I)
        plt.axis('off')

        # Load and display instance annotations
        annIds = coco.getAnnIds(imgIds=img['id'], catIds=catIds, iscrowd=None)
        anns = coco.loadAnns(annIds)
        plt.figure(figsize=(12, 9))
        plt.imshow(I)
        plt.axis('off')
        coco.showAnns(anns)
        plt.show()
except Exception as e:
    print(f"Error working with COCO: {e}")
    print("Note: You need to download the COCO dataset separately.")

    # Display a sample image from the COCO website instead
    try:
        url = "http://images.cocodataset.org/val2017/000000013729.jpg"
        response = requests.get(url)
        I = io.imread(BytesIO(response.content))
        plt.figure(figsize=(12, 9))
        plt.imshow(I)
        plt.axis('off')
        plt.title("Sample COCO Image")
        plt.show()
    except:
        print("Could not display sample image")
```

Natural Language Processing

GLUE Benchmark

- **Description**: General Language Understanding Evaluation benchmark
- **Contents**: Collection of 9 NLP tasks for evaluating language understanding
- **Access**: https://gluebenchmark.com/
- **Documentation**: nyu-mll/GLUE-baselines

```python
# Example: Loading GLUE tasks with the datasets library
# !pip install datasets transformers
from datasets import load_dataset
from transformers import AutoTokenizer
import pandas as pd

# Load a GLUE task (MNLI for example)
dataset = load_dataset("glue", "mnli")

# Print dataset structure
print("Dataset structure:")
print(dataset)

# Load a tokenizer
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")

# Display some examples
print("\nExample data:")
df = pd.DataFrame(dataset["train"][:5])
print(df[["premise", "hypothesis", "label"]])

# Define a function to tokenize the data
def tokenize_function(examples):
    return tokenizer(
        examples["premise"],
        examples["hypothesis"],
        padding="max_length",
        truncation=True,
    )

# Tokenize the dataset
tokenized_datasets = dataset.map(tokenize_function, batched=True)
print("\nTokenized dataset features:")
print(tokenized_datasets["train"].column_names)
```

SQuAD

- **Description**: Stanford Question Answering Dataset
- **Contents**: 100,000+ question-answer pairs on Wikipedia articles
- **Access**: https://rajpurkar.github.io/SQuAD-explorer/
- **Documentation**: rajpurkar/SQuAD-explorer

```python
# Example: Working with SQuAD dataset
# !pip install datasets transformers
from datasets import load_dataset
import pandas as pd
import random

# Load SQuAD dataset
squad = load_dataset("squad")

# Print dataset structure
print("Dataset structure:")
print(squad)

# Convert a few examples to DataFrame for easier viewing
train_examples = []
for example in squad["train"][:5]:
    train_examples.append({
        "question": example["question"],
        "context": example["context"][:100] + "...",  # Truncate for display
        "answers": example["answers"]["text"][0]
    })

df = pd.DataFrame(train_examples)
print("\nSample questions and answers:")
print(df)

# Select a random example and show full context
random_idx = random.randint(0, len(squad["train"]) - 1)
example = squad["train"][random_idx]

print("\nRandom example:")
print(f"Question: {example['question']}")
print(f"Answer: {example['answers']['text'][0]}")
print(f"\nContext: {example['context']}")
```

WikiText

- **Description**: Language modeling dataset of Wikipedia articles
- **Contents**: 103 million words with long-context dependencies
- **Access**: https://blog.salesforceairesearch.com/the-wikitext-long-term-dependency-language-modeling-dataset/
- **Documentation**: salesforce/awd-lstm-lm

```
# Example: Loading WikiText dataset
# !pip install datasets
from datasets import load_dataset
import matplotlib.pyplot as plt
import numpy as np

# Load WikiText-2 dataset
wikitext = load_dataset("wikitext", "wikitext-2-raw-v1")

# Print dataset structure
print("Dataset structure:")
print(wikitext)

# Display a sample from the training set
print("\nSample from training set:")
print(wikitext["train"][100]["text"])

# Compute statistics
train_lens = [len(sample["text"].split()) for sample in wikitext["train"] if sample[

# Plot distribution of sequence lengths
plt.figure(figsize=(10, 6))
plt.hist(train_lens, bins=50)
plt.xlabel('Sequence Length (words)')
plt.ylabel('Count')
plt.title('Distribution of Sequence Lengths in WikiText-2')
plt.axvline(np.mean(train_lens), color='r', linestyle='dashed', linewidth=1, label=f
plt.axvline(np.median(train_lens), color='g', linestyle='dashed', linewidth=1, label
plt.legend()
plt.show()

# Count total words
total_words = sum(train_lens)
print(f"\nTotal words in training set: {total_words:,}")

# Compute vocabulary size (unique words)
all_text = " ".join([sample["text"] for sample in wikitext["train"] if sample["text"
vocab = set(all_text.split())
print(f"Vocabulary size: {len(vocab):,} unique words")
```

Reinforcement Learning

OpenAI Gym

- **Description**: Toolkit for developing and comparing RL algorithms
- **Contents**: Collection of environments from simple to complex

- **Access**: https://gym.openai.com/
- **Documentation**: openai/gym

```
# Example: Basic usage of OpenAI Gym
# !pip install gym
import gym
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import animation

# Create an environment
env = gym.make('CartPole-v1')

# Reset the environment
observation = env.reset()

# Run a simple random policy
frames = []
done = False
for _ in range(100):
    # Render the environment
    frames.append(env.render(mode='rgb_array'))

    # Take a random action
    action = env.action_space.sample()

    # Step in the environment
    observation, reward, done, info = env.step(action)

    if done:
        observation = env.reset()

env.close()

# Display the animation
plt.figure(figsize=(8, 6))

def update_frame(i):
    plt.clf()
    plt.imshow(frames[i])
    plt.axis('off')
    return [plt.gca()]

ani = animation.FuncAnimation(plt.gcf(), update_frame, frames=len(frames), interval=
plt.show()

# Print environment details
print(f"Observation space: {env.observation_space}")
print(f"Action space: {env.action_space}")
```

DeepMind Lab

- **Description**: 3D learning environment based on Quake III Arena
- **Contents**: Navigation and puzzle-solving tasks in 3D environment
- **Access**: deepmind/lab
- **Documentation**: deepmind/lab

```python
# Example: Using DeepMind Lab (note: requires separate installation)
try:
    import deepmind_lab
    import numpy as np
    import matplotlib.pyplot as plt

    # Create a simple DeepMind Lab environment
    lab = deepmind_lab.Lab(
        'seekavoid_arena_01',  # Level name
        ['RGB_INTERLEAVED'],   # Observations to return
        config={
            'width': 320,
            'height': 240,
            'fps': 60
        }
    )

    # Reset the environment
    lab.reset()

    # Run a few random actions and display observations
    for _ in range(10):
        # Generate a random action (forward/backward, strafe, look)
        action = np.zeros([7], dtype=np.intc)  # DeepMind Lab actions are 7D
        action[0] = np.random.choice([-1, 0, 1])  # Forward/backward
        action[2] = np.random.choice([-1, 0, 1])  # Strafe left/right

        # Step in the environment
        reward = lab.step(action, num_steps=4)  # Execute 4 frames

        # Get observation
        obs = lab.observations()['RGB_INTERLEAVED']

        # Display observation
        plt.figure(figsize=(8, 6))
        plt.imshow(obs)
        plt.title(f"Reward: {reward}")
        plt.axis('off')
        plt.show()

    lab.close()
except ImportError:
    print("DeepMind Lab not installed. It requires a separate installation.")
    print("See: https://github.com/deepmind/lab for installation instructions.")
```

MuJoCo

- **Description**: Physics-based robot simulation environment
- **Contents**: Various robot control tasks

- **Access**: openai/mujoco-py
- **Documentation**: https://www.gymlibrary.ml/environments/mujoco/

```
# Example: Using MuJoCo environments in Gym
# !pip install gym mujoco-py
try:
    import gym
    import numpy as np
    import matplotlib.pyplot as plt
    from matplotlib import animation

