Planet Labs Imagery Processing Pipeline - Complete User Guide

A comprehensive, modular Python pipeline for querying, downloading, processing, and preparing Planet Labs satellite imagery for machine learning workflows.

■ Table of Contents

■ Installation & Setup

Step 1: Download and Organize Files

1. Download all files from Claude
2. Put them in a folder (e.g., `planet-pipeline`)
3. Open folder in VSCode: **File** → **Open Folder**
4. Open terminal: **Terminal** → **New Terminal** (or Ctrl+`)

Step 2: Run Setup Script

```
python SETUP_INSTRUCTIONS.py
```

This creates the proper directory structure.

Step 3: Create Virtual Environment

```
# Create virtual environment
python -m venv venv

# Activate it
# Mac/Linux:
source venv/bin/activate
```

```
# Windows (PowerShell):
venv\Scripts\Activate.ps1

# Windows (Command Prompt):
venv\Scripts\activate.bat
```

✓ **You should see** `(venv)` **at the start of your terminal prompt**

Step 4: Install Dependencies

```
pip install -r requirements.txt
```

This installs all necessary packages (takes 2-3 minutes).

■■ Configuration (Where YOU Input Information)

■ 1. SET UP YOUR API KEY (Required)

Option A: Environment Variable (Recommended)

Create `.env` **file:**

```
# Copy the template
cp .env.example .env

# Open in VSCode
code .env
```

Edit `.env` **file and replace with YOUR key:**

```
# ■■ REPLACE THIS WITH YOUR ACTUAL PLANET API KEY ■■
PL_API_KEY=your_actual_planet_api_key_here
```

Where to get your API key:

1. Go to: https://www.planet.com/account/#/

2. Sign in

3. Copy your API key from the Account Settings page

Option B: Pass Directly in Code

```
from planet_pipeline import PlanetPipeline

# ■■ REPLACE WITH YOUR KEY ■■
pipeline = PlanetPipeline(api_key="your_actual_planet_api_key_here")
```

■ 2. DEFINE YOUR AREAS OF INTEREST (AOIs)

You have multiple options for defining AOIs:

Option A: Create AOI from Coordinates (Inline)

```
from planet_pipeline import PlanetPipeline

pipeline = PlanetPipeline(storage_dir="./my_data")

# ■■ REPLACE WITH YOUR AOI COORDINATES ■■
pipeline.add_aoi(
    name="my_study_area",  # ■■ YOUR AOI NAME
```

```
    geometry={
        "type": "Polygon",
        "coordinates": [[
            [-122.5, 37.7],   # ■■ YOUR COORDINATES [longitude, latitude]
            [-122.5, 37.8],   # Northwest corner
            [-122.4, 37.8],   # Northeast corner
            [-122.4, 37.7],   # Southeast corner
            [-122.5, 37.7]    # Close the polygon (same as first point)
        ]]
    },
    metadata={"description": "My description"}  # ■■ OPTIONAL: Your metadata
)
```

■ Coordinate Format:

• `[longitude, latitude]` (NOT latitude, longitude!)

• Longitude: -180 to 180 (negative = West, positive = East)

• Latitude: -90 to 90 (negative = South, positive = North)

Option B: Create AOI from GeoJSON File

Create a GeoJSON file (e.g., `my_aoi.geojson`):

```
{
  "type": "Feature",
  "properties": {
    "name": "San Francisco Bay"
  },
  "geometry": {
    "type": "Polygon",
    "coordinates": [[
      [-122.5, 37.7],
      [-122.5, 37.8],
      [-122.4, 37.8],
      [-122.4, 37.7],
      [-122.5, 37.7]
    ]]
  }
}
```

Load it in Python:

```
# ■■ REPLACE WITH YOUR FILENAME ■■
pipeline.add_aoi(
    name="sf_bay",              # ■■ YOUR AOI NAME
    geometry_file="my_aoi.geojson"  # ■■ YOUR GEOJSON FILE PATH
)
```

■■ Tools to Create GeoJSON:

• **geojson.io** - Draw polygons on a map, download GeoJSON

• **QGIS** - Professional GIS software (free)

• **Google Earth** - Draw, export as KML, convert to GeoJSON

Option C: Multiple AOIs from One File

Create a JSON file (e.g., `all_aois.json`):

```
{
  "field_1": {
    "geometry": {
      "type": "Polygon",
      "coordinates": [[
        [-121.5, 37.5],
        [-121.5, 37.51],
        [-121.49, 37.51],
```

```
            [-121.49, 37.5],
            [-121.5, 37.5]
          ]]
        },
        "metadata": {"crop": "corn", "field_id": "F001"}
      },
      "field_2": {
        "geometry": {
          "type": "Polygon",
          "coordinates": [[
            [-121.6, 37.6],
            [-121.6, 37.61],
            [-121.59, 37.61],
            [-121.59, 37.6],
            [-121.6, 37.6]
          ]]
        },
        "metadata": {"crop": "wheat", "field_id": "F002"}
      }
    }
```

Load in Python:

```
    # ■■ REPLACE WITH YOUR FILENAME ■■
    pipeline.add_aois_from_file("all_aois.json")
```

■ 3. SET YOUR DATE RANGES

```
    # ■■ REPLACE WITH YOUR DATES (format: YYYY-MM-DD) ■■
    pipeline.query_all_aois(
        start_date="2024-01-01",  # ■■ YOUR START DATE
        end_date="2024-01-31"      # ■■ YOUR END DATE
    )
```

Date Format Examples:

• `"2024-01-01"` - January 1, 2024

• `"2024-06-15"` - June 15, 2024

• `"2023-12-31"` - December 31, 2023

■■ 4. CONFIGURE STORAGE DIRECTORY

```
    # ■■ REPLACE WITH YOUR DESIRED STORAGE PATH ■■
    pipeline = PlanetPipeline(
        storage_dir="./planet_data"  # ■■ YOUR STORAGE DIRECTORY
    )
```

Default structure created:

```
    planet_data/           # ■■ Your storage_dir
    ■■■ aois/               # AOI definitions
    ■■■ imagery/            # Downloaded imagery
    ■■■ processed/          # Preprocessed imagery
    ■■■ indices/            # Spectral indices
    ■■■ ml_datasets/        # ML-ready datasets
    ■■■ cache/              # API response cache
```

■■ 5. CUSTOMIZE QUERY PARAMETERS

```
    pipeline.query_all_aois(
```

```
    start_date="2024-01-01",          # ■■ YOUR DATE
    end_date="2024-01-31",            # ■■ YOUR DATE
    item_types=["PSScene"],           # ■■ YOUR ITEM TYPES (see options below)
    cloud_cover_max=0.1,              # ■■ YOUR MAX CLOUD COVER (0.0 to 1.0)
    min_gsd=None,                     # ■■ YOUR MIN RESOLUTION (meters) or None
    max_gsd=3.5                       # ■■ YOUR MAX RESOLUTION (meters) or None
)
```

Item Type Options:

• `"PSScene"` - PlanetScope 3-4m resolution (most common)

• `"SkySatCollect"` - SkySat ~0.5m resolution (very high res)

• `"SkySatScene"` - SkySat ~0.8m orthorectified

• `"REOrthoTile"` - RapidEye 5m resolution

• `"Sentinel2L1C"` - Sentinel-2 10m resolution

Resolution (GSD) Guide:

• `max_gsd=3.0` - Only high-resolution imagery (3m or better)

• `max_gsd=5.0` - Medium to high resolution

• `min_gsd=1.0, max_gsd=3.0` - Between 1m and 3m only

■ 6. CONFIGURE DOWNLOAD SETTINGS

```
pipeline.download_imagery(
    asset_types=["ortho_analytic_4b"],  # ■■ YOUR ASSET TYPES (see options below
  )
    aoi_filter=None,                     # ■■ YOUR AOI NAMES or None for all
    max_cloud_cover=0.1,                 # ■■ YOUR CLOUD COVER FILTER
    limit_per_aoi=10                     # ■■ YOUR LIMIT (or None for all)
)
```

Asset Type Options:

• `"ortho_analytic_4b"` - 4-band multispectral (Blue, Green, Red, NIR) - Most common

• `"ortho_analytic_8b"` - 8-band multispectral (more bands)

• `"ortho_visual"` - RGB visual imagery

• `"ortho_analytic_sr"` - Surface reflectance corrected

• `"basic_analytic"` - Uncorrected analytic

• `"basic_analytic_dn"` - Digital numbers

■ 7. SELECT SPECTRAL INDICES

```
# ■■ CHOOSE YOUR INDICES ■■
pipeline.calculate_indices(
    indices=["ndvi", "ndwi", "evi"],  # ■■ YOUR INDEX NAMES (see full list below
  )
    aoi_filter=None                    # ■■ YOUR AOI NAMES or None for all
)
```

Available Indices:

• **Vegetation:** `"ndvi"`, `"evi"`, `"savi"`, `"msavi"`, `"gndvi"`, `"gci"`

• **Water:** `"ndwi"`, `"ndmi"`

• **Urban:** `"ndbi"`

• **Fire/Burn:** `"bai"`, `"nbr"`

• **Other:** `"arvi"`, `"sipi"`, `"vari"`

■ 8. ML DATASET PREPARATION

```
dataset_path = pipeline.prepare_for_ml(
    model_type="pytorch",        # ■■ YOUR FRAMEWORK: "pytorch", "tensorflow", o
  r "sklearn"
    output_format="chips",       # ■■ YOUR FORMAT: "chips", "patches", or "full"
    chip_size=256,               # ■■ YOUR CHIP SIZE (pixels)
    overlap=32,                  # ■■ YOUR OVERLAP (pixels)
    train_split=0.7,             # ■■ YOUR TRAIN % (0.0 to 1.0)
    val_split=0.15,              # ■■ YOUR VALIDATION %
    test_split=0.15,             # ■■ YOUR TEST %
    normalize=True,              # ■■ YOUR CHOICE: True or False
    augment=True,                # ■■ YOUR CHOICE: True or False (adds flips/rot
  ations)
    label_file=None              # ■■ YOUR LABELS FILE or None
)
```

Output Format Options:

• `"chips"` - Regular grid of tiles with overlap

• `"patches"` - Random samples from images

• `"full"` - Entire images (no tiling)

Model Type Options:

• `"pytorch"` - Creates PyTorch Dataset class

• `"tensorflow"` - Creates tf.data pipeline

• `"sklearn"` - Flattened arrays for sklearn

■ Quick Start Examples

Example 1: Basic Single AOI Workflow

```
from planet_pipeline import PlanetPipeline

# Initialize
pipeline = PlanetPipeline(
    storage_dir="./my_project_data"  # ■■ YOUR DIRECTORY
)

# Add AOI
pipeline.add_aoi(
    name="test_site",  # ■■ YOUR AOI NAME
    geometry={
        "type": "Polygon",
```

```
        "coordinates": [[
            [-122.5, 37.7],   # ■■ YOUR COORDINATES
            [-122.5, 37.8],
            [-122.4, 37.8],
            [-122.4, 37.7],
            [-122.5, 37.7]
        ]]
    }
)

# Query Planet API
results = pipeline.query_all_aois(
    start_date="2024-01-01",  # ■■ YOUR DATES
    end_date="2024-01-31",
    cloud_cover_max=0.1       # ■■ YOUR CLOUD THRESHOLD
)

print(f"Found {len(results['test_site'])} scenes")

# Download imagery (limit to 5 for testing)
pipeline.download_imagery(
    asset_types=["ortho_analytic_4b"],
    limit_per_aoi=5  # ■■ YOUR LIMIT
)

# Calculate indices
pipeline.calculate_indices(
    indices=["ndvi", "ndwi"]  # ■■ YOUR INDICES
)

print("■ Complete! Check ./my_project_data/ for outputs")
```

Example 2: Multiple AOIs from File

Create `my_farms.json`**:**

```
{
  "farm_north": {
    "geometry": {
      "type": "Polygon",
      "coordinates": [[
        [-121.5, 37.5],
        [-121.5, 37.51],
        [-121.49, 37.51],
        [-121.49, 37.5],
        [-121.5, 37.5]
      ]]
    },
    "metadata": {"crop": "corn"}
  },
  "farm_south": {
    "geometry": {
      "type": "Polygon",
      "coordinates": [[
        [-121.6, 37.4],
        [-121.6, 37.41],
        [-121.59, 37.41],
        [-121.59, 37.4],
        [-121.6, 37.4]
      ]]
    },
    "metadata": {"crop": "wheat"}
  }
}
```

Run pipeline:

```
from planet_pipeline import PlanetPipeline

pipeline = PlanetPipeline(storage_dir="./farm_data")

# Load multiple AOIs
pipeline.add_aois_from_file("my_farms.json")  # ■■ YOUR FILENAME
```

```
# Query all at once
results = pipeline.query_all_aois(
    start_date="2024-06-01",  # ■■ YOUR DATES
    end_date="2024-08-31"
)

# Download for each
pipeline.download_imagery(limit_per_aoi=20)  # ■■ YOUR LIMIT

# Calculate vegetation indices
pipeline.calculate_indices(
    indices=["ndvi", "evi", "savi"]  # ■■ YOUR INDICES
)

# Get summary
summary = pipeline.get_summary()
print(summary)
```

Example 3: ML Dataset Preparation

```
from planet_pipeline import PlanetPipeline

pipeline = PlanetPipeline(storage_dir="./ml_project")

# Add labeled AOIs for different classes
pipeline.add_aoi(
    name="forest",  # ■■ YOUR CLASS NAME
    geometry_file="forest_samples.geojson"  # ■■ YOUR FILE
)
pipeline.add_aoi(
    name="urban",  # ■■ YOUR CLASS NAME
    geometry_file="urban_samples.geojson"  # ■■ YOUR FILE
)
pipeline.add_aoi(
    name="water",  # ■■ YOUR CLASS NAME
    geometry_file="water_samples.geojson"  # ■■ YOUR FILE
)

# Query and download
pipeline.query_all_aois(
    start_date="2024-01-01",  # ■■ YOUR DATES
    end_date="2024-12-31"
)
pipeline.download_imagery(limit_per_aoi=50)  # ■■ YOUR LIMIT

# Prepare ML dataset
dataset_path = pipeline.prepare_for_ml(
    model_type="pytorch",     # ■■ YOUR FRAMEWORK
    chip_size=256,            # ■■ YOUR SIZE
    overlap=32,               # ■■ YOUR OVERLAP
    train_split=0.7,          # ■■ YOUR SPLITS
    val_split=0.15,
    test_split=0.15,
    augment=True
)

print(f"Dataset ready at: {dataset_path}")
```

■ Detailed Usage Guide

Running the Examples Script

```
# Interactive menu
python examples.py

# Run specific example
python examples.py 1  # ■■ YOUR EXAMPLE NUMBER (1-5)

# Run all examples
python examples.py all
```

Using Individual Modules

```
    # Import specific components
    from planet_pipeline.query import PlanetAPIClient
    from planet_pipeline.storage import ImageryStorage
    from planet_pipeline.indices import SpectralIndices

    # Use independently
    client = PlanetAPIClient(api_key="your_key")  # ■■ YOUR KEY
    storage = ImageryStorage(base_dir="./data")   # ■■ YOUR DIR
    indices = SpectralIndices()

    # Build custom workflow
    search_request = client.build_search_request(
        geometry=your_geometry,      # ■■ YOUR GEOMETRY
        start_date="2024-01-01",     # ■■ YOUR DATES
        end_date="2024-01-31",
        item_types=["PSScene"],
        cloud_cover_max=0.1
    )

    features, _ = client.quick_search(search_request)
```

■ Spectral Indices Reference

Vegetation Indices

|-------|------|----------|-------|----------------|

`ndvi`	Normalized Difference Vegetation Index	General vegetation health	-1 to 1	0.2-0.8 = healthy vegetation
`evi`	Enhanced Vegetation Index	Dense vegetation, high biomass	-1 to 1	Better than NDVI in dense canopy
`savi`	Soil-Adjusted Vegetation Index	Sparse vegetation, exposed soil	-1 to 1.5	Reduces soil brightness effects
`msavi`	Modified SAVI	Variable soil brightness	-1 to 1.5	Self-adjusting soil correction
`gndvi`	Green NDVI	Chlorophyll content	-1 to 1	More sensitive to chlorophyll
`gci`	Green Chlorophyll Index	Chlorophyll concentration	-1 to ~5	Higher = more chlorophyll

Water & Moisture Indices

|-------|------|----------|----------------|

`ndwi`	Normalized Difference Water Index	Water body detection	> 0.3 = water bodies
`ndmi`	Normalized Difference Moisture Index	Vegetation moisture content	Higher = more moisture

Urban & Built-up Indices

|-------|------|----------|----------------|

Fire & Burn Indices

|-------|------|----------|----------------|

bai	Burned Area Index	Burned area detection	Higher = more burned
nbr	Normalized Burn Ratio	Burn severity	Pre/post fire differencing

Other Indices

|-------|------|----------|

arvi	Atmospherically Resistant VI	Areas with atmospheric effects
sipi	Structure Insensitive Pigment Index	Canopy stress detection
vari	Visible Atmospherically Resistant Index	RGB-only vegetation fraction

Usage Example:

```
# Calculate single index
pipeline.calculate_indices(indices=["ndvi"])

# Calculate multiple indices
pipeline.calculate_indices(
    indices=["ndvi", "ndwi", "evi", "savi"]  # ■■ YOUR CHOICES
)

# Calculate all vegetation indices
pipeline.calculate_indices(
    indices=["ndvi", "evi", "savi", "msavi", "gndvi", "gci"]
)
```

■ ML Integration

PyTorch Example

```
# After preparing dataset
import sys
from pathlib import Path

# ■■ REPLACE WITH YOUR DATASET PATH ■■
dataset_path = Path("./planet_data/ml_datasets/pytorch_chips_256")
sys.path.append(str(dataset_path))

from pytorch_loader import PlanetDataset
from torch.utils.data import DataLoader
import torch

# Load datasets
train_dataset = PlanetDataset(dataset_path / "train")
val_dataset = PlanetDataset(dataset_path / "val")
test_dataset = PlanetDataset(dataset_path / "test")
```

```
# Create data loaders
train_loader = DataLoader(
    train_dataset,
    batch_size=32,      # ■■ YOUR BATCH SIZE
    shuffle=True,
    num_workers=4       # ■■ YOUR NUM WORKERS
)

val_loader = DataLoader(val_dataset, batch_size=32, shuffle=False)

# Training loop example
for epoch in range(10):  # ■■ YOUR NUM EPOCHS
    for chips, labels in train_loader:
        # chips shape: [batch_size, 4, 256, 256] for 4-band imagery
        # Your training code here
        pass
```

TensorFlow Example

```
import sys
from pathlib import Path

# ■■ REPLACE WITH YOUR DATASET PATH ■■
dataset_path = Path("./planet_data/ml_datasets/tensorflow_chips_256")
sys.path.append(str(dataset_path))

from tensorflow_loader import create_dataset
import tensorflow as tf

# Create datasets
train_dataset = create_dataset(
    dataset_path / "train",
    batch_size=32  # ■■ YOUR BATCH SIZE
)
val_dataset = create_dataset(dataset_path / "val", batch_size=32)

# Build model
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(64, 3, activation='relu', padding='same',
                           input_shape=(256, 256, 4)),  # ■■ YOUR SHAPE
    # Add your layers here
])

model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)

# Train
model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=10  # ■■ YOUR NUM EPOCHS
)
```

■ Configuration File (Optional)

Create `config.yaml` from template:

```
cp config.yaml.template config.yaml
code config.yaml  # Open in VSCode
```

Edit the configuration:

```
# ■■ CUSTOMIZE THESE VALUES ■■
planet:
  api_key: ${PL_API_KEY}  # Or set directly
```

```
storage:
  base_dir: ./planet_data           # ■■ YOUR STORAGE PATH
  cache_enabled: true
  cache_ttl_days: 30

download:
  max_workers: 4                    # ■■ YOUR PARALLEL DOWNLOADS
  default_asset_types:
    - ortho_analytic_4b             # ■■ YOUR DEFAULT ASSETS

query:
  default_item_types:
    - PSScene                       # ■■ YOUR DEFAULT ITEMS
  default_cloud_cover_max: 0.1      # ■■ YOUR DEFAULT CLOUD COVER

indices:
  default_indices:
    - ndvi                          # ■■ YOUR DEFAULT INDICES
    - ndwi
    - evi

ml:
  default_model_type: pytorch       # ■■ YOUR DEFAULT FRAMEWORK
  chip_size: 256                    # ■■ YOUR DEFAULT CHIP SIZE
  overlap: 32                       # ■■ YOUR DEFAULT OVERLAP
```

■ Troubleshooting

Common Issues

"Planet API key is required"

```
# Solution 1: Set environment variable
# In terminal: export PL_API_KEY='your_key'

# Solution 2: Create .env file
# File: .env
# Content: PL_API_KEY=your_key

# Solution 3: Pass directly
pipeline = PlanetPipeline(api_key="your_key")  # ■■ YOUR KEY
```

"No AOIs registered"

```
# Make sure you added AOIs before querying
pipeline.add_aoi("my_site", geometry_file="aoi.geojson")  # ■■ YOUR AOI
# THEN query
pipeline.query_all_aois(start_date="2024-01-01", end_date="2024-01-31")
```

"Cannot import planet_pipeline"

```
# Make sure you:
# 1. Ran SETUP_INSTRUCTIONS.py
# 2. Are in the correct directory
# 3. Activated virtual environment

# Check structure:
ls planet_pipeline/  # Should show .py files

# Activate venv:
source venv/bin/activate  # Mac/Linux
venv\Scripts\activate     # Windows
```

"File not found" errors

```
# Use absolute paths or check your working directory
import os
print(os.getcwd())  # Shows current directory
```

```
    # Use full paths
    pipeline.add_aoi(
        "site",
        geometry_file="/full/path/to/your/file.geojson"  # ■■ YOUR FULL PATH
    )
```

Download fails / "Asset not ready"

```
    # Planet assets need activation (pipeline handles this automatically)
    # If downloads fail:
    # 1. Check your API key is valid
    # 2. Check you have access to the item type
    # 3. Try with fewer scenes first (limit_per_aoi=5)
    # 4. Reduce max_workers if network issues:

    from planet_pipeline.download import ImageryDownloader
    downloader = ImageryDownloader(
        api_key="your_key",
        storage=storage,
        max_workers=2  # ■■ REDUCE IF HAVING ISSUES
    )
```

■ Getting Help

Check Pipeline Status

```
    # Get detailed summary
    summary = pipeline.get_summary()
    print(summary)

    # Check storage usage
    from planet_pipeline.storage import ImageryStorage
    storage = ImageryStorage(base_dir="./planet_data")  # ■■ YOUR DIR
    usage = storage.get_storage_usage()
    print(f"Total storage: {usage['total'] / 1e9:.2f} GB")

    # List available AOIs
    aois = storage.list_aois()
    print(f"AOIs: {aois}")

    # Check imagery files
    files = storage.get_imagery_files("your_aoi_name")  # ■■ YOUR AOI NAME
    print(f"Found {len(files)} imagery files")
```

Enable Debug Logging

```
    import logging
    logging.basicConfig(level=logging.DEBUG)

    # Now run your pipeline - you'll see detailed logs
```

Resources

• **Planet API Docs:** https://developers.planet.com/docs/apis/

• **Planet Item Types:** https://developers.planet.com/docs/data/items-assets/

• **GeoJSON.io:** http://geojson.io (draw AOIs visually)

• **Planet Explorer:** https://www.planet.com/explorer/ (browse imagery)

■ Summary Checklist

Before running the pipeline, make sure you have:

• [ ] ✓ Installed Python (3.8+)

• [ ] ✓ Created virtual environment (`python -m venv venv`)

• [ ] ✓ Activated virtual environment (see `(venv)` in terminal)

• [ ] ✓ Installed dependencies (`pip install -r requirements.txt`)

• [ ] ✓ Set up API key in `.env` file or environment variable

• [ ] ✓ Created/defined your AOI(s) (GeoJSON or coordinates)

• [ ] ✓ Decided on date range for imagery query

• [ ] ✓ Know what spectral indices you want (or use defaults)

• [ ] ✓ Configured storage directory path

• [ ] ✓ (Optional) Set up labels file if doing ML classification

---

■ Where to Input Your Information - Quick Reference

|---|---|---|

Planet API Key	`.env` file or `api_key=` parameter	`PL_API_KEY=pl1a2b3c4d5e6f`
AOI Coordinates	`geometry=` parameter or GeoJSON file	`[[-122.5, 37.7], ...]`
AOI Name	`name=` parameter in `add_aoi()`	`"san_francisco_bay"`
Date Range	`start_date=` and `end_date=`	`"2024-01-01"` to `"2024-01-31"`
Storage Location	`storage_dir=` parameter	`"./my_data"` or `"/full/path/to/data"`
Imagery Type	`item_types=` parameter	`["PSScene"]` or `["SkySatCollect"]`
Cloud Cover Limit	`cloud_cover_max=` parameter	`0.1` (10%)
Asset Types	`asset_types=` parameter	`["ortho_analytic_4b"]`
Spectral Indices	`indices=` parameter	`["ndvi", "ndwi", "evi"]`
ML Framework	`model_type=` parameter	`"pytorch"` or `"tensorflow"`
Chip Size	`chip_size=` parameter	`256` (pixels)
Train/Val/Test Split	`train_split=`, `val_split=`, `test_split=`	`0.7, 0.15, 0.15`

You're ready to go! Start with Example 1 and customize from there. ■