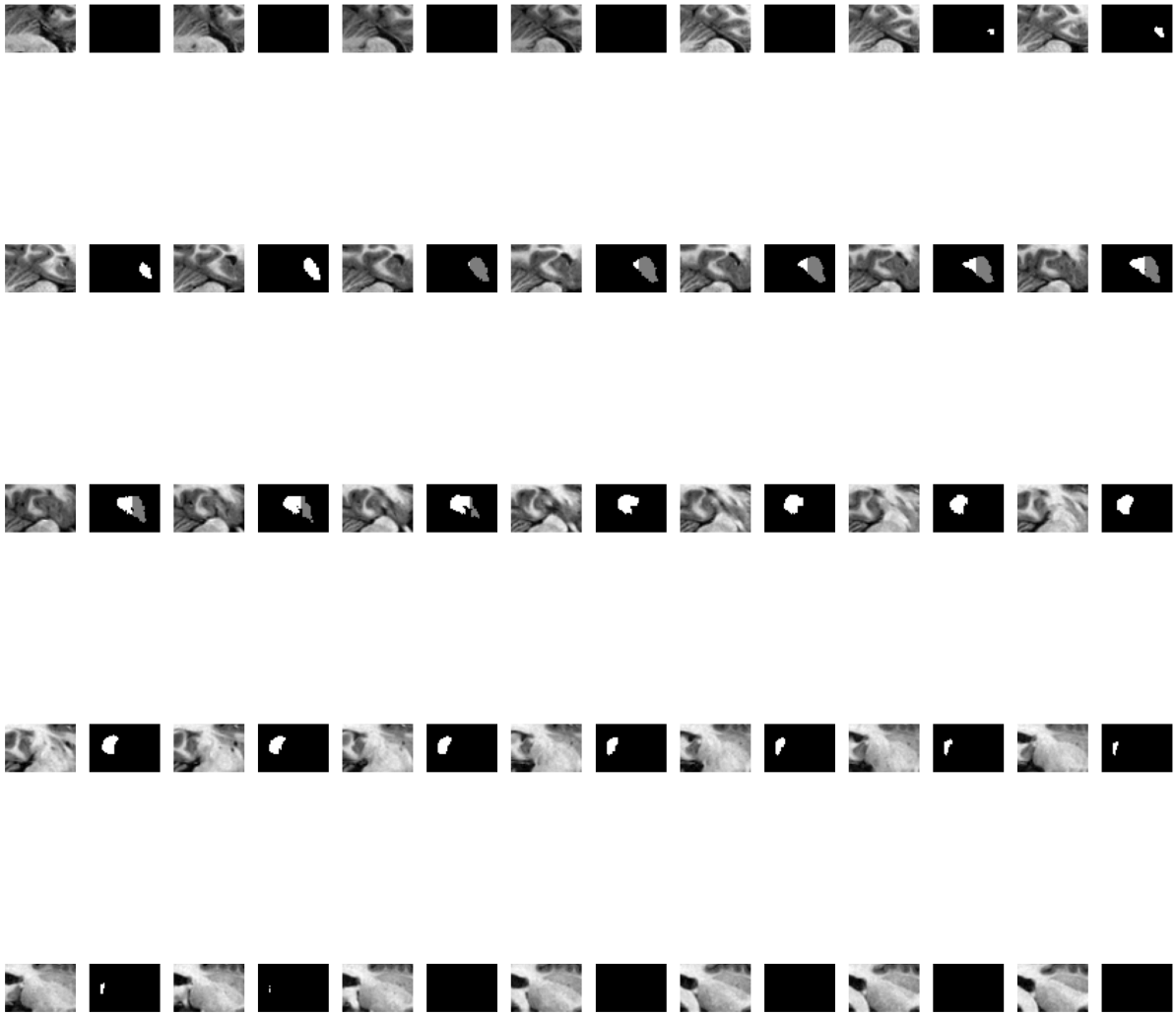
In [1]:
```
import os
import glob
import nibabel as nib
import numpy as np
import matplotlib.pyplot as plt
import shutil


# Set up inline plotting
%matplotlib inline

# Define the path to your dataset
data_path = '/Users/tina/Desktop/Projects/nd320-c3-3d-imaging-starter/data/T
images_path = os.path.join(data_path, 'images')
labels_path = os.path.join(data_path, 'labels')

# Function to verify and list files, excluding any unwanted files like '.DS_
def verify_and_list_files(directory):
    try:
        files = sorted(f for f in os.listdir(directory) if not f.startswith(
        print(f"Found {len(files)} files in {directory}.")
        return files
    except FileNotFoundError:
        print(f"Error: Directory {directory} does not exist.")
        return []

image_files = verify_and_list_files(images_path)
label_files = verify_and_list_files(labels_path)

# Match images and labels based on filenames
def match_files(images, labels):
    matched_images = []
    matched_labels = []
    unmatched = []

    for img in images:
        img_name = os.path.basename(img)
        label_path = os.path.join(labels_path, img_name)
        if img_name in labels:
            matched_images.append(os.path.join(images_path, img_name))
            matched_labels.append(label_path)
        else:
            unmatched.append(img_name)
            print(f"Unmatched image: {img_name}")

    return matched_images, matched_labels, unmatched

image_files, label_files, unmatched_images = match_files(image_files, label_

# Filter out outlier images based on shape threshold
def filter_outliers(image_list, threshold=60):
    outliers = []
    for img in image_list:
        img_data = nib.load(img).get_fdata()