    # Create a MuJoCo environment
    env = gym.make('HalfCheetah-v2')
    observation = env.reset()

    # Run a random policy
    frames = []
    for _ in range(100):
        # Render
        frames.append(env.render(mode='rgb_array'))

        # Random action
        action = env.action_space.sample()

        # Step
        observation, reward, done, info = env.step(action)

        if done:
            observation = env.reset()

    env.close()

    # Display the animation
    plt.figure(figsize=(8, 6))

    def update_frame(i):
        plt.clf()
        plt.imshow(frames[i])
        plt.axis('off')
        return [plt.gca()]

    ani = animation.FuncAnimation(plt.gcf(), update_frame, frames=len(frames), inter
    plt.show()

    # Print environment details
    print(f"Observation space: {env.observation_space}")
    print(f"Action space: {env.action_space}")
except ImportError as e:
    print(f"Error: {e}")
    print("MuJoCo requires a separate installation and license.")
    print("See: https://github.com/openai/mujoco-py for installation instructions.")
```

B.3 NeuroAI Specific Datasets

Brain-Score

- **Description**: Benchmark for neural network models of the visual system
- **Contents**: Neural and behavioral data for evaluating computational models
- **Access**: https://www.brain-score.org/
- **Documentation**: brain-score/brain-score

```
# Example: Using Brain-Score to evaluate models
# !pip install brain-score
try:
    import brainscore
    from brainscore.benchmarks.public_benchmarks import MajajHongITPublicBenchmark
    from brainscore.model_interface import BrainModel

    # Create a simple mock model for illustration
    class MockModel(BrainModel):
        def __init__(self):
            self.recording_target = None
            self.stimuli_identifier = None

        def look_at(self, stimuli, number_of_trials=1):
            import numpy as np
            # Return random activations
            return np.random.rand(len(stimuli), 10)  # 10 neurons

        def start_task(self, task, fitting_stimuli=None):
            pass

        def start_recording(self, recording_target='IT', time_bins=None):
            self.recording_target = recording_target

    # Create the model
    model = MockModel()

    # Create benchmark
    benchmark = MajajHongITPublicBenchmark()

    # Score the model
    score = benchmark(model)
    print(f"Model score: {score}")
except ImportError:
    print("brain-score not installed. Use: pip install brain-score")
    print("Note: brain-score may have specific dependencies and requirements.")
```

Neural Data Challenge

- **Description**: Neural decoding competitions
- **Contents**: Neural recordings with specific decoding tasks
- **Access**: https://neurodatachallenge.org/
- **Documentation**: Varies by challenge

```python
# Example: Generic neural decoding pipeline (using simulated data)
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report
import matplotlib.pyplot as plt

# Simulate neural data with 3 target classes
def simulate_neural_data(n_samples=1000, n_neurons=100, n_classes=3):
    # Create class-specific patterns
    X = np.zeros((n_samples, n_neurons))
    y = np.zeros(n_samples, dtype=int)

    # Generate different patterns for each class
    patterns = np.random.randn(n_classes, n_neurons)

    samples_per_class = n_samples // n_classes
    for c in range(n_classes):
        start_idx = c * samples_per_class
        end_idx = (c + 1) * samples_per_class if c < n_classes - 1 else n_samples

        # Assign class labels
        y[start_idx:end_idx] = c

        # Generate neural activity
        X[start_idx:end_idx] = patterns[c] + np.random.randn(end_idx - start_idx, n_

    return X, y

# Generate simulated data
X, y = simulate_neural_data()

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_stat

# Preprocess: Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Train a decoder
model = LogisticRegression(max_iter=1000)
model.fit(X_train_scaled, y_train)

# Evaluate
y_pred = model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Decoding accuracy: {accuracy:.3f}")
print(classification_report(y_test, y_pred))

# Plot confusion matrix
from sklearn.metrics import confusion_matrix
```

```
import seaborn as sns

cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Visualize feature importance
coef = model.coef_
plt.figure(figsize=(12, 4))
for i, c in enumerate(coef):
    plt.subplot(1, len(coef), i+1)
    plt.bar(range(len(c)), c)
    plt.title(f'Class {i} vs Rest')
    plt.xlabel('Neuron ID')
    plt.ylabel('Weight')
plt.tight_layout()
plt.show()
```

Algonauts Project

- **Description**: Bridging human and machine vision
- **Contents**: fMRI data for training and evaluating computer vision models
- **Access**: http://algonauts.csail.mit.edu/
- **Documentation**: Brain-Bridge-Lab/Algonauts2023

```python
# Example: Working with Algonauts data (simulated)
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression
from sklearn.model_selection import KFold
from sklearn.metrics import r2_score

# Simulate fMRI data and CNN features
def simulate_algonauts_data(n_samples=500, n_voxels=1000, n_cnn_features=4096):
    # Simulate CNN features
    cnn_features = np.random.randn(n_samples, n_cnn_features)

    # Create "true" mapping weights
    true_weights = np.random.randn(n_cnn_features, n_voxels) * 0.01

    # Generate fMRI data using the features and weights
    fmri_data = cnn_features @ true_weights + np.random.randn(n_samples, n_voxels) *

    return cnn_features, fmri_data

# Generate simulated data
cnn_features, fmri_data = simulate_algonauts_data()

# Reduce dimensionality of CNN features for visualization
pca = PCA(n_components=100)
cnn_reduced = pca.fit_transform(cnn_features)

# Set up cross-validation
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Train a model to predict fMRI responses from CNN features
pls = PLSRegression(n_components=20)

# For simplicity, just train on a subset of voxels
voxel_subset = np.random.choice(fmri_data.shape[1], 100, replace=False)
fmri_subset = fmri_data[:, voxel_subset]

# Cross-validation
print("Running cross-validation...")
r2_scores = []
for train_idx, test_idx in kf.split(cnn_features):
    # Train/test split
    X_train, X_test = cnn_features[train_idx], cnn_features[test_idx]
    y_train, y_test = fmri_subset[train_idx], fmri_subset[test_idx]

    # Fit model
    pls.fit(X_train, y_train)

    # Predict
    y_pred = pls.predict(X_test)

    # Compute R² for each voxel
    r2 = [r2_score(y_test[:, i], y_pred[:, i]) for i in range(y_test.shape[1])]
    r2_scores.append(np.mean(r2))

print(f"Mean R² across folds: {np.mean(r2_scores):.3f}")

# Visualize distribution of R² scores
plt.figure(figsize=(10, 6))
plt.hist(r2_scores, bins=10)
plt.xlabel('Mean R² Score')
plt.ylabel('Count')
plt.title('Distribution of R² Scores Across CV Folds')
plt.grid(True)
plt.show()
```

B.4 Data Loading Examples

The examples below demonstrate how to load specific file formats common in neuroscience research.

Loading NIfTI Files

```
# Example: Working with NIfTI files (neuroimaging data)
# !pip install nibabel matplotlib
import nibabel as nib
import matplotlib.pyplot as plt
import numpy as np
import requests
import os

# Download a sample NIfTI file if needed
nifti_url = "https://ndownloader.figshare.com/files/12965017"
filename = "example.nii.gz"

if not os.path.exists(filename):
    try:
        print(f"Downloading {filename}...")
        response = requests.get(nifti_url)
        with open(filename, 'wb') as f:
            f.write(response.content)
        print("Download complete.")
    except Exception as e:
        print(f"Error downloading file: {e}")

# Load NIfTI file
try:
    img = nib.load(filename)

    # Get data as numpy array
    data = img.get_fdata()
    print(f"Image shape: {data.shape}")
    print(f"Data type: {data.dtype}")

    # Get header information
    header = img.header
    print(f"\nVoxel dimensions: {header.get_zooms()}")
    print(f"Units: {header.get_xyzt_units()}")

