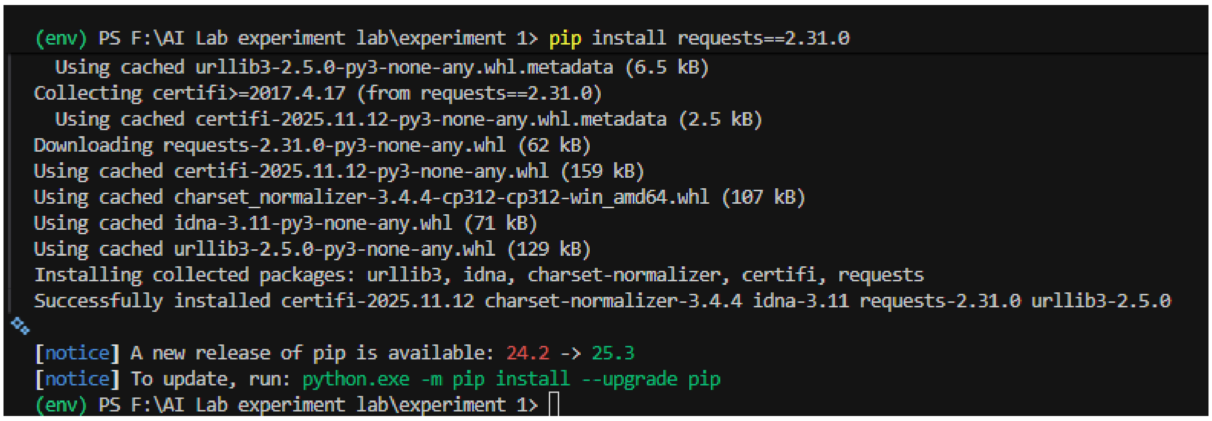
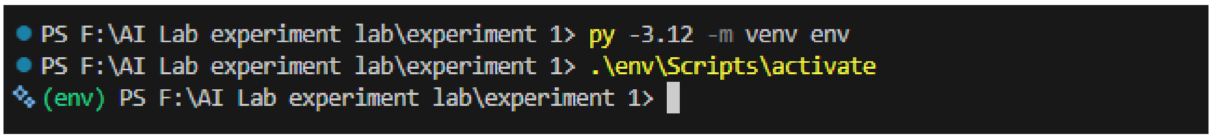
Task 1: Network Crawling Images

Technology sued:
Language : Python 3.12
Library: requests library version 2.31.0



```
(env) PS F:\AI Lab experiment lab\experiment 1> pip install requests==2.31.0
  Using cached urllib3-2.5.0-py3-none-any.whl.metadata (6.5 kB)
Collecting certifi>=2017.4.17 (from requests==2.31.0)
  Using cached certifi-2025.11.12-py3-none-any.whl.metadata (2.5 kB)
Downloading requests-2.31.0-py3-none-any.whl (62 kB)
Using cached certifi-2025.11.12-py3-none-any.whl (159 kB)
Using cached charset_normalizer-3.4.4-cp312-cp312-win_amd64.whl (107 kB)
Using cached idna-3.11-py3-none-any.whl (71 kB)
Using cached urllib3-2.5.0-py3-none-any.whl (129 kB)
Installing collected packages: urllib3, idna, charset-normalizer, certifi, requests
Successfully installed certifi-2025.11.12 charset-normalizer-3.4.4 idna-3.11 requests-2.31.0 urllib3-2.5.0

[notice] A new release of pip is available: 24.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
(env) PS F:\AI Lab experiment lab\experiment 1>
```

Environment setup for python 3.12.x

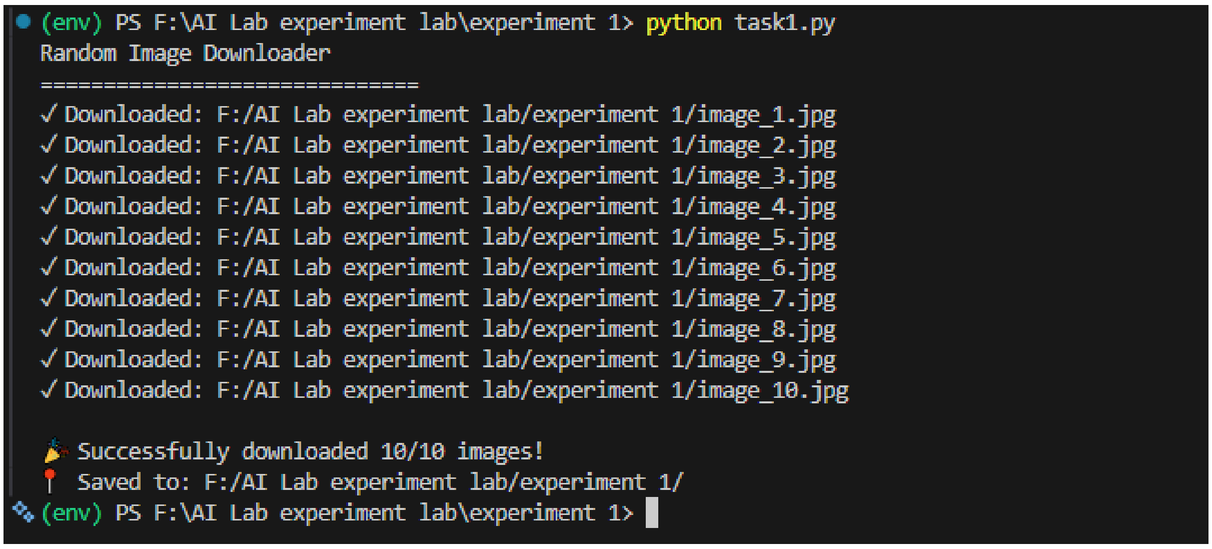


```
PS F:\AI Lab experiment lab\experiment 1> py -3.12 -m venv env
PS F:\AI Lab experiment lab\experiment 1> .\env\Scripts\activate
(env) PS F:\AI Lab experiment lab\experiment 1>
```

task1.py:

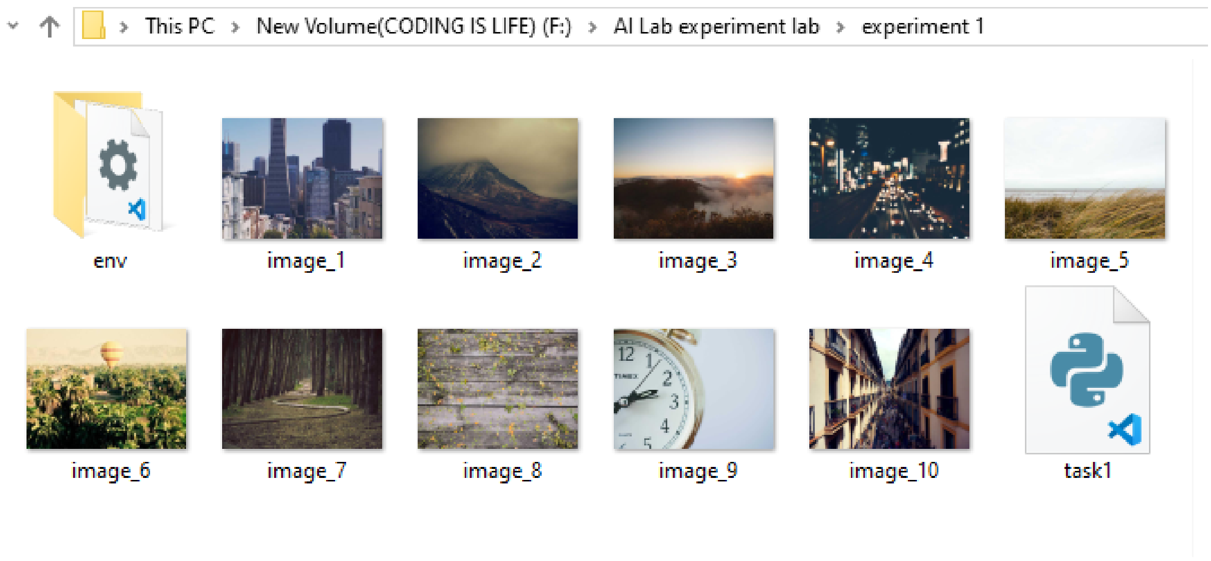
```
import requests
import os
from contextlib import closing

# ------------------------------
# Configuration
# ------------------------------
HEADERS = {
        'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36'
}

SAVE_PATH = 'F:/AI Lab experiment lab/experiment 1/'

# ------------------------------
# Download functions
# ------------------------------
def download_image(img_url, filename):
    """Download a single image"""
    try:
```

```python
            with closing(requests.get(img_url, headers=HEADERS, stream=True)) as resp:
            resp.raise_for_status()
            with open(filename, 'wb') as f:
                for chunk in resp.iter_content(128):
                    f.write(chunk)
        print(f"✓ Downloaded: {filename}")
        return True
    except Exception as e:
        print(f"✗ Failed to download {filename}: {e}")
        return False

def download_multiple_images(num_images=10):
    """Download multiple random images from Picsum"""
    # Create save directory
    if not os.path.exists(SAVE_PATH):
        os.makedirs(SAVE_PATH)
        print(f"Created directory: {SAVE_PATH}")

    success_count = 0

    for i in range(1, num_images + 1):
        # Picsum provides random images with different dimensions
        img_url = f"https://picsum.photos/800/600?random={i}"
        filename = f"{SAVE_PATH}image_{i}.jpg"

        if download_image(img_url, filename):
            success_count += 1

    print(f"\n🎉 Successfully downloaded {success_count}/{num_images} images!")
    print(f"📍 Saved to: {SAVE_PATH}")

# ------------------------------
# Main function
# ------------------------------
def main():
    print("Random Image Downloader")
    print("=" * 30)
    download_multiple_images(10)

if __name__ == "__main__":
    main()
```



```
(env) PS F:\AI Lab experiment lab\experiment 1> python task1.py
Random Image Downloader
==============================
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_1.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_2.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_3.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_4.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_5.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_6.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_7.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_8.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_9.jpg
✓ Downloaded: F:/AI Lab experiment lab/experiment 1/image_10.jpg

🎉 Successfully downloaded 10/10 images!
📍 Saved to: F:/AI Lab experiment lab/experiment 1/
(env) PS F:\AI Lab experiment lab\experiment 1>
```

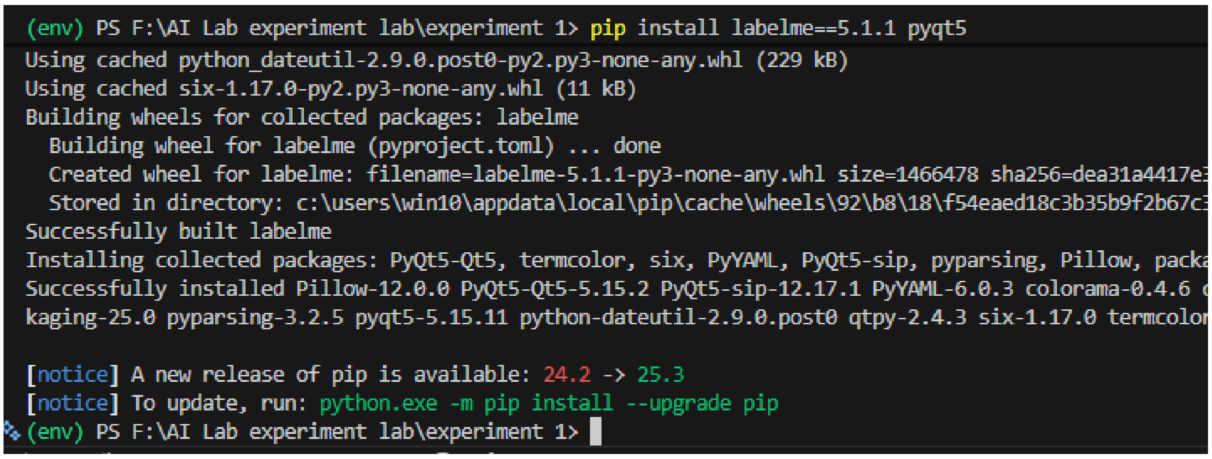
Output:



Task 2: Image Scene Segmentation and Annotation:

Library:

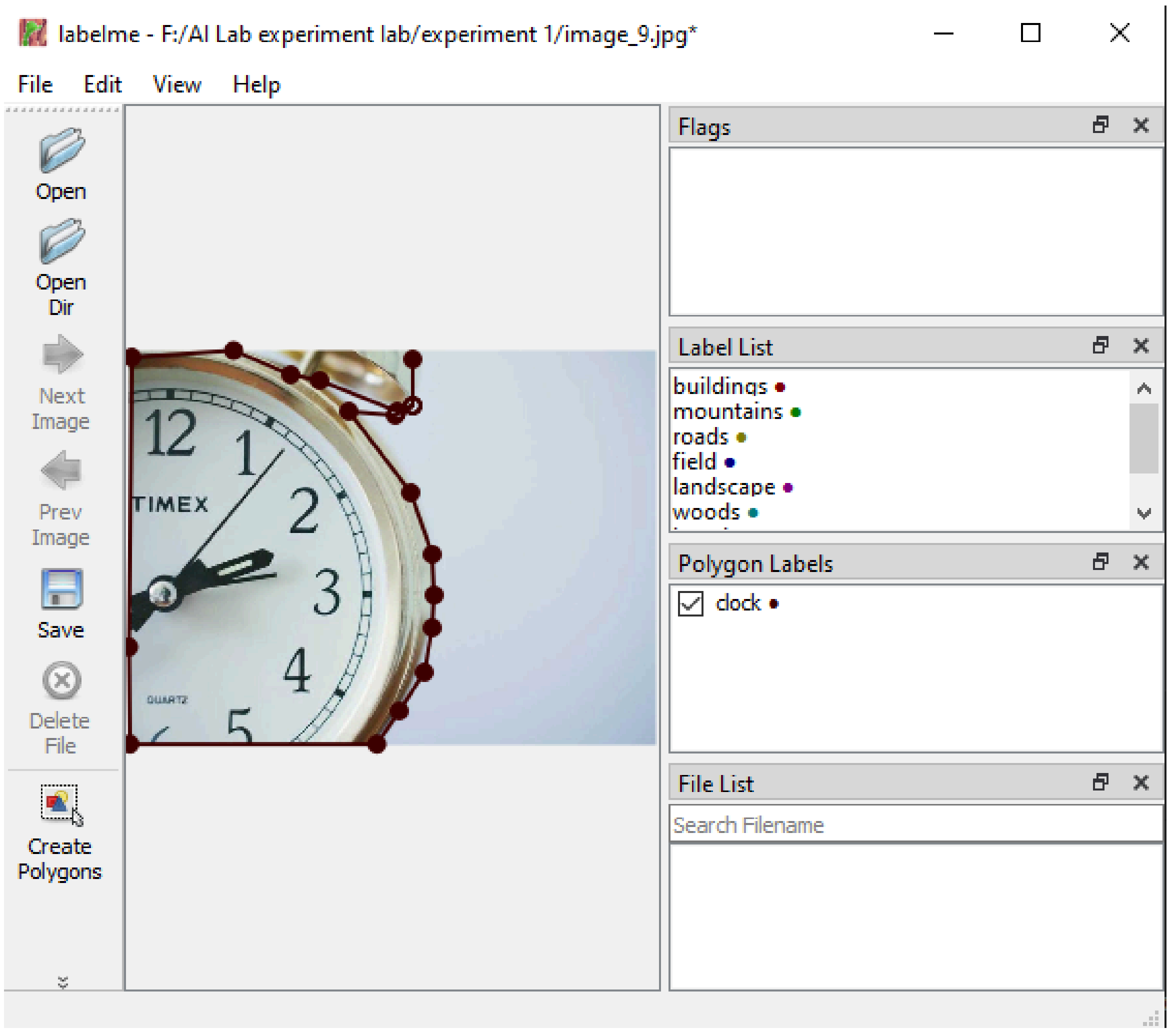
```
pip install labelme==5.1.1 pyqt5
```