        if any(dim > threshold for dim in img_data.shape):
            outliers.append(img)
            print(f"Outlier detected: {img} with shape {img_data.shape}")
    return [img for img in image_list if img not in outliers]

image_files = filter_outliers(image_files)

# Load and display an image and its corresponding label
```

```
def display_image_and_label(image_path, label_path, num_slices=35):
    image = nib.load(image_path).get_fdata()
    label = nib.load(label_path).get_fdata()

    fig, ax = plt.subplots(num_slices // 7, 14, figsize=[30, 30])
    for i in range(num_slices):
        if i < image.shape[2]:  # assuming the slices are along the z-axis
            ax[i // 7, 2 * (i % 7)].imshow(image[:, :, i], cmap='gray')
            ax[i // 7, 2 * (i % 7) + 1].imshow(label[:, :, i], cmap='gray')

            ax[i // 7, 2 * (i % 7)].axis("off")
            ax[i // 7, 2 * (i % 7) + 1].axis("off")
    plt.show()

# Check if there are any matched image-label pairs to display
if image_files and label_files:
    display_image_and_label(image_files[0], label_files[0])
else:
    print("No matched image-label pairs available to display.")
```

```
Found 263 files in /Users/tina/Desktop/Projects/nd320-c3-3d-imaging-starte
r/data/TrainingSet/images.
Found 262 files in /Users/tina/Desktop/Projects/nd320-c3-3d-imaging-starte
r/data/TrainingSet/labels.
Unmatched image: hippocampus_118.nii.gz
Outlier detected: /Users/tina/Desktop/Projects/nd320-c3-3d-imaging-starter/
data/TrainingSet/images/hippocampus_010.nii.gz with shape (512, 512, 241)
Outlier detected: /Users/tina/Desktop/Projects/nd320-c3-3d-imaging-starter/
data/TrainingSet/images/hippocampus_281.nii.gz with shape (512, 512, 31)
```