    # Display central slices
    if len(data.shape) >= 3:
        middle_x = data.shape[0] // 2
        middle_y = data.shape[1] // 2
        middle_z = data.shape[2] // 2

        plt.figure(figsize=(12, 4))

        plt.subplot(131)
        plt.imshow(data[middle_x, :, :].T, cmap='gray')
        plt.title('Sagittal View')
        plt.axis('off')

        plt.subplot(132)
        plt.imshow(data[:, middle_y, :].T, cmap='gray')
        plt.title('Coronal View')
        plt.axis('off')

        plt.subplot(133)
        plt.imshow(data[:, :, middle_z].T, cmap='gray')
        plt.title('Axial View')
        plt.axis('off')

        plt.tight_layout()
        plt.show()
except Exception as e:
    print(f"Error processing NIfTI file: {e}")
    print("Will use a simulated 3D volume instead for demonstration:")

    # Create a simple simulated volume
    sim_data = np.zeros((30, 30, 30))

    # Add a sphere in the middle
    x, y, z = np.ogrid[-15:15, -15:15, -15:15]
    sphere = x*x + y*y + z*z <= 10*10
    sim_data[sphere] = 1

    # Display central slices
    middle_x = sim_data.shape[0] // 2
    middle_y = sim_data.shape[1] // 2
    middle_z = sim_data.shape[2] // 2

    plt.figure(figsize=(12, 4))

    plt.subplot(131)
    plt.imshow(sim_data[middle_x, :, :].T, cmap='gray')
    plt.title('Sagittal View (Simulated)')
    plt.axis('off')

    plt.subplot(132)
    plt.imshow(sim_data[:, middle_y, :].T, cmap='gray')
    plt.title('Coronal View (Simulated)')
    plt.axis('off')

    plt.subplot(133)
    plt.imshow(sim_data[:, :, middle_z].T, cmap='gray')
    plt.title('Axial View (Simulated)')
    plt.axis('off')

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

Working with EEG Data

```python
# Example: Working with EEG data
# !pip install mne
try:
    import mne
    import numpy as np
    import matplotlib.pyplot as plt

    # Download sample EEG data
    sample_data_folder = mne.datasets.sample.data_path()
    sample_data_raw_file = f"{sample_data_folder}/MEG/sample/sample_audvis_raw.fif"

    # Load the data
    raw = mne.io.read_raw_fif(sample_data_raw_file, preload=True)

    # Print information about the dataset
    print(raw.info)

    # Extract EEG channels only
    raw.pick_types(meg=False, eeg=True, eog=False, stim=False)

    # Filter the data
    raw.filter(l_freq=1.0, h_freq=40.0)

    # Plot EEG data
    raw.plot(duration=5, n_channels=10, scalings='auto')

    # Extract events
    events = mne.find_events(raw, stim_channel='STI 014')
    event_id = {'auditory/left': 1, 'auditory/right': 2, 'visual/left': 3, 'visual/r

    # Create epochs
    epochs = mne.Epochs(raw, events, event_id, tmin=-0.2, tmax=0.5,
                        proj=True, baseline=(None, 0), preload=True)

    # Plot evoked responses
    evoked = epochs['auditory/left'].average()
    evoked.plot()

    # Create and plot a topographic map
    evoked.plot_topomap(times=[0.1], ch_type='eeg')

    # Plot time-frequency representation
    frequencies = np.arange(6, 20, 2)  # Frequencies from 6-20Hz
    power = mne.time_frequency.tfr_morlet(epochs['auditory/left'], freqs=frequencies
                                          n_cycles=2, return_itc=False)
    power.plot([0])
except Exception as e:
    print(f"Error processing EEG data: {e}")
    print("\nNote: MNE-Python requires sample data to be downloaded.")
    print("You can still work with EEG data by following these steps:")
    print("1. Install MNE: pip install mne")
    print("2. Download sample data: mne.datasets.sample.data_path()")
    print("3. Load and process EEG data as shown in the example")

    # Create a simulated EEG dataset for demonstration
    try:
        import numpy as np
        import matplotlib.pyplot as plt

        # Parameters
        n_channels = 16
        sfreq = 256  # Hz
        duration = 5  # seconds
        t = np.arange(0, duration, 1/sfreq)
        n_samples = len(t)

        # Generate simulated EEG signals
        sim_eeg = np.zeros((n_channels, n_samples))

        for i in range(n_channels):
            # Combine different frequency components with channel-specific weights
            alpha = 0.5 * np.sin(2 * np.pi * 10 * t)  # 10 Hz alpha
            beta = 0.25 * np.sin(2 * np.pi * 20 * t)  # 20 Hz beta
            theta = 0.3 * np.sin(2 * np.pi * 5 * t)  # 5 Hz theta
            delta = 0.4 * np.sin(2 * np.pi * 2 * t)  # 2 Hz delta

            # Mix components with random weights and add noise
            weights = np.random.rand(4)
            weights = weights / np.sum(weights)
            sim_eeg[i] = (weights[0] * alpha +
                         weights[1] * beta +
                         weights[2] * theta +
                         weights[3] * delta +
                         0.1 * np.random.randn(n_samples))

        # Plot simulated EEG data
        plt.figure(figsize=(10, 8))
        for i in range(min(8, n_channels)):
            plt.subplot(8, 1, i+1)
            plt.plot(t, sim_eeg[i])
            plt.ylabel(f'Ch {i+1}')
            if i == 0:
                plt.title('Simulated EEG Data')
            if i == 7:
                plt.xlabel('Time (s)')
        plt.tight_layout()
        plt.show()

        # Create a simulated topographic map
        plt.figure(figsize=(8, 8))
        channel_locs = []
        for i in range(n_channels):
            # Create circular layout
            angle = i * 2 * np.pi / n_channels
            x = 0.8 * np.cos(angle)
```

```
            y = 0.8 * np.sin(angle)
            channel_locs.append((x, y))

        # Create a grid for interpolation
        grid_size = 100
        xi = np.linspace(-1, 1, grid_size)
        yi = np.linspace(-1, 1, grid_size)
        X, Y = np.meshgrid(xi, yi)

        # Simulate some values at a given time point
        values = np.random.rand(n_channels)

        # Plot values at channel locations
        plt.scatter([loc[0] for loc in channel_locs], [loc[1] for loc in channel_loc
                    c=values, s=200, cmap='RdBu_r')

        # Add channel labels
        for i, loc in enumerate(channel_locs):
            plt.text(loc[0], loc[1], f'{i+1}', ha='center', va='center', color='whit

        plt.title('Simulated EEG Topographic Map')
        plt.colorbar(label='Activation')
        plt.axis('equal')
        plt.axis('off')
        plt.tight_layout()
        plt.show()
    except Exception as e:
        print(f"Error creating simulated EEG data: {e}")
```