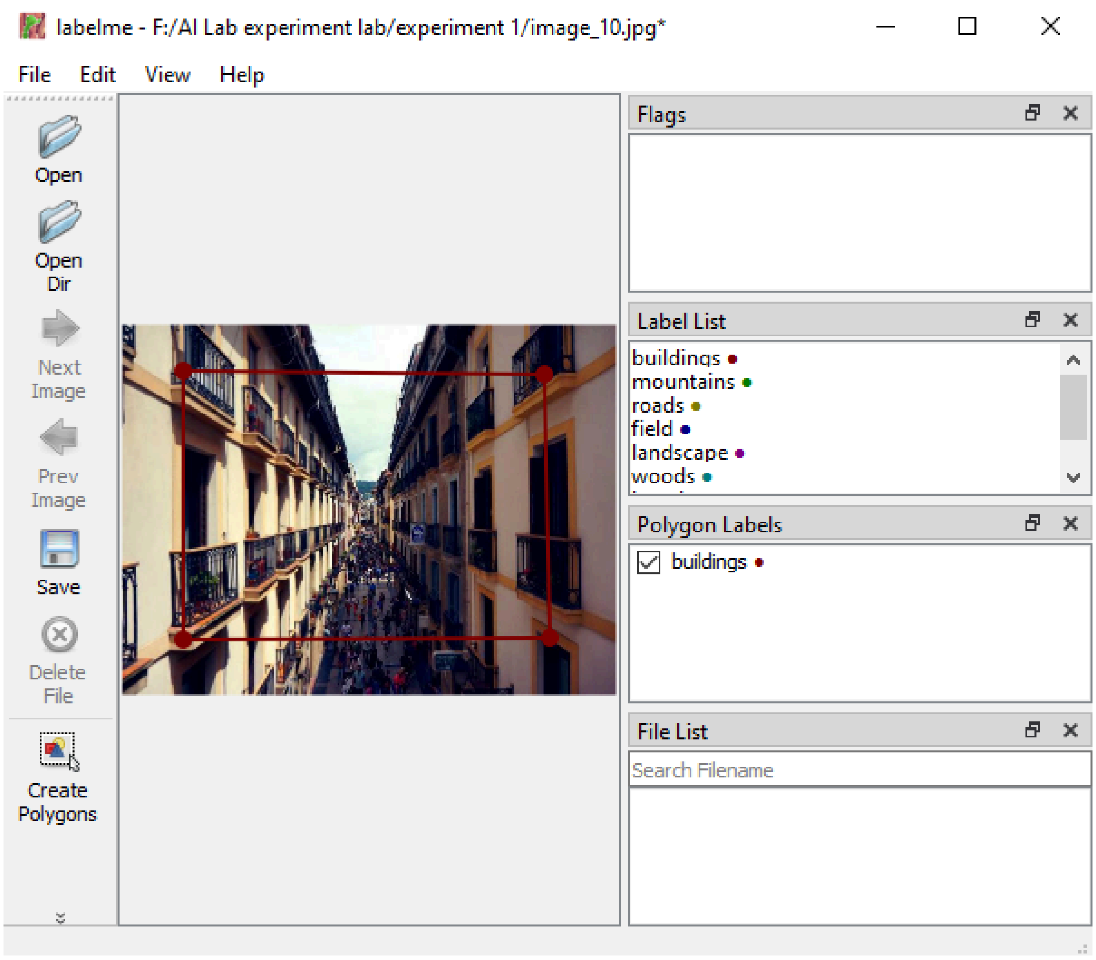
```
(env) PS F:\AI Lab experiment lab\experiment 1> pip install labelme==5.1.1 pyqt5
Using cached python_dateutil-2.9.0.post0-py2.py3-none-any.whl (229 kB)
Using cached six-1.17.0-py2.py3-none-any.whl (11 kB)
Building wheels for collected packages: labelme
  Building wheel for labelme (pyproject.toml) ... done
  Created wheel for labelme: filename=labelme-5.1.1-py3-none-any.whl size=1466478 sha256=dea31a4417e3
  Stored in directory: c:\users\win10\appdata\local\pip\cache\wheels\92\b8\18\f54eaed18c3b35b9f2b67c3
Successfully built labelme
Installing collected packages: PyQt5-Qt5, termcolor, six, PyYAML, PyQt5-sip, pyparsing, Pillow, packa
Successfully installed Pillow-12.0.0 PyQt5-Qt5-5.15.2 PyQt5-sip-12.17.1 PyYAML-6.0.3 colorama-0.4.6 c
kaging-25.0 pyparsing-3.2.5 pyqt5-5.15.11 python-dateutil-2.9.0.post0 qtpy-2.4.3 six-1.17.0 termcolor

[notice] A new release of pip is available: 24.2 -> 25.3
[notice] To update, run: python.exe -m pip install --upgrade pip
(env) PS F:\AI Lab experiment lab\experiment 1> 
```

Powershell command: for Opening: *Labelme GUI*

```
python -m labelme
```



labelme - F:/AI Lab experiment lab/experiment 1/image_9.jpg*


labelme - F:/AI Lab experiment lab/experiment 1/image_10.jpg*

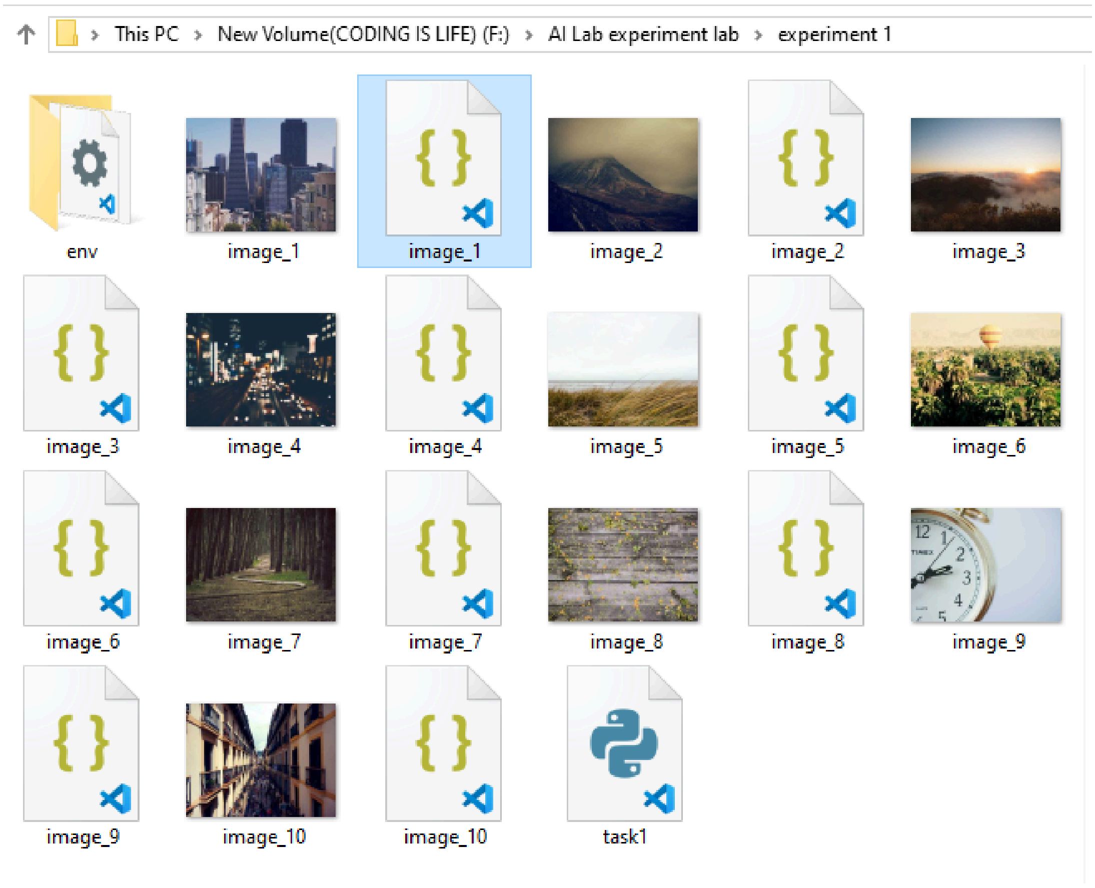
File Edit View Help

Flags

Label List
buildings ●
mountains ●
roads ●
field ●
landscape ●
woods ●

Polygon Labels
☑ buildings ●

File List
Search Filename


This PC › New Volume(CODING IS LIFE) (F:) › AI Lab experiment lab › experiment 1

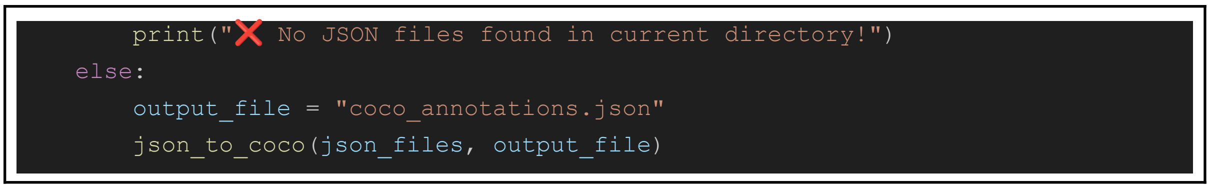
env, image_1, image_1, image_2, image_2, image_3, image_3, image_4, image_4, image_5, image_5, image_6, image_6, image_7, image_7, image_8, image_8, image_9, image_9, image_10, image_10, task1

Convert JSON to VOC(xml format)

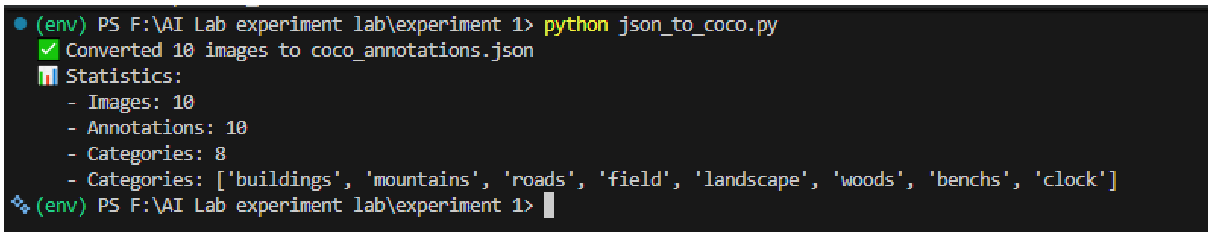
```python
import json
import xml.etree.ElementTree as ET
import os
from pathlib import Path

def json_to_voc_xml(json_file, output_dir):
    with open(json_file, 'r', encoding='utf-8') as f:
        data = json.load(f)

    # Create XML structure
    annotation = ET.Element('annotation')

    # Add folder and filename
    folder = ET.SubElement(annotation, 'folder')
    folder.text = 'VOC2007'

    filename = ET.SubElement(annotation, 'filename')
    filename.text = data['imagePath']

    # Add source
    source = ET.SubElement(annotation, 'source')
    ET.SubElement(source, 'database').text = 'Unknown'

    # Add image size
    size = ET.SubElement(annotation, 'size')
    ET.SubElement(size, 'width').text = str(data['imageWidth'])
    ET.SubElement(size, 'height').text = str(data['imageHeight'])
    ET.SubElement(size, 'depth').text = '3'

    # Add segmented
    ET.SubElement(annotation, 'segmented').text = '0'

    # Add objects (bounding boxes)
    for shape in data['shapes']:
        obj = ET.SubElement(annotation, 'object')
        ET.SubElement(obj, 'name').text = shape['label']
        ET.SubElement(obj, 'pose').text = 'Unspecified'
        ET.SubElement(obj, 'truncated').text = '0'
        ET.SubElement(obj, 'difficult').text = '0'

        # Create bounding box from points
        bndbox = ET.SubElement(obj, 'bndbox')
        points = shape['points']
        x_coords = [p[0] for p in points]
        y_coords = [p[1] for p in points]

        ET.SubElement(bndbox, 'xmin').text = str(int(min(x_coords)))
        ET.SubElement(bndbox, 'ymin').text = str(int(min(y_coords)))
        ET.SubElement(bndbox, 'xmax').text = str(int(max(x_coords)))
        ET.SubElement(bndbox, 'ymax').text = str(int(max(y_coords)))

    # Create XML file
    xml_filename = Path(json_file).stem + '.xml'
    xml_path = Path(output_dir) / xml_filename

    # Write XML file
    tree = ET.ElementTree(annotation)
    tree.write(xml_path, encoding='utf-8', xml_declaration=True)
    print(f"✅ Created: {xml_path}")

# Main conversion
if __name__ == "__main__":
    output_dir = "voc_xml_annotations"
    os.makedirs(output_dir, exist_ok=True)

    # Convert all JSON files in current directory
    json_files = [f for f in os.listdir('.') if f.endswith('.json')]

    if not json_files:
        print("❌ No JSON files found in current directory!")
    else:
        for json_file in json_files:
            json_to_voc_xml(json_file, output_dir)
        print(f"🎉 Converted {len(json_files)} JSON files to VOC XML format!")
```

Convert all JSON files to COCO:

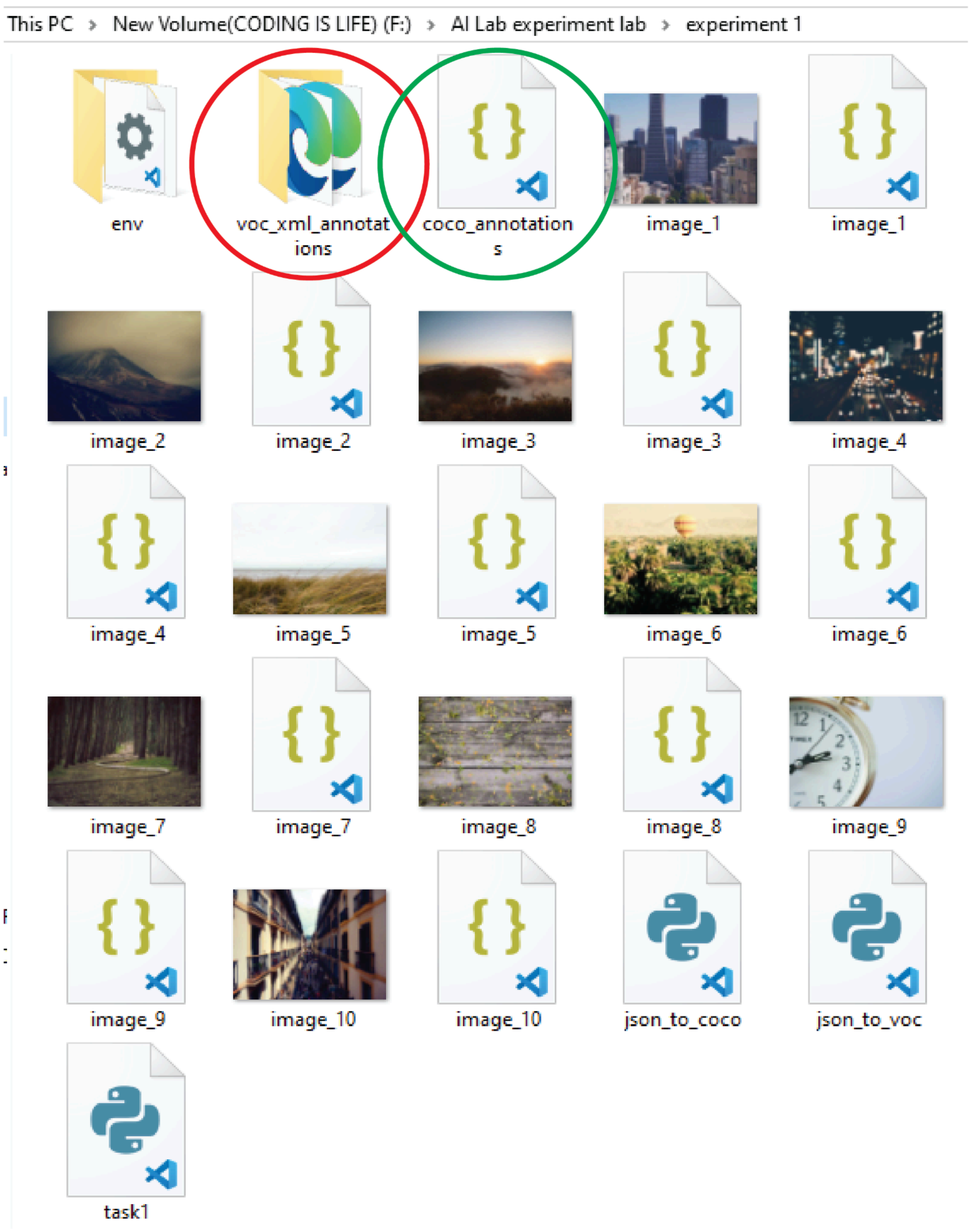
```python
import json
import os
import base64
from datetime import datetime
from labelme import utils
import numpy as np

def json_to_coco(json_files, output_file):
    coco = {
        "info": {
            "description": "COCO Dataset",
            "url": "",
            "version": "1.0",
            "year": datetime.now().year,
            "contributor": "",
            "date_created": datetime.now().strftime("%Y/%m/%d")
        },
        "licenses": [
            {
                "url": "",
                "id": 1,
                "name": "Unknown"
            }
        ],
        "images": [],
        "annotations": [],
        "categories": []
    }

    # Get all categories
    categories = {}
    category_id = 1

    # First pass: collect all categories
    for json_file in json_files:
        with open(json_file, 'r') as f:
            data = json.load(f)
            for shape in data['shapes']:
                if shape['label'] not in categories:
                    categories[shape['label']] = category_id
                    coco['categories'].append({
                        "id": category_id,
                        "name": shape['label'],
                        "supercategory": "none"
                    })
                    category_id += 1

    # Second pass: create images and annotations
    image_id = 1
    annotation_id = 1

    for json_file in json_files:
        with open(json_file, 'r') as f:
            data = json.load(f)

        # Add image
        coco['images'].append({
            "id": image_id,
            "width": data['imageWidth'],
            "height": data['imageHeight'],
            "file_name": data['imagePath'],
            "license": 1,
            "flickr_url": "",
            "coco_url": "",
            "date_captured": ""
        })

        # Add annotations
        for shape in data['shapes']:
            points = shape['points']

            # Convert points to segmentation format
            segmentation = []
            for point in points:
                segmentation.extend([point[0], point[1]])

            # Calculate bounding box [x, y, width, height]
            x_coords = [p[0] for p in points]
            y_coords = [p[1] for p in points]
            bbox = [
                float(min(x_coords)),
                float(min(y_coords)),
                float(max(x_coords) - min(x_coords)),
                float(max(y_coords) - min(y_coords))
            ]

            # Calculate area
            area = bbox[2] * bbox[3]

            coco['annotations'].append({
                "id": annotation_id,
                "image_id": image_id,
                "category_id": categories[shape['label']],
                "segmentation": [segmentation],
                "area": area,
                "bbox": bbox,
                "iscrowd": 0
            })
            annotation_id += 1

        image_id += 1

    # Save COCO JSON
    with open(output_file, 'w') as f:
        json.dump(coco, f, indent=2)

    print(f"✅ Converted {len(json_files)} images to {output_file}")
    print(f"📊 Statistics:")
    print(f"   - Images: {len(coco['images'])}")
    print(f"   - Annotations: {len(coco['annotations'])}")
    print(f"   - Categories: {len(coco['categories'])}")
    print(f"   - Categories: {[cat['name'] for cat in coco['categories']]}")

if __name__ == "__main__":
    # Get all JSON files in current directory
    json_files = [f for f in os.listdir('.') if f.endswith('.json')]

    if not json_files:
```

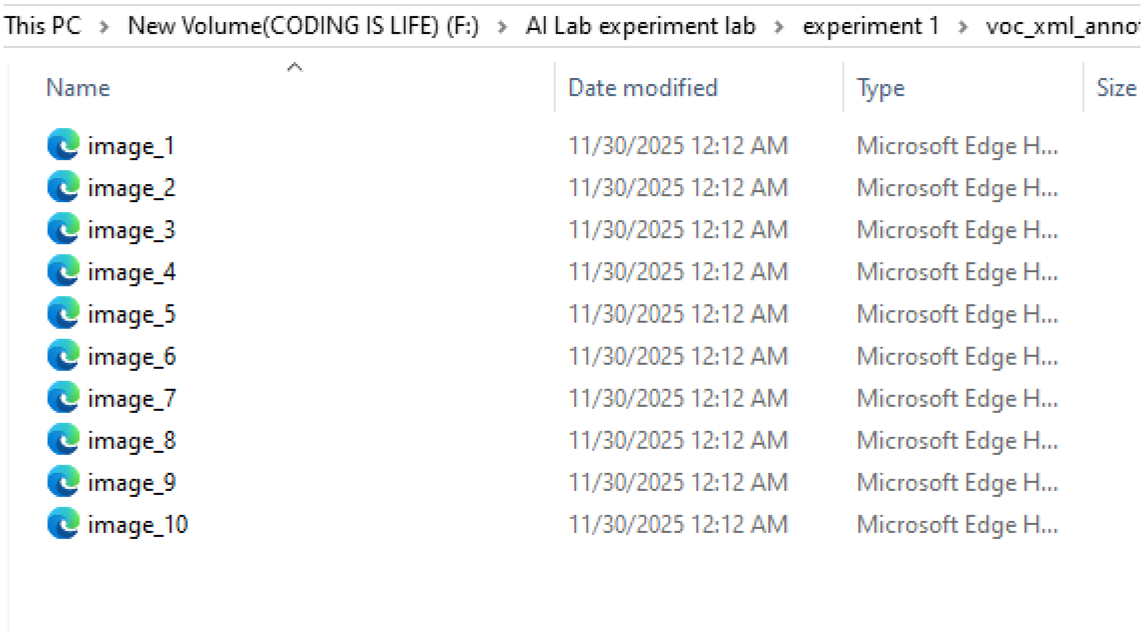


```
        print("❌ No JSON files found in current directory!")
    else:
        output_file = "coco_annotations.json"
        json_to_coco(json_files, output_file)
```

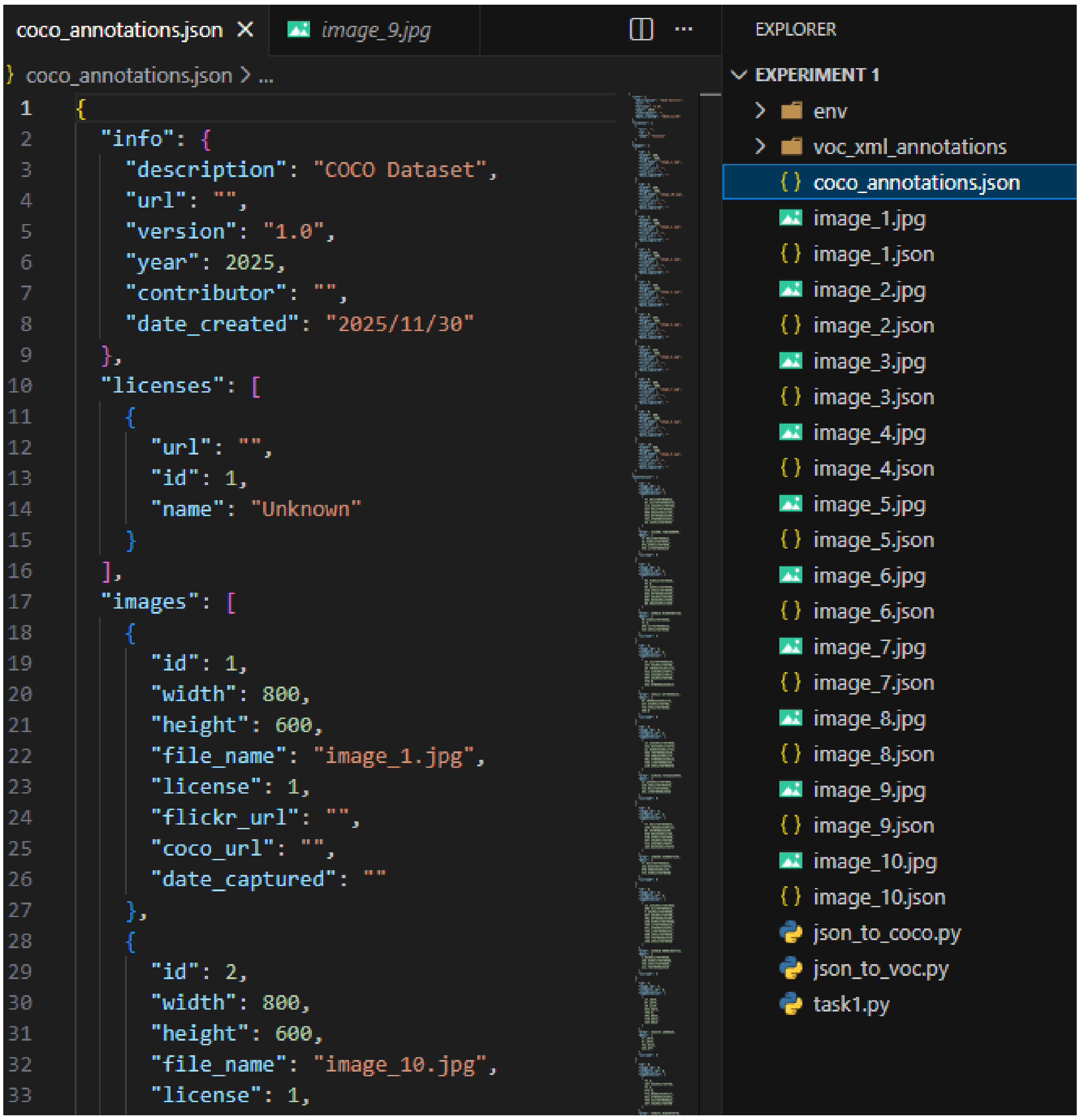


```
● (env) PS F:\AI Lab experiment lab\experiment 1> python json_to_coco.py
  ✅ Converted 10 images to coco_annotations.json
  📊 Statistics:
     - Images: 10
     - Annotations: 10
     - Categories: 8
     - Categories: ['buildings', 'mountains', 'roads', 'field', 'landscape', 'woods', 'benchs', 'clock']
  (env) PS F:\AI Lab experiment lab\experiment 1> 
```




This PC > New Volume(CODING IS LIFE) (F:) > AI Lab experiment lab > experiment 1 > voc_xml_anno

Name	Date modified	Type	Size
image_1	11/30/2025 12:12 AM	Microsoft Edge H...	
image_2	11/30/2025 12:12 AM	Microsoft Edge H...	
image_3	11/30/2025 12:12 AM	Microsoft Edge H...	
image_4	11/30/2025 12:12 AM	Microsoft Edge H...	
image_5	11/30/2025 12:12 AM	Microsoft Edge H...	
image_6	11/30/2025 12:12 AM	Microsoft Edge H...	
image_7	11/30/2025 12:12 AM	Microsoft Edge H...	
image_8	11/30/2025 12:12 AM	Microsoft Edge H...	
image_9	11/30/2025 12:12 AM	Microsoft Edge H...	
image_10	11/30/2025 12:12 AM	Microsoft Edge H...	



```json
{
  "info": {
    "description": "COCO Dataset",
    "url": "",
    "version": "1.0",
    "year": 2025,
    "contributor": "",
    "date_created": "2025/11/30"
  },
  "licenses": [
    {
      "url": "",
      "id": 1,
      "name": "Unknown"
    }
  ],
  "images": [
    {
      "id": 1,
      "width": 800,
      "height": 600,
      "file_name": "image_1.jpg",
      "license": 1,
      "flickr_url": "",
      "coco_url": "",
      "date_captured": ""
    },
    {
      "id": 2,
      "width": 800,
      "height": 600,
      "file_name": "image_10.jpg",
      "license": 1,
```

Task 3: Data Visualization

Bar Chart:

```
import matplotlib.pyplot as plt
import os
from PIL import Image
import pandas as pd

# Set Chinese font support
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

def analyze_downloaded_images():
    # Analyze images downloaded in Task 1
    image_folder = './'  # Where Task 1 saved images
    image_files = [f for f in os.listdir(image_folder) if f.endswith('.jpg')]

    # Get image sizes and stats
    image_data = []
    for img_file in image_files:
        try:
            with Image.open(os.path.join(image_folder, img_file)) as img:
                width, height = img.size
                 file_size = os.path.getsize(os.path.join(image_folder, img_file)) /
1024  # KB
                image_data.append({
                    'filename': img_file,
                    'width': width,
                    'height': height,
                    'file_size_kb': file_size,
                    'aspect_ratio': width / height
                })
        except:
            continue
    return pd.DataFrame(image_data)
df = analyze_downloaded_images()

if not df.empty:
    # Bar chart: Image file sizes
    plt.figure(figsize=(10, 6))
    plt.bar(range(len(df)), df['file_size_kb'], color='skyblue')
    plt.xlabel('Image Index')
    plt.ylabel('File Size (KB)')
    plt.title('Downloaded Image File Sizes from Baidu Tieba')
    plt.xticks(range(len(df)), df['filename'], rotation=45)
    plt.tight_layout()
    plt.show()

    # Scatter plot: Image dimensions
    plt.figure(figsize=(10, 6))
        plt.scatter(df['width'],  df['height'],  s=df['file_size_kb']*2,  alpha=0.6,
color='red')
    plt.xlabel('Image Width (pixels)')
    plt.ylabel('Image Height (pixels)')
    plt.title('Image Dimensions and File Sizes (Bubble Size = File Size)')

    # Add labels for some points
    for i, row in df.iterrows():
        if i % 2 == 0:  # Label every other image to avoid clutter
            plt.annotate(row['filename'], (row['width'], row['height']),
                         xytext=(5, 5), textcoords='offset points', fontsize=8)

    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()
    print(f"Total images analyzed: {len(df)}")
    print(f"Average file size: {df['file_size_kb'].mean():.2f} KB")
                print(f"Average    dimensions:    {df['width'].mean():.0f}    x
{df['height'].mean():.0f} pixels")
else:
    print("No images found! Run Task 1 first to download images.")
```

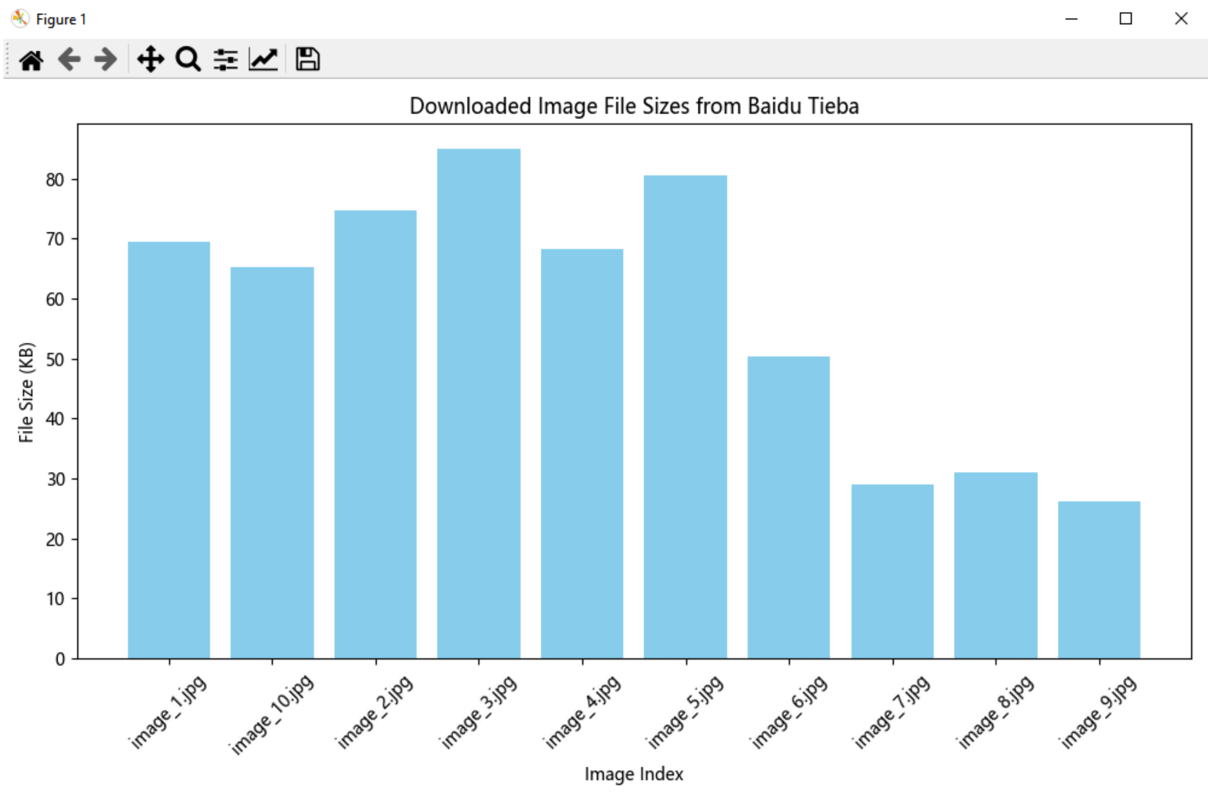


Fig 1 : Barchart

Bubble Plot:

```
import matplotlib.pyplot as plt
import os
```

```python
from PIL import Image
import pandas as pd
import numpy as np

# Set Chinese font support
plt.rcParams['font.sans-serif'] = ['Microsoft YaHei']
plt.rcParams['axes.unicode_minus'] = False

def analyze_downloaded_images():
    """
    Analyze images downloaded in Task 1
    Returns: DataFrame with image statistics
    """
    image_folder = './'  # Where Task 1 saved images
    image_files = [f for f in os.listdir(image_folder) if f.endswith('.jpg')]

    print(f"Found {len(image_files)} JPG files")
    image_data = []
    for img_file in image_files:
        try:
            with Image.open(os.path.join(image_folder, img_file)) as img:
                width, height = img.size
                file_size = os.path.getsize(os.path.join(image_folder, img_file)) / 1024  #
KB

                total_pixels = width * height
                image_data.append({
                    'filename': img_file,
                    'width': width,
                    'height': height,
                    'file_size_kb': file_size,
                    'aspect_ratio': width / height,
                    'total_pixels': total_pixels,
                    'pixels_per_kb': total_pixels / file_size if file_size > 0 else 0
                })
                print(f"Analyzed: {img_file} - {width}x{height} - {file_size:.1f}KB")
        except Exception as e:
            print(f"Error analyzing {img_file}: {e}")
            continue
    return pd.DataFrame(image_data)
print("=== Analyzing Downloaded Images ===")
df = analyze_downloaded_images()

if not df.empty:
    print(f"\n=== Successfully analyzed {len(df)} images ===")
    plt.figure(figsize=(16, 10))  # Wider figure to accommodate side legend
  scatter = plt.scatter(
        df['total_pixels'] / 1000,           # X: Total pixels (in thousands)
        df['file_size_kb'],                  # Y: File size (KB)
        s=df['aspect_ratio'] * 100,          # Bubble size = aspect ratio
        c=df['height'],                      # Color = image height
        cmap='viridis',                      # Color map
        alpha=0.7,                           # Transparency
        edgecolors='black',                  # Bubble borders
        linewidth=0.8,
        marker='o'
    )
    plt.xlabel('Image Resolution (thousands of pixels)',
               fontdict={'family': 'Microsoft YaHei', 'color': 'k', 'size': 14},
               labelpad=15)
    plt.ylabel('File Size (KB)',
               fontdict={'family': 'Microsoft YaHei', 'color': 'k', 'size': 14},
               labelpad=15)
    plt.title('Image Resolution vs File Size Analysis\n(Bubble Size = Aspect Ratio, Color =
Image Height)',
              fontdict={'family': 'Microsoft YaHei', 'color': 'k', 'size': 16},
              pad=20)
    cbar = plt.colorbar(scatter, pad=0.02)
    cbar.set_label('Image Height (pixels)',
                   fontdict={'family': 'Microsoft YaHei', 'size': 12})

    plt.grid(True, alpha=0.3, linestyle='--')
    df['efficiency'] = df['pixels_per_kb']
    for i, row in df.iterrows():
        x_pos = row['total_pixels'] / 1000
        y_pos = row['file_size_kb']
        plt.annotate(
            f"{row['efficiency']:.0f} px/KB",
            (x_pos, y_pos),
            xytext=(10, 10),  # Increased offset
            textcoords='offset points',
            fontsize=9,
            fontweight='bold',
                        bbox=dict(boxstyle='round,pad=0.3',  facecolor='white',  alpha=0.9,
edgecolor='gray')
        )
       aspect_sizes = [0.5, 1.0, 2.0]  # Portrait, Square, Landscape
    legend_labels = ['Portrait (0.5:1)', 'Square (1:1)', 'Landscape (2:1)']
    legend_elements = []
    for size, label in zip(aspect_sizes, legend_labels):
        legend_elements.append(
            plt.scatter([], [], s=size * 100, c='gray', alpha=0.7,
                       edgecolors='black', label=label)
        )
    plt.legend(handles=legend_elements, title='Aspect Ratios',
               loc='upper left', framealpha=0.9, fontsize=10,
               bbox_to_anchor=(0.35, 0.98))
    x_padding = (df['total_pixels'].max() / 1000) * 0.1
    y_padding = df['file_size_kb'].max() * 0.1
    plt.xlim(0, (df['total_pixels'].max() / 1000) + x_padding)
    plt.ylim(0, df['file_size_kb'].max() + y_padding)
    avg_efficiency = df['pixels_per_kb'].mean()
    max_resolution = df['total_pixels'].max()
    min_file_size = df['file_size_kb'].min()
    stats_text = f"""Statistical Summary:
• Average efficiency: {avg_efficiency:.0f} pixels/KB
• Max resolution: {max_resolution:,} pixels
• Min file size: {min_file_size:.1f} KB
• Images analyzed: {len(df)}"""
    plt.annotate(stats_text,
                xy=(0.02, 0.98), xycoords='axes fraction',  # Upper left
                fontsize=10, va='top', ha='left',
                bbox=dict(boxstyle='round', facecolor='wheat', alpha=0.8))
    plt.subplots_adjust(right=0.85)  # Make room for colorbar and legend
    plt.show()
    print(f"Resolution range: {df['total_pixels'].min():,} - {df['total_pixels'].max():,}
pixels")
                   print(f"File    size    range:    {df['file_size_kb'].min():.1f}    -
{df['file_size_kb'].max():.1f} KB")
                  print(f"Aspect    ratio    range:    {df['aspect_ratio'].min():.2f}    -
{df['aspect_ratio'].max():.2f}")
    print(f"Average efficiency: {df['pixels_per_kb'].mean():.0f} pixels per KB")
    print(f"Most efficient image: {df.loc[df['pixels_per_kb'].idxmax(), 'filename']} "
          f"({df['pixels_per_kb'].max():.0f} px/KB)")
    print(f"Least efficient image: {df.loc[df['pixels_per_kb'].idxmin(), 'filename']} "
          f"({df['pixels_per_kb'].min():.0f} px/KB)")
```

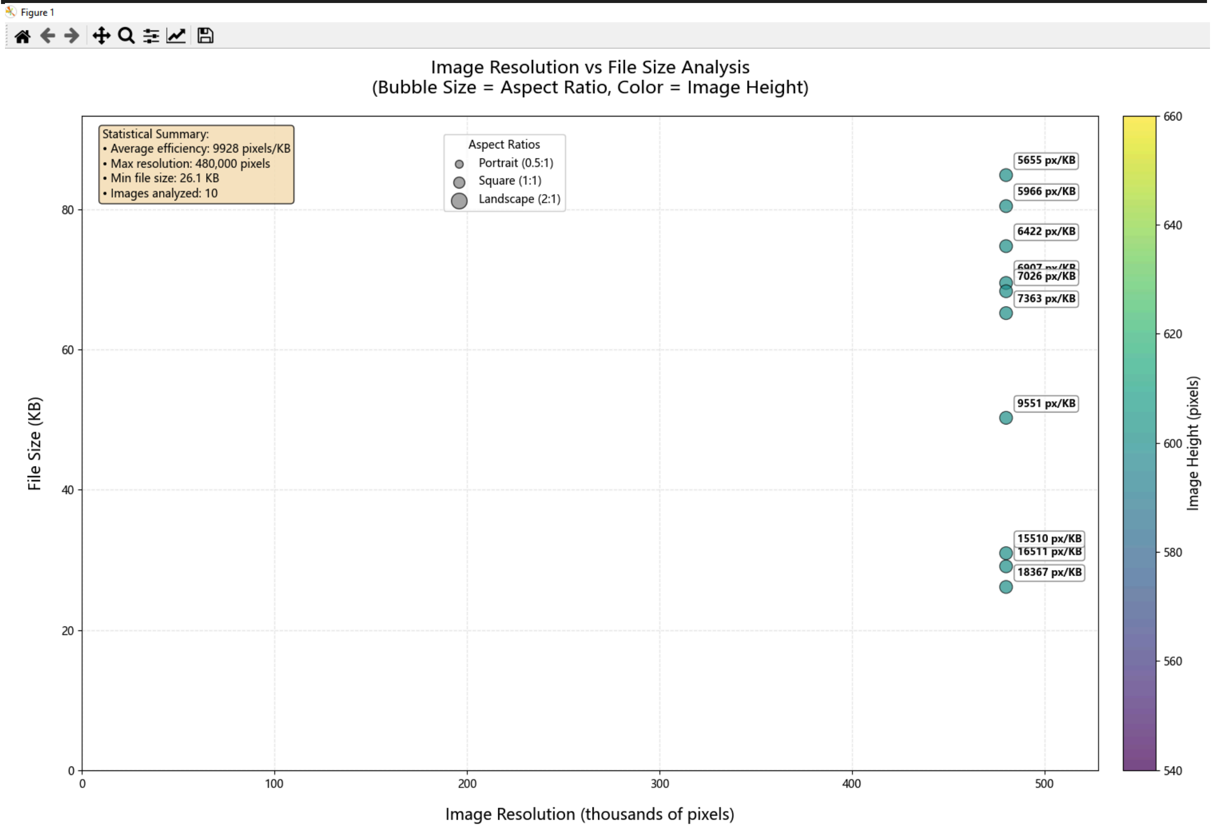


Fig 2 : Bubble Plot

Project Summary

The experiment successfully demonstrated a complete **data workflow** using Python, focusing on **acquisition, annotation, and visualization**. Key phases included using a Python crawler for **Web Scraping** images from Baidu Tieba, **Image Annotation** with Labelme for labeling objects (dogs, cats, wood) to generate machine learning-ready datasets, and **Data Visualization** using Matplotlib to analyze trends. This confirmed the experiment's success in building essential data handling competencies and establishing a foundation for object detection and image segmentation applications.