In [2]:
```
# Load the first image to check its header information
img = nib.load(image_files[0])
```

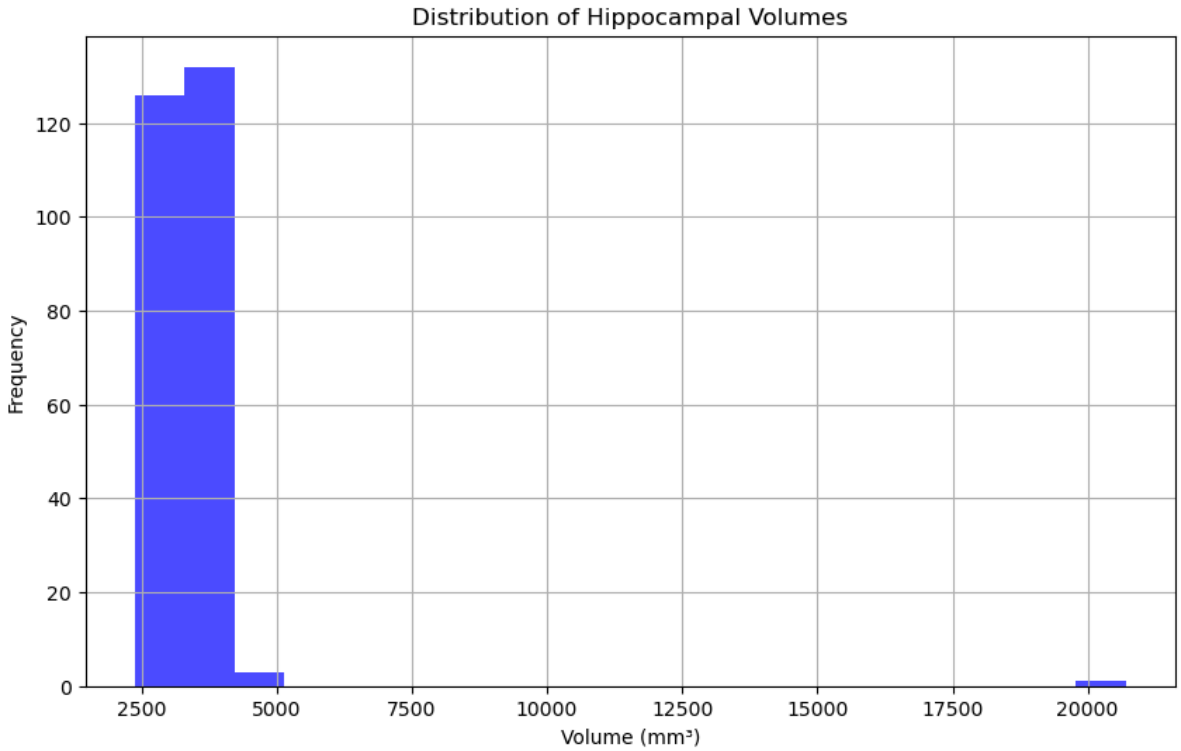
```
# Check the format of the image
print('Format of the image: ', img.header_class)

# Check the number of bits per pixel
print('Bits per pixel: ', img.header['bitpix'])

# Check the units of measurement for the image
unit_code = img.header['xyzt_units']
print('Units of measurement:', unit_code)

# Explanation of the unit code 10
if unit_code == 10:
    print("Value 10 corresponds to 2 | 8:")
    print("Spatial Units (xyz): Millimeters (mm)")
    print("Temporal Units (t): Seconds (s)")

# Check if the grid is regular and the spacing
print('Grid Spacing (pixdim):', img.header['pixdim'])

# Check the dimensions of the image
print('Image Dimensions:', img.header['dim'])

# Check the shape of the image
image_shape = img.shape
print(f"Image shape: {image_shape}")
```

```
Format of the image:  <class 'nibabel.nifti1.Nifti1Header'>
Bits per pixel:  8
Units of measurement: 10
Value 10 corresponds to 2 | 8:
Spatial Units (xyz): Millimeters (mm)
Temporal Units (t): Seconds (s)
Grid Spacing (pixdim): [1. 1. 1. 1. 1. 0. 0. 0.]
Image Dimensions: [ 3 35 51 35  1  1  1  1]
Image shape: (35, 51, 35)
```