Working with Spike Train Data

```python
# Example: Working with spike train data
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator

# Simulate spike train data for multiple neurons
def simulate_spike_trains(n_neurons=10, rate=10, duration=5.0, bin_size=0.001):
    """Simulate Poisson spike trains for multiple neurons"""
    # Create time bins
    bins = np.arange(0, duration + bin_size, bin_size)
    n_bins = len(bins) - 1

    # Initialize spike rasters and times
    spike_rasters = np.zeros((n_neurons, n_bins), dtype=bool)
    spike_times = [[] for _ in range(n_neurons)]

    # Generate spikes for each neuron
    for i in range(n_neurons):
        # Rate might vary between neurons
        neuron_rate = rate * (0.5 + np.random.rand())

        # Probability of spike in each bin
        spike_prob = neuron_rate * bin_size

        # Generate spikes
        for j in range(n_bins):
            if np.random.rand() < spike_prob:
                spike_rasters[i, j] = True
                spike_times[i].append(bins[j])

    return spike_rasters, spike_times, bins

# Simulate data
n_neurons = 10
duration = 5.0  # seconds
bin_size = 0.001  # 1 ms bins
spike_rasters, spike_times, bins = simulate_spike_trains(
    n_neurons=n_neurons, rate=10, duration=duration, bin_size=bin_size)

# Plot spike raster
plt.figure(figsize=(14, 6))

# Raster plot
plt.subplot(2, 1, 1)
for i in range(n_neurons):
    plt.plot(spike_times[i], np.ones_like(spike_times[i]) * i, '|', markersize=3)
plt.ylabel('Neuron ID')
plt.title('Spike Raster Plot')
plt.ylim(-0.5, n_neurons - 0.5)
plt.gca().yaxis.set_major_locator(MaxNLocator(integer=True))
```

```python
# PSTH (Population Peristimulus Time Histogram)
plt.subplot(2, 1, 2)
bin_edges = np.arange(0, duration + 0.1, 0.1)  # 100 ms bins for PSTH
population_rate = np.zeros((n_neurons, len(bin_edges) - 1))

for i in range(n_neurons):
    # Count spikes in each time bin
    counts, _ = np.histogram(spike_times[i], bins=bin_edges)

    # Convert to firing rate (Hz)
    population_rate[i] = counts / 0.1  # 0.1s bin size

# Average across neurons
avg_population_rate = np.mean(population_rate, axis=0)

# Plot PSTH
plt.bar(bin_edges[:-1], avg_population_rate, width=0.1, alpha=0.7)
plt.xlabel('Time (s)')
plt.ylabel('Firing Rate (Hz)')
plt.title('Population PSTH')

plt.tight_layout()
plt.show()

# Calculate and plot various spike train statistics
plt.figure(figsize=(14, 8))

# 1. Calculate ISI (Inter-Spike Intervals) for all neurons
isis = [np.diff(times) for times in spike_times if len(times) > 1]

# 2. Calculate CV (Coefficient of Variation) of ISIs
cvs = [np.std(isi)/np.mean(isi) if len(isi) > 0 else np.nan for isi in isis]

# 3. Calculate mean firing rates
rates = [len(times)/duration for times in spike_times]

# Plot ISI histogram
plt.subplot(2, 2, 1)
all_isis = np.concatenate(isis) if isis else np.array([])
if len(all_isis) > 0:
    plt.hist(all_isis, bins=50, density=True, alpha=0.7)
    plt.xlabel('Inter-Spike Interval (s)')
    plt.ylabel('Density')
    plt.title('ISI Distribution')
else:
    plt.text(0.5, 0.5, "No sufficient spikes for ISI", ha='center')

# Plot CV distribution
plt.subplot(2, 2, 2)
valid_cvs = [cv for cv in cvs if not np.isnan(cv)]
if valid_cvs:
    plt.hist(valid_cvs, bins=10, alpha=0.7)
    plt.axvline(x=1.0, color='r', linestyle='--', label='Poisson')
    plt.xlabel('CV of ISI')
    plt.ylabel('Count')
    plt.title('CV Distribution')
    plt.legend()
else:
    plt.text(0.5, 0.5, "No sufficient spikes for CV", ha='center')

# Plot firing rate distribution
plt.subplot(2, 2, 3)
plt.hist(rates, bins=10, alpha=0.7)
plt.xlabel('Firing Rate (Hz)')
plt.ylabel('Count')
plt.title('Firing Rate Distribution')

# Plot autocorrelation for the first neuron
plt.subplot(2, 2, 4)
if len(spike_rasters[0]) > 0:
    # Convert spike raster to 0/1 array
    spikes_01 = spike_rasters[0].astype(int)

    # Compute autocorrelation
    max_lag = int(0.5 / bin_size)  # 500 ms max lag
    lags = np.arange(-max_lag, max_lag + 1) * bin_size

    # Manual autocorrelation (numpy.correlate doesn't handle this well)
    autocorr = np.zeros(len(lags))
    for i, lag in enumerate(range(-max_lag, max_lag + 1)):
        if lag < 0:
            autocorr[i] = np.sum(spikes_01[:-lag] * spikes_01[-lag:])
        else:
            autocorr[i] = np.sum(spikes_01[lag:] * spikes_01[:-lag if lag > 0 else N

    # Plot autocorrelation
    plt.plot(lags, autocorr)
    plt.xlabel('Lag (s)')
    plt.ylabel('Autocorrelation')
    plt.title(f'Autocorrelation (Neuron 0)')
else:
    plt.text(0.5, 0.5, "No spikes for autocorrelation", ha='center')

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

These examples and resources should provide a comprehensive starting point for working with a wide range of neuroscience and AI datasets.

B.5 Neurological and Neurodegenerative Disease

Datasets

This section catalogs datasets specifically focused on neurological and neurodegenerative diseases, which are valuable resources for developing diagnostic, prognostic, and treatment monitoring AI systems.

Alzheimer’s Disease

Alzheimer’s Disease Neuroimaging Initiative (ADNI)

- **Description**: Longitudinal multimodal data to track Alzheimer’s progression
- **Contents**: MRI, PET, cognitive assessments, biospecimen data from 800+ subjects
- **Access**: http://adni.loni.usc.edu/ (requires application)
- **Documentation**: http://adni.loni.usc.edu/methods/

```python
# Example: Working with ADNI MRI data
# Note: You need approved access to ADNI to download data
import nibabel as nib
import matplotlib.pyplot as plt
import numpy as np
import os

def visualize_adni_mri(filepath):
    """Visualize a 3D MRI from ADNI dataset."""
    # Load MRI volume
    img = nib.load(filepath)
    data = img.get_fdata()

    # Get middle slices for each plane
    x_mid = data.shape[0] // 2
    y_mid = data.shape[1] // 2
    z_mid = data.shape[2] // 2

    # Create a figure with three subplots
    fig, axes = plt.subplots(1, 3, figsize=(15, 5))

    # Sagittal view (YZ plane)
    axes[0].imshow(data[x_mid, :, :].T, cmap='gray', origin='lower')
    axes[0].set_title('Sagittal View')

    # Coronal view (XZ plane)
    axes[1].imshow(data[:, y_mid, :].T, cmap='gray', origin='lower')
    axes[1].set_title('Coronal View')

    # Axial view (XY plane)
    axes[2].imshow(data[:, :, z_mid].T, cmap='gray', origin='lower')
    axes[2].set_title('Axial View')

    plt.tight_layout()
    plt.show()

# Example of preprocessing ADNI data for machine learning
def preprocess_adni_for_ml(mri_files, diagnoses):
    """
    Preprocess ADNI MRI files for ML classification.

    Parameters:
    mri_files: List of paths to MRI files
    diagnoses: List of diagnoses (e.g., 'CN', 'MCI', 'AD')

    Returns:
    features: Extracted features for each MRI
    labels: Corresponding diagnostic labels
    """
    # For demonstration - would require actual data
    features = []

    for mri_file in mri_files:
        # Load MRI
        img = nib.load(mri_file)
        data = img.get_fdata()

        # Example feature extraction (hippocampal ROI)
        # This is simplified - real implementation would use proper segmentation
        x_range = slice(90, 110)  # Example hippocampal coordinates
        y_range = slice(60, 80)
        z_range = slice(60, 80)

        hippocampus_roi = data[x_range, y_range, z_range]

        # Extract features (e.g., mean, variance, etc.)
        features.append([
            np.mean(hippocampus_roi),
            np.var(hippocampus_roi),
            np.max(hippocampus_roi),
            np.min(hippocampus_roi)
        ])

    # Convert diagnoses to numeric labels
    label_map = {'CN': 0, 'MCI': 1, 'AD': 2}  # Control, Mild Cognitive Impairment,
    labels = [label_map[d] for d in diagnoses]

    return np.array(features), np.array(labels)

# Usage example (pseudocode)
# mri_files = ['/path/to/adni/001.nii', '/path/to/adni/002.nii', ...]
# diagnoses = ['CN', 'MCI', 'AD', ...]
# features, labels = preprocess_adni_for_ml(mri_files, diagnoses)
```

OASIS (Open Access Series of Imaging Studies)

- **Description**: Free datasets for brain aging and Alzheimer’s research
- **Contents**: MRI scans, demographics, clinical data for 400+ subjects
- **Access**: https://www.oasis-brains.org/
- **Documentation**: https://www.oasis-brains.org/#data

```python
# Example: Downloading and visualizing OASIS-3 data
# !pip install nilearn nibabel matplotlib
import os
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
from nilearn import datasets, plotting

# OASIS data can be accessed via nilearn
oasis_dataset = datasets.fetch_oasis_vbm(n_subjects=5)
print(f"Dataset directory: {oasis_dataset.dirname}")
print(f"Gray matter anatomy: {len(oasis_dataset.gray_matter_maps)}")

# Visualize a gray matter map
gm_img = oasis_dataset.gray_matter_maps[0]
plotting.plot_img(gm_img, title="OASIS Gray Matter Map")
plt.show()

# Plot the first 5 subject MRIs
fig, axes = plt.subplots(1, min(5, len(oasis_dataset.gray_matter_maps)), figsize=(15

for i, gm_file in enumerate(oasis_dataset.gray_matter_maps[:5]):
    # Load the image
    img = nib.load(gm_file)
    # Get a slice at the middle of the 3rd dimension
    data = img.get_fdata()
    slice_idx = data.shape[2] // 2
    # Plot on the corresponding axis
    axes[i].imshow(data[:, :, slice_idx], cmap='gray')
    axes[i].set_title(f'Subject {i+1}')
    axes[i].axis('off')

plt.tight_layout()
plt.show()

# Plot clinical information
y = oasis_dataset.ext_vars['y']  # Mini-Mental State Examination (MMSE) scores
age = oasis_dataset.ext_vars['age']
plt.figure(figsize=(10, 6))
plt.scatter(age, y)
plt.xlabel('Age')
plt.ylabel('MMSE Score')
plt.title('Age vs. Cognitive Function in OASIS Dataset')
plt.grid(True)
plt.show()
```

Parkinson’s Disease

Parkinson’s Progression Markers Initiative (PPMI)

- **Description**: Comprehensive dataset for Parkinson’s disease biomarkers
- **Contents**: Clinical, imaging, genetic, and biospecimen data from 1,400+ subjects
- **Access**: https://www.ppmi-info.org/ (requires application)
- **Documentation**: https://www.ppmi-info.org/access-data-specimens/

```python
# Example: Analyzing PPMI clinical data
# Note: You need approved access to PPMI to download data
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def analyze_ppmi_clinical(clinical_file):
    """Analyze PPMI clinical data from CSV file."""
    # Load clinical data
    df = pd.read_csv(clinical_file)

    # Example exploratory analysis
    print(f"Number of participants: {df['PATNO'].nunique()}")
    print(f"Average age: {df['AGE'].mean():.1f} years")

    # UPDRS (Unified Parkinson's Disease Rating Scale) analysis
    if 'UPDRS_TOT' in df.columns:
        # Distribution of UPDRS scores
        plt.figure(figsize=(10, 6))
        sns.histplot(df['UPDRS_TOT'].dropna(), kde=True)
        plt.xlabel('Total UPDRS Score')
        plt.ylabel('Frequency')
        plt.title('Distribution of UPDRS Scores in PPMI Dataset')
        plt.show()

    # Compare scores between different disease stages
    if 'DIAGNOSIS' in df.columns and 'UPDRS_TOT' in df.columns:
        plt.figure(figsize=(10, 6))
        sns.boxplot(x='DIAGNOSIS', y='UPDRS_TOT', data=df)
        plt.xlabel('Diagnosis Group')
        plt.ylabel('UPDRS Score')
        plt.title('UPDRS Scores by Diagnosis Group')
        plt.show()

    return df

# Example of accessing DaTscan images
def visualize_datscan(datscan_file):
    """Visualize a DaTscan SPECT image from PPMI."""
    # Load DaTscan image
    img = nib.load(datscan_file)
    data = img.get_fdata()

    # Get middle slice
    slice_idx = data.shape[2] // 2

    # Display
    plt.figure(figsize=(8, 8))
    plt.imshow(data[:, :, slice_idx], cmap='hot')
    plt.colorbar(label='Intensity')
    plt.title('DaTscan SPECT Image (Axial Slice)')
    plt.axis('off')
    plt.show()
```

```
# Usage example (pseudocode)
# clinical_data = analyze_ppmi_clinical('/path/to/ppmi/clinical.csv')
# visualize_datscan('/path/to/ppmi/datscan.nii')
```

PhysioNet Freeze-Gait in Parkinson’s Disease

- **Description**: Gait data for patients with and without freezing of gait
- **Contents**: Accelerometer data from 16 PD patients during various activities
- **Access**: https://physionet.org/content/freeze-gait-pd/1.0.0/
- **Documentation**: https://physionet.org/content/freeze-gait-pd/1.0.0/

```python
# Example: Analyzing freezing of gait data from PhysioNet
# !pip install wfdb
import wfdb
import numpy as np
import matplotlib.pyplot as plt
from scipy import signal

# Path to the dataset (update with your path or download)
# data_path = '/path/to/physionet/freeze-gait-pd/1.0.0/'
data_path = 'freeze-gait-pd/1.0.0/'

def analyze_freeze_gait(record_name, data_path):
    """Analyze a freeze of gait recording."""
    try:
        # Load the data
        record = wfdb.rdrecord(data_path + record_name)
        signals = record.p_signal

        # Extract the acceleration channels
        # Typically channels are x, y, z acceleration
        x_acc = signals[:, 0]
        y_acc = signals[:, 1]
        z_acc = signals[:, 2]

        # Time vector
        fs = record.fs  # Sampling frequency
        t = np.arange(len(x_acc)) / fs

        # Compute the frequency spectrum
        f, Pxx_x = signal.welch(x_acc, fs, nperseg=1024)
        _, Pxx_y = signal.welch(y_acc, fs, nperseg=1024)
        _, Pxx_z = signal.welch(z_acc, fs, nperseg=1024)

        # Plot time domain signals
        plt.figure(figsize=(15, 10))

        plt.subplot(2, 1, 1)
        plt.plot(t, x_acc, label='X')
        plt.plot(t, y_acc, label='Y')
        plt.plot(t, z_acc, label='Z')
        plt.xlabel('Time (s)')
        plt.ylabel('Acceleration')
        plt.title(f'Acceleration Data - {record_name}')
        plt.legend()
        plt.grid(True)

        # Plot frequency domain - useful for detecting freezing frequency bands
        plt.subplot(2, 1, 2)
        plt.semilogy(f, Pxx_x, label='X')
        plt.semilogy(f, Pxx_y, label='Y')
        plt.semilogy(f, Pxx_z, label='Z')
        plt.xlabel('Frequency (Hz)')
        plt.ylabel('PSD (g²/Hz)')
        plt.title('Power Spectral Density')
        plt.axvspan(3, 8, color='r', alpha=0.3, label='Freezing Band')
        plt.legend()
        plt.grid(True)

        plt.tight_layout()
        plt.show()

        # Calculate freezing index
        # Freezing Index = Power in freezing band (3-8 Hz) / Power in locomotion bar
        freeze_band = (f >= 3) & (f <= 8)
        locomotion_band = (f >= 0.5) & (f <= 3)

        freeze_power = np.mean(Pxx_y[freeze_band])
        locomotion_power = np.mean(Pxx_y[locomotion_band])

        freezing_index = freeze_power / locomotion_power if locomotion_power > 0 els

        print(f"Freezing Index: {freezing_index:.2f}")

        return freezing_index

    except Exception as e:
        print(f"Error processing record {record_name}: {e}")
        return None

# Example usage (uncomment to use)
# analyze_freeze_gait('S01R01', data_path)
```

Epilepsy

Temple University EEG Corpus

- **Description**: Massive clinical EEG dataset from Temple University Hospital
- **Contents**: 30,000+ EEG recordings from 16,000+ patients, including many with epilepsy
- **Access**: https://isip.piconepress.com/projects/tuh_eeg/
- **Documentation**: https://isip.piconepress.com/projects/tuh_eeg/html/downloads.shtml

```
# Example: Working with Temple University EEG data
# !pip install pyedflib numpy matplotlib
import pyedflib
import numpy as np
import matplotlib.pyplot as plt

def read_eeg_file(edf_file):
    """Read and plot EEG data from EDF file."""
    try:
        # Open the EDF file
        f = pyedflib.EdfReader(edf_file)

        # Get information about the file
        n_channels = f.signals_in_file
        channel_names = f.getSignalLabels()
        fs = f.getSampleFrequency(0)  # Assuming same sampling rate for all channels

        print(f"File: {edf_file}")
        print(f"Number of channels: {n_channels}")
        print(f"Channel names: {channel_names}")
        print(f"Sampling frequency: {fs} Hz")

        # Read a subset of channels (e.g., first 8)
        n_display = min(8, n_channels)
        signals = np.zeros((n_display, f.getNSamples()[0]))

        for i in range(n_display):
            signals[i, :] = f.readSignal(i)

        # Calculate time vector
        seconds = len(signals[0]) / fs
        t = np.linspace(0, seconds, len(signals[0]))

        # Plot the signals
        plt.figure(figsize=(15, 10))

        # Default display: 10 seconds window with appropriate scaling
        window_size = min(10, seconds)
        samples_to_display = int(window_size * fs)

        for i in range(n_display):
            plt.subplot(n_display, 1, i+1)

            # Z-score normalize for display
            signal_std = np.std(signals[i][:samples_to_display])
            signal_mean = np.mean(signals[i][:samples_to_display])
            normalized_signal = (signals[i][:samples_to_display] - signal_mean) / (s

            plt.plot(t[:samples_to_display], normalized_signal)
            plt.ylabel(channel_names[i])
            plt.xlim(0, window_size)

            # Only show x-axis for bottom plot
```

```python
            if i < n_display - 1:
                plt.xticks([])

        plt.xlabel('Time (s)')
        plt.tight_layout()
        plt.show()

        # Close the file
        f.close()

        return signals, channel_names, fs

    except Exception as e:
        print(f"Error reading EEG file: {e}")
        return None, None, None

# Example seizure detection function
def detect_seizures(eeg_signal, fs, window_size=1.0, overlap=0.5):
    """
    Basic seizure detection based on signal energy and line length.

    Parameters:
    eeg_signal: EEG signal (single channel)
    fs: Sampling frequency
    window_size: Analysis window size in seconds
    overlap: Window overlap fraction

    Returns:
    detection_times: List of potential seizure onset times
    """
    # Convert window size from seconds to samples
    window_samples = int(window_size * fs)
    step_samples = int(window_samples * (1 - overlap))

    # Features
    energy = []
    line_length = []

    # Process windows
    for i in range(0, len(eeg_signal) - window_samples, step_samples):
        window = eeg_signal[i:i + window_samples]

        # Calculate energy (sum of squared amplitudes)
        energy.append(np.sum(window**2) / window_samples)

        # Calculate line length (sum of absolute differences)
        line_length.append(np.sum(np.abs(np.diff(window))) / window_samples)

    # Normalize features
    energy = np.array(energy)
    energy_norm = (energy - np.mean(energy)) / (np.std(energy) if np.std(energy) > 0

    line_length = np.array(line_length)
    ll_norm = (line_length - np.mean(line_length)) / (np.std(line_length) if np.std(
    # Combined feature
    combined = energy_norm + ll_norm

    # Threshold for detection (this is a simple approach, real detection would use m
    threshold = 3.0

    # Find potential seizures
    detection_indices = np.where(combined > threshold)[0]

    # Convert indices to times
    detection_times = []
    if len(detection_indices) > 0:
        # Group consecutive detections
        groups = np.split(detection_indices, np.where(np.diff(detection_indices) !=

        for group in groups:
            if len(group) > 0:
                # Convert from window index to time
                onset_time = (group[0] * step_samples) / fs
                detection_times.append(onset_time)

    return detection_times, combined

# Example usage (pseudocode)
# signals, channel_names, fs = read_eeg_file('/path/to/tuh_eeg/patient/eeg.edf')
# if signals is not None:
#     # Analyze a specific channel (e.g., channel index 2)
#     channel_idx = 2
#     detection_times, feature = detect_seizures(signals[channel_idx], fs)
#     print(f"Potential seizure onsets at: {detection_times} seconds")
```

Stroke

International Stroke Perfusion Imaging Registry (INSPIRE)

- **Description**: Multi-center dataset for stroke imaging
- **Contents**: CT, MRI, and clinical data from stroke patients
- **Access**: https://inspire.app/
- **Documentation**: Available upon registration

```python
# Example: Simulated analysis of stroke perfusion images
import numpy as np
import matplotlib.pyplot as plt
from skimage import filters, measure
import nibabel as nib

def analyze_stroke_perfusion(perfusion_file, t_max_threshold=6.0):
    """
    Analyze perfusion imaging (Tmax map) to identify potential stroke penumbra.

    Parameters:
    perfusion_file: Path to the perfusion Tmax map
    t_max_threshold: Threshold in seconds for penumbra definition

    Returns:
    penumbra_volume: Volume of tissue with Tmax > threshold
    """
    # Load the perfusion map
    img = nib.load(perfusion_file)
    tmax_data = img.get_fdata()

    # Get voxel dimensions to calculate volume
    voxel_size = np.prod(img.header.get_zooms())

    # Threshold the image to identify penumbra
    penumbra_mask = tmax_data > t_max_threshold

    # Calculate penumbra volume in ml
    penumbra_volume = np.sum(penumbra_mask) * voxel_size / 1000.0

    # Get a middle slice for visualization
    slice_idx = tmax_data.shape[2] // 2

    # Visualize the perfusion map and penumbra
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))

    # Perfusion map
    im1 = ax1.imshow(tmax_data[:, :, slice_idx], cmap='jet')
    ax1.set_title('Tmax Perfusion Map')
    ax1.axis('off')
    plt.colorbar(im1, ax=ax1, label='Tmax (s)')

    # Penumbra overlay
    ax2.imshow(tmax_data[:, :, slice_idx], cmap='gray')
    penumbra_overlay = np.ma.masked_where(
        ~penumbra_mask[:, :, slice_idx],
        np.ones_like(tmax_data[:, :, slice_idx])
    )
    ax2.imshow(penumbra_overlay, cmap='autumn', alpha=0.7)
    ax2.set_title(f'Penumbra (Tmax > {t_max_threshold}s)')
    ax2.axis('off')

    plt.tight_layout()
    plt.show()

    print(f"Estimated penumbra volume: {penumbra_volume:.2f} ml")

    return penumbra_volume

# Example of decision support for stroke treatment
def stroke_treatment_decision(penumbra_volume, core_volume, onset_time):
    """
    Simple decision support for stroke treatment based on imaging and clinical facto

    Parameters:
    penumbra_volume: Volume of salvageable tissue (ml)
    core_volume: Volume of infarcted core (ml)
    onset_time: Time since stroke onset (hours)

    Returns:
    recommendation: Treatment recommendation
    """
    mismatch_ratio = penumbra_volume / core_volume if core_volume > 0 else float('in

    # Decision logic (simplified for demonstration)
    if onset_time <= 4.5:
        if core_volume < 70:
            recommendation = "Consider IV thrombolysis"
        else:
            recommendation = "Large core infarct. Consider mechanical thrombectomy e
    elif onset_time <= 24:
        if mismatch_ratio >= 1.8 and penumbra_volume >= 15:
            recommendation = "Consider extended window mechanical thrombectomy"
        else:
            recommendation = "Insufficient penumbra for endovascular treatment"
    else:
        recommendation = "Outside treatment window"

    # Visualization
    plt.figure(figsize=(10, 6))
    plt.bar(['Core', 'Penumbra'], [core_volume, penumbra_volume], color=['red', 'ora
    plt.axhline(y=70, linestyle='--', color='black', label='Core Volume Threshold (7
    plt.ylabel('Volume (ml)')
    plt.title(f'Stroke Tissue Volumes\nMismatch Ratio: {mismatch_ratio:.1f}, Time fr
    plt.legend()
    plt.show()

    print(f"Treatment recommendation: {recommendation}")

    return recommendation

# Usage example (pseudocode)
# penumbra_volume = analyze_stroke_perfusion('/path/to/tmax.nii.gz')
# stroke_treatment_decision(penumbra_volume=45, core_volume=15, onset_time=3.5)
```

Multiple Sclerosis

Multiple Sclerosis Lesion Segmentation Challenge Dataset

- **Description**: Public dataset for evaluating MS lesion segmentation algorithms
- **Contents**: MRI scans with manual lesion segmentations from multiple centers
- **Access**: https://smart-stats-tools.org/lesion-challenge
- **Documentation**: https://smart-stats-tools.org/lesion-challenge-dataset

```python
# Example: MS lesion segmentation
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
from skimage import measure, filters

def segment_ms_lesions(flair_file, t1_file, t2_file, threshold_factor=2.5):
    """
    Simple MS lesion segmentation using FLAIR hyperintensities.

    Parameters:
    flair_file: Path to FLAIR MRI
    t1_file: Path to T1 MRI
    t2_file: Path to T2 MRI
    threshold_factor: Multiplier for intensity threshold

    Returns:
    lesion_mask: Binary mask of detected lesions
    """
    # Load MRI sequences
    flair_img = nib.load(flair_file)
    flair_data = flair_img.get_fdata()

    t1_data = nib.load(t1_file).get_fdata()
    t2_data = nib.load(t2_file).get_fdata()

    # Simple lesion segmentation based on FLAIR hyperintensities
    # Real segmentation would use more sophisticated methods

    # Calculate brain mask (simple thresholding)
    brain_mask = t1_data > np.percentile(t1_data, 15)

    # Threshold FLAIR within brain mask
    flair_brain = flair_data * brain_mask
    flair_threshold = np.mean(flair_brain[flair_brain > 0]) + threshold_factor * np.

    # Initial lesion segmentation
    lesion_mask = (flair_data > flair_threshold) & brain_mask

    # Remove small lesions (noise)
    # Label connected components
    labels = measure.label(lesion_mask)

    # Calculate properties for each lesion candidate
    props = measure.regionprops(labels)

    # Filter by size (remove very small objects)
    min_size = 10  # voxels
    lesion_mask = np.zeros_like(lesion_mask)

    for prop in props:
        if prop.area >= min_size:
            lesion_mask[labels == prop.label] = 1
    # Visualize results (middle slice)
    slice_idx = flair_data.shape[2] // 2

    fig, axes = plt.subplots(2, 2, figsize=(15, 12))

    # FLAIR image
    axes[0, 0].imshow(flair_data[:, :, slice_idx], cmap='gray')
    axes[0, 0].set_title('FLAIR Image')
    axes[0, 0].axis('off')

    # T1 image
    axes[0, 1].imshow(t1_data[:, :, slice_idx], cmap='gray')
    axes[0, 1].set_title('T1 Image')
    axes[0, 1].axis('off')

    # T2 image
    axes[1, 0].imshow(t2_data[:, :, slice_idx], cmap='gray')
    axes[1, 0].set_title('T2 Image')
    axes[1, 0].axis('off')

    # FLAIR with lesion overlay
    axes[1, 1].imshow(flair_data[:, :, slice_idx], cmap='gray')
    lesion_overlay = np.ma.masked_where(
        ~lesion_mask[:, :, slice_idx],
        np.ones_like(flair_data[:, :, slice_idx])
    )
    axes[1, 1].imshow(lesion_overlay, cmap='hot', alpha=0.7)
    axes[1, 1].set_title('Lesion Segmentation')
    axes[1, 1].axis('off')

    plt.tight_layout()
    plt.show()

    # Calculate total lesion volume
    voxel_size = np.prod(flair_img.header.get_zooms())
    lesion_volume = np.sum(lesion_mask) * voxel_size / 1000.0  # ml

    print(f"Estimated MS lesion volume: {lesion_volume:.2f} ml")

    return lesion_mask, lesion_volume

# Example of lesion load progression analysis
def analyze_lesion_progression(lesion_volumes, scan_dates):
    """Analyze progression of MS lesion volumes over time."""
    # Convert dates to numeric (days since first scan)
    from datetime import datetime
    dates = [datetime.strptime(d, '%Y-%m-%d') for d in scan_dates]
    days = [(d - dates[0]).days for d in dates]

    # Visualize progression
    plt.figure(figsize=(10, 6))
    plt.plot(days, lesion_volumes, 'o-', linewidth=2)
    plt.xlabel('Days from Baseline')
    plt.ylabel('Lesion Volume (ml)')
    plt.title('MS Lesion Volume Progression')
    plt.grid(True)

    # Calculate rate of change
    if len(lesion_volumes) > 1:
        annual_change = 365.0 * (lesion_volumes[-1] - lesion_volumes[0]) / days[-1]
        plt.axline((0, lesion_volumes[0]), slope=annual_change/365.0,
                  color='r', linestyle='--', label=f'Trend: {annual_change:.2f} ml/y
        plt.legend()

    plt.show()

# Usage example (pseudocode)
# lesion_mask, volume = segment_ms_lesions(
#     '/path/to/flair.nii.gz',
#     '/path/to/t1.nii.gz',
#     '/path/to/t2.nii.gz'
# )

# Lesion progression analysis example
# analyze_lesion_progression(
#     [10.5, 12.3, 15.7, 14.9, 18.2],
#     ['2020-01-15', '2020-07-20', '2021-01-10', '2021-07-05', '2022-01-12']
# )
```

Other Neurological Disorders

Brain Tumor Segmentation (BraTS) Challenge

- **Description**: Multimodal brain tumor MRI dataset
- **Contents**: T1, T1ce, T2, and FLAIR MRIs with expert segmentations
- **Access**: http://braintumorsegmentation.org/
- **Documentation**: https://www.med.upenn.edu/cbica/brats2020/data.html

```python
# Example: Brain tumor visualization and segmentation
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt

def visualize_brain_tumor(t1_file, t1ce_file, t2_file, flair_file, seg_file=None):
    """
    Visualize multimodal brain tumor MRI data.

    Parameters:
    t1_file: Path to T1 MRI
    t1ce_file: Path to T1 contrast-enhanced MRI
    t2_file: Path to T2 MRI
    flair_file: Path to FLAIR MRI
    seg_file: Path to segmentation (optional)
    """
    # Load MRI data
    t1 = nib.load(t1_file).get_fdata()
    t1ce = nib.load(t1ce_file).get_fdata()
    t2 = nib.load(t2_file).get_fdata()
    flair = nib.load(flair_file).get_fdata()

    # Load segmentation if available
    if seg_file:
        seg = nib.load(seg_file).get_fdata()
        has_seg = True
    else:
        has_seg = False

    # Get the middle slice for each dimension
    axial_slice = t1.shape[2] // 2

    # Create a figure
    fig, axes = plt.subplots(2, 3 if has_seg else 2, figsize=(15, 10))

    # Display T1
    axes[0, 0].imshow(t1[:, :, axial_slice], cmap='gray')
    axes[0, 0].set_title('T1')
    axes[0, 0].axis('off')

    # Display T1ce
    axes[0, 1].imshow(t1ce[:, :, axial_slice], cmap='gray')
    axes[0, 1].set_title('T1ce')
    axes[0, 1].axis('off')

    # Display T2
    axes[1, 0].imshow(t2[:, :, axial_slice], cmap='gray')
    axes[1, 0].set_title('T2')
    axes[1, 0].axis('off')

    # Display FLAIR
    axes[1, 1].imshow(flair[:, :, axial_slice], cmap='gray')
    axes[1, 1].set_title('FLAIR')
    axes[1, 1].axis('off')

    # Display segmentation if available
    if has_seg:
        axes[0, 2].imshow(t1ce[:, :, axial_slice], cmap='gray')

        # Create a colored overlay for different tumor components
        # In BraTS: 1=necrotic core, 2=edema, 4=enhancing tumor
        seg_slice = seg[:, :, axial_slice]

        # Necrotic core (red)
        necrotic = np.ma.masked_where(seg_slice != 1, np.ones_like(seg_slice))
        axes[0, 2].imshow(necrotic, cmap='Reds', alpha=0.7)

        # Edema (green)
        edema = np.ma.masked_where(seg_slice != 2, np.ones_like(seg_slice))
        axes[0, 2].imshow(edema, cmap='Greens', alpha=0.7)

        # Enhancing tumor (blue)
        enhancing = np.ma.masked_where(seg_slice != 4, np.ones_like(seg_slice))
        axes[0, 2].imshow(enhancing, cmap='Blues', alpha=0.7)

        axes[0, 2].set_title('Tumor Segmentation')
        axes[0, 2].axis('off')

        # Calculate tumor volumes
        voxel_size = 1.0  # mm³, adjust based on actual voxel size
        necrotic_vol = np.sum(seg == 1) * voxel_size / 1000.0  # ml
        edema_vol = np.sum(seg == 2) * voxel_size / 1000.0  # ml
        enhancing_vol = np.sum(seg == 4) * voxel_size / 1000.0  # ml
        total_vol = necrotic_vol + edema_vol + enhancing_vol

        # Display volumes
        text = (f"Tumor Components:\n"
                f"Necrotic core: {necrotic_vol:.2f} ml\n"
                f"Edema: {edema_vol:.2f} ml\n"
                f"Enhancing tumor: {enhancing_vol:.2f} ml\n"
                f"Total volume: {total_vol:.2f} ml")

        axes[1, 2].axis('off')
        axes[1, 2].text(0.1, 0.5, text, fontsize=12)

    plt.tight_layout()
    plt.show()

# Usage example (pseudocode)
# visualize_brain_tumor(
#     '/path/to/t1.nii.gz',
#     '/path/to/t1ce.nii.gz',
#     '/path/to/t2.nii.gz',
#     '/path/to/flair.nii.gz',
#     '/path/to/seg.nii.gz'
# )
```

Sleep-EDF Database

- **Description**: Sleep recordings with annotations for normal and abnormal sleep
- **Contents**: PSG and EEG recordings with sleep stage annotations
- **Access**: https://physionet.org/content/sleep-edfx/1.0.0/
- **Documentation**: https://physionet.org/content/sleep-edfx/1.0.0/

```python
# Example: Sleep stage classification from EEG
import numpy as np
import matplotlib.pyplot as plt
import pyedflib
from scipy import signal

def analyze_sleep_edf(edf_file, hypnogram_file):
    """
    Analyze Sleep-EDF data with sleep stage annotations.

    Parameters:
    edf_file: Path to EEG recording
    hypnogram_file: Path to sleep stage annotations
    """
    # Load EEG data
    try:
        f = pyedflib.EdfReader(edf_file)

        # Get signal information
        n_channels = f.signals_in_file
        channel_names = f.getSignalLabels()

        # Find EEG channels (typically Fpz-Cz and Pz-Oz in Sleep-EDF)
        eeg_channels = [i for i, name in enumerate(channel_names) if 'EEG' in name]

        if not eeg_channels:
            print("No EEG channels found")
            return

        # Read EEG data from first EEG channel
        eeg_channel = eeg_channels[0]
        fs = f.getSampleFrequency(eeg_channel)
        eeg_signal = f.readSignal(eeg_channel)

        # Load hypnogram (sleep stages)
        with open(hypnogram_file, 'r') as hyp_file:
            hypnogram = hyp_file.readlines()

        # Parse sleep stages (simplified, actual parsing depends on hypnogram format
        # In Sleep-EDF annotations: 0=Wake, 1=REM, 2=N1, 3=N2, 4=N3/N4
        sleep_stages = []
        for line in hypnogram:
            if line.strip() and line[0].isdigit():
                sleep_stages.append(int(line[0]))

        sleep_stages = np.array(sleep_stages)

        # Calculate time vectors
        eeg_time = np.arange(len(eeg_signal)) / fs  # seconds
        stage_time = np.arange(len(sleep_stages)) * 30  # 30-second epochs

        # Extract features for visualization
        # (in real application, would extract features for each epoch)
        # Power spectral density (whole signal)
        f, psd = signal.welch(eeg_signal, fs, nperseg=fs*4)

        # Find frequency bands
        delta_mask = (f >= 0.5) & (f <= 4)
        theta_mask = (f >= 4) & (f <= 8)
        alpha_mask = (f >= 8) & (f <= 13)
        beta_mask = (f >= 13) & (f <= 30)

        # Visualize data
        fig, axes = plt.subplots(3, 1, figsize=(15, 12))

        # Plot EEG signal (first 60 seconds)
        plot_duration = min(60, len(eeg_signal) / fs)
        plot_samples = int(plot_duration * fs)

        axes[0].plot(eeg_time[:plot_samples], eeg_signal[:plot_samples])
        axes[0].set_xlabel('Time (s)')
        axes[0].set_ylabel('Amplitude')
        axes[0].set_title(f'EEG Signal ({channel_names[eeg_channel]})')

        # Plot sleep stages
        stage_labels = ['Wake', 'REM', 'N1', 'N2', 'N3/N4']

        axes[1].step(stage_time / 3600, sleep_stages, where='post')
        axes[1].set_yticks(range(5))
        axes[1].set_yticklabels(stage_labels)
        axes[1].set_xlabel('Time (hours)')
        axes[1].set_ylabel('Sleep Stage')
        axes[1].set_title('Hypnogram')
        axes[1].grid(True)

        # Plot power spectral density
        axes[2].semilogy(f, psd)
        axes[2].set_xlabel('Frequency (Hz)')
        axes[2].set_ylabel('PSD (μV²/Hz)')
        axes[2].set_title('Power Spectral Density')

        # Highlight frequency bands
        axes[2].fill_between(f[delta_mask], 0, psd[delta_mask], alpha=0.3, label='De
        axes[2].fill_between(f[theta_mask], 0, psd[theta_mask], alpha=0.3, label='Th
        axes[2].fill_between(f[alpha_mask], 0, psd[alpha_mask], alpha=0.3, label='Al
        axes[2].fill_between(f[beta_mask], 0, psd[beta_mask], alpha=0.3, label='Beta
        axes[2].legend()

        plt.tight_layout()
        plt.show()

        # Calculate sleep metrics
        total_epochs = len(sleep_stages)
        total_sleep_time = sum(sleep_stages != 0) * 30 / 60  # minutes

        # Sleep efficiency (total sleep time / time in bed)
```

```
        sleep_efficiency = sum(sleep_stages != 0) / total_epochs * 100

        # Time in each stage
        stage_minutes = {
            stage_labels[i]: sum(sleep_stages == i) * 30 / 60
            for i in range(5)
        }

        # Print sleep metrics
        print(f"Sleep Metrics:")
        print(f"Total recording time: {total_epochs * 30 / 60:.1f} minutes")
        print(f"Total sleep time: {total_sleep_time:.1f} minutes")
        print(f"Sleep efficiency: {sleep_efficiency:.1f}%")
        print("Time in each stage:")
        for stage, minutes in stage_minutes.items():
            print(f"  {stage}: {minutes:.1f} minutes ({minutes/total_sleep_time*100:

        f.close()

    except Exception as e:
        print(f"Error analyzing Sleep-EDF data: {e}")

# Usage example (pseudocode)
# analyze_sleep_edf('/path/to/sleep_edf.edf', '/path/to/hypnogram.txt')
```

These neurological and neurodegenerative disease datasets provide valuable resources for developing and validating AI systems for diagnosis, treatment planning, and monitoring disease progression. The examples demonstrate basic approaches to working with these diverse data types and highlight the potential for AI applications in clinical neurology.