In [3]:

```
# Function to calculate the volume of a region in the label
def calculate_volume(label_file):
    label_data = nib.load(label_file).get_fdata()
    voxel_volume = np.prod(img.header['pixdim'][1:4])  # Calculate the volu
    hippocampal_volume = np.sum(label_data != 0) * voxel_volume
    return hippocampal_volume

# Calculate volumes for all labels in the dataset
volumes = [calculate_volume(label) for label in label_files]

# Print the first few volumes as a sanity check
print("Sample Hippocampal Volumes (in mm³):", volumes[:10])

# Plot a histogram of the hippocampal volumes
plt.figure(figsize=(10, 6))
plt.hist(volumes, bins=20, color='blue', alpha=0.7)
plt.title('Distribution of Hippocampal Volumes')
plt.xlabel('Volume (mm³)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

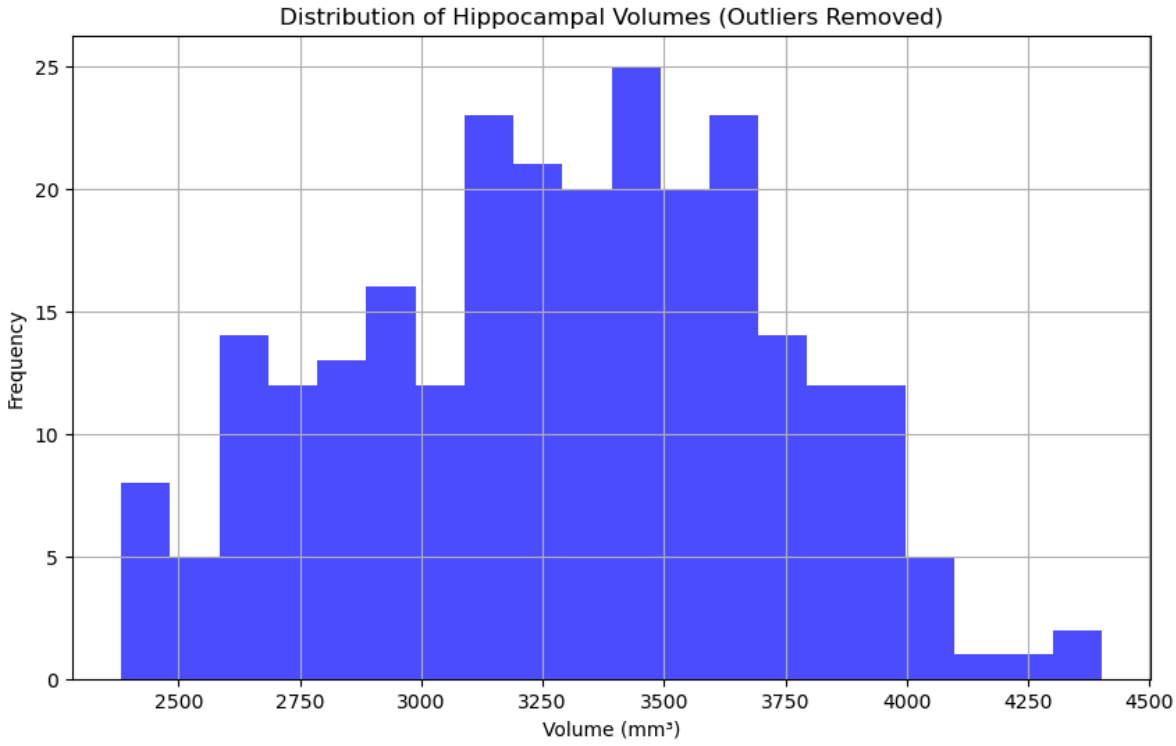
```
Sample Hippocampal Volumes (in mm³): [2948.0, 3353.0, 3698.0, 4263.0, 3372.
0, 3248.0, 3456.0, 3456.0, 3622.0, 2819.0]
```



In [4]:
```python
# Function to calculate the volume of a region in the label
def calculate_volume(label_file):
    label_data = nib.load(label_file).get_fdata()
    voxel_volume = np.prod(img.header['pixdim'][1:4])  # Calculate the volu
    hippocampal_volume = np.sum(label_data != 0) * voxel_volume
    return hippocampal_volume

# Calculate volumes for all labels in the dataset
volumes = [calculate_volume(label) for label in label_files]

# Identify outliers using IQR method
def identify_outliers(volumes):
    q1 = np.percentile(volumes, 25)
    q3 = np.percentile(volumes, 75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    return lower_bound, upper_bound

lower_bound, upper_bound = identify_outliers(volumes)

# Filter out the outliers
filtered_volumes = []
filtered_images = []
filtered_labels = []

for vol, img, label in zip(volumes, image_files, label_files):
    if lower_bound <= vol <= upper_bound:
        filtered_volumes.append(vol)
        filtered_images.append(img)
        filtered_labels.append(label)

print(f"Removed {len(volumes) - len(filtered_volumes)} outliers.")

# Plot the histogram of the filtered hippocampal volumes
plt.figure(figsize=(10, 6))
plt.hist(filtered_volumes, bins=20, color='blue', alpha=0.7)
plt.title('Distribution of Hippocampal Volumes (Outliers Removed)')
plt.xlabel('Volume (mm³)')
```

```
plt.ylabel('Frequency')
plt.grid(True)
plt.show()

# Display the number of images after filtering
print(f"Remaining images: {len(filtered_images)}")
```

```
Removed 3 outliers.
```




```
Remaining images: 259
```

In [5]:
```
# TASK: Copy the cleaned dataset to the output folder inside section1/out
output_images_dir = '/Users/tina/Desktop/Projects/nd320-c3-3d-imaging-starte
output_labels_dir = '/Users/tina/Desktop/Projects/nd320-c3-3d-imaging-starte

# Ensure destination directories exist
os.makedirs(output_images_dir, exist_ok=True)
os.makedirs(output_labels_dir, exist_ok=True)

# Copy the cleaned images and labels to the output directories
for img_file in filtered_images:
    shutil.copy(img_file, os.path.join(output_images_dir, os.path.basename(i

for lbl_file in filtered_labels:
    shutil.copy(lbl_file, os.path.join(output_labels_dir, os.path.basename(l

print("Cleaned dataset has been copied to the output folder.")
```

```
Cleaned dataset has been copied to the output folder.
```

In [ ]:

In [ ]: