Skydipper Documentation

Release 0.0.5

Vizzuality

May 25, 2020

Contents:

CHAPTER 1

Example of use

[1]:
```
import Skydipper
```

[2]:
```
Skydipper.__version__
```

[2]:
```
'0.1.10'
```

1.1 Dataset objects

Using known id's you can instantiate a dataset object directly.

[3]:
```
ds = Skydipper.Dataset(id_hash='94241cb4-9e91-4a1a-9fcc-993b8ac9c2b7')
ds
```

[3]:
```
Dataset 94241cb4-9e91-4a1a-9fcc-993b8ac9c2b7 bio.014 Biodiversity Intactness
```

You can access the attributes of a dataset.

[4]:
```
ds.attributes
```

[4]:
```
{'name': 'bio.014 Biodiversity Intactness',
 'slug': 'bio014-Biodiversity-Intactness',
 'type': 'raster',
 'subtitle': None,
 'application': ['skydipper'],
 'dataPath': None,
 'attributesPath': None,
 'connectorType': 'rest',
 'provider': 'gee',
 'userId': '5dc9210cbe566d0010df6752',
 'connectorUrl': None,
 'sources': [],
 'tableName': 'users/resourcewatch/bio_014_bio_intactness',
```

(continues on next page)

(continued from previous page)

```
 'status': 'pending',
 'published': False,
 'overwrite': False,
 'verified': False,
 'blockchain': {},
 'mainDateField': None,
 'env': 'production',
 'geoInfo': False,
 'protected': False,
 'legend': {'date': [],
  'region': [],
  'country': [],
  'nested': [],
  'integer': [],
  'short': [],
  'byte': [],
  'double': [],
  'float': [],
  'half_float': [],
  'scaled_float': [],
  'boolean': [],
  'binary': [],
  'text': [],
  'keyword': []},
 'clonedHost': {},
 'errorMessage': None,
 'taskId': None,
 'createdAt': '2019-11-15T10:25:30.592Z',
 'updatedAt': '2019-11-15T10:25:30.650Z',
 'dataLastUpdated': None,
 'metadata': [],
 'widgetRelevantProps': [],
 'layerRelevantProps': []}
```

You can also access the metadata of a dataset.

```
[ ]: #ds.metadata[0].attributes
```

1.2 Creation of Datasets

If you have a valid API key with permissions you can instantiate a Dataset object from a dictionary. To find out more about keys in the Skydipper REST API you can read our docs.

```
[ ]: token="XXX"
```

```
[ ]: atts = {'name': 'Carto_test1',
      'application': ['skydipper'],
      'connectorType': 'rest',
      'provider': 'cartodb',
      'connectorUrl': 'http://35.233.41.65/user/skydipper/dataset/gadm36_countries',
      'tableName': 'gadm36_countries',
      'env': 'production'}
```

[ ]:
```
ds = Skydipper.Dataset(attributes=atts, token=token)
ds
```

1.3 Update Dataset attributes

Again, update requires a valid API KEY.

[ ]:
```
ds = ds.update(update_params={'name':'My_Carto_table'}, token=token)
ds
```

[ ]:
```
ds.id
```

1.4 Instantiate a Dataset object

If you know the ID you can call a Dataset object directly.

[5]:
```
ds = Skydipper.Dataset(id_hash='9bcc71f2-1492-4c08-a683-472496ab4fdc')
ds
```

[5]:
```
Dataset 9bcc71f2-1492-4c08-a683-472496ab4fdc My_Carto_table
```

1.5 Queries on Datasets

Datasets can be queried via SQL, with a table returned. Currently this is only supported for Carto-type data:

[6]:
```
ds
```

[6]:
```
Dataset 9bcc71f2-1492-4c08-a683-472496ab4fdc My_Carto_table
```

[7]:
```
ds.query('SELECT iso, ST_ASGEOJSON(the_geom_webmercator) FROM data LIMIT 5')
```

[7]:
```
   iso                                          st_asgeojson
0  CHN  {"type":"MultiPolygon","coordinates":[[[[12209...
1  CIV  {"type":"MultiPolygon","coordinates":[[[[-7424...
2  CMR  {"type":"MultiPolygon","coordinates":[[[[12664...
3  COD  {"type":"MultiPolygon","coordinates":[[[[13592...
4  CUB  {"type":"MultiPolygon","coordinates":[[[[-8602...
```

1.6 Layer Objects

Similarly, you can also instantiate an existing Layer object.

[8]:
```
ly = Skydipper.Layer(id_hash='f685085c-9103-4e87-997f-a6c088c52614')
ly
```

[8]:
```
Layer f685085c-9103-4e87-997f-a6c088c52614 GADM36 Updated
```

[9]:
```
# You will see that layers may have SQL and Cartocss or mapbox styles attached, which
↪can be used to visulize the data.
ly.attributes.get('layerConfig').get('body')
```

[9]:
```
{'layers': [{'options': {'cartocss': "#layer[unregion2='Europe']  { polygon-fill:
↪#826DBA; polygon-opacity: 0.9; ::outline { line-color: #FFFFFF; line-width: 1; line-
↪opacity: 0.5; } } #layer[unregion2='Africa']  { polygon-fill: #cf003e; polygon-
↪opacity: 0.9; ::outline { line-color: #FFFFFF; line-width: 1; line-opacity: 0.5; } }
↪ #layer[unregion2='Americas']  { polygon-fill: #cf9700; polygon-opacity: 0.9; ::
↪outline { line-color: #FFFFFF; line-width: 1; line-opacity: 0.5; } } #layer[iso='ESP
↪']  { polygon-fill: #5aa91f; polygon-opacity: 0.9; ::outline { line-color: #FFFFFF;
↪line-width: 1; line-opacity: 0.5; } }",
    'cartocss_version': '2.3.0',
    'sql': 'SELECT * FROM gadm36_countries'},
   'type': 'cartodb'}],
 'maxzoom': 19,
 'minzoom': 2}
```

Layers can be visulized if appropriate via a call to Layer().map()

[10]:
```
ly.map()
```

[10]:
```
<folium.folium.Map at 0x1216c24a8>
```

1.7 Collection objects: Searching

If you don't know what data you are interested in advance, you can search by keywords and return a list of objects.

[11]:
```
c = Skydipper.Collection('a', object_type=['layer','dataset'], app=['skydipper'],
↪limit=10)
```

[12]:
```
c
```

[12]:
```
[0. Dataset 4b34879d-3648-4b21-83e6-b20d8a9f0472 Blah_CSV,
 1. Dataset fb2683e2-3280-46fe-b2cc-8dee034b602d Carto_test,
 2. Dataset a39a7f7f-85e1-4332-ba9d-2eae29768394 Carto_test,
 3. Dataset f15c2377-e1f9-4062-89b0-1652f99a167c Carto_test,
 4. Dataset 31fb1f7c-edd9-47a1-9e1a-4bbc73cacdf1 Carto_test1,
 5. Dataset 7ed9f27e-2d53-476e-ad3d-deba75ae3000 Carto_test1,
 6. Dataset 1cb193a7-4df2-40d7-97ac-4215e53b8e0d Carto_testXX,
 7. Dataset d3a1184e-2a6f-48f5-b920-16fff69e7132 Carto_testXX,
 8. Dataset 8ed89afd-0ffa-4218-8b96-b585a80ddb9c First_Carto_table,
 9. Dataset d69ac786-4fe0-4f7b-831e-49c869dd85a5 Landsat 7 Surface Reflectance]
```

Searching can be restricted with keyword arguments to specifically search types of items, applications, and more. If you want to render those items, you will need to do the following.

You can access items from a collection using subscripts, slices and more. Note that slicing, or selecting by element instantiates the Layer, Table, or Dataset object.

[13]:
```
c[0:3]
```

[13]:
```
[Dataset 4b34879d-3648-4b21-83e6-b20d8a9f0472 Blah_CSV,
 Dataset fb2683e2-3280-46fe-b2cc-8dee034b602d Carto_test,
 Dataset a39a7f7f-85e1-4332-ba9d-2eae29768394 Carto_test]
```

[14]:
```
c[-1]
```

[14]:
```
Dataset d69ac786-4fe0-4f7b-831e-49c869dd85a5 Landsat 7 Surface Reflectance
```

1.8 Create a Geometry object

Often you will need to perform some kind of intersect analysis between data held in datasets and tables and a geometry. We will now show you multiple ways to create your geometry objects.

1.8.1 From an ID

Vizzuality's API holds geometry objects as a Geostore item. Geostore items are accessed by an id-hash. If you know the hash of your object already you can simply call a geometry like so:

[3]:
```
g = Skydipper.Geometry(id_hash='de43241398f124ec2e3d6a3720439a99')
g
```

[3]:
```
Geometry de43241398f124ec2e3d6a3720439a99
```

1.8.2 Geometry attributes

The attributes can be accessed as a dictionary.

[4]:
```
g.attributes
```

[4]:
```
{'geojson': {'features': [{'type': 'Feature',
    'geometry': {'type': 'Polygon',
     'coordinates': [[[28.000041976337, 49.7101919873524],
       [28.000041976337, 48.1873700139574],
       [27.7501030114934, 48.1873700139574],
       [27.5001640466497, 48.1873700139574],
       [27.250225081806, 48.1873700139574],
       [26.9998283532904, 48.1873700139574],
       [26.9998283532904, 49.7101919873524],
       [27.250225081806, 49.7101919873524],
       [27.5001640466497, 49.7101919873524],
       [27.7501030114934, 49.7101919873524],
       [28.000041976337, 49.7101919873524]]]}}],
  'crs': {},
  'type': 'FeatureCollection'},
 'hash': 'de43241398f124ec2e3d6a3720439a99',
 'provider': {},
 'areaHa': 1239540.338786885,
 'bbox': [26.9998283532904,
  48.1873700139574,
  28.000041976337,
  49.7101919873524],
 'lock': False,
 'info': {'use': {}}}
```

1.8.3 Geometry as a Table

Table method returns a dataframe of the geometry object. Map will add a Folium map with the geomerty rendered.

[17]:
```
g.table()
```

[17]:
```
          areaHa                                                bbox  \
0  1.239540e+06  [26.9998283532904, 48.1873700139574, 28.000041...

                                                geometry  \
0  POLYGON ((28.00004 49.71019, 28.00004 48.18737...

                                  id use
0  de43241398f124ec2e3d6a3720439a99  {}
```

1.8.4 Mapping the Geometry

Calling .map() will create a Folium map with the geomerty rendered.

[18]:
```
g.map()
```

[18]:
```
<folium.folium.Map at 0x1216f1048>
```

1.8.5 From Geojson - Points

You can create an object as you need on the fly from geojson. The act of creating an object will also register it to a Geostore service of your choice (locally, or on a remote server). You can create a geometry object from geojson Points and MultiPoints type data as follows:

[19]:
```
atts = {'geojson': {'type': 'FeatureCollection',
  'features': [{'type': 'Feature',
    'properties': {},
    'geometry': {'type': 'MultiPoint', 'coordinates': [[-4.29, 39.1097]]}}]}}

point = Skydipper.Geometry(attributes=atts)
point
```

[19]:
```
Geometry 09405592c74fc4f2443d8245ddc0a45e
```

1.8.6 From Geojson - Polygons

You can create an object as you need on the fly from geojson. The act of creating an object will also register it to a Geostore service of your choice (locally, or on a remote server). You can create a geometry object from Geojson Polygon and Multipolygon type data as follows:

[5]:
```
atts={'geojson': {'type': 'FeatureCollection',
  'features': [{'type': 'Feature',
    'properties': {},
    'geometry': {'type': 'Polygon',
     'coordinates': [[[82.265625, 32.84267363195431],
       [77.34374999999999, 27.059125784374068],
       [85.4296875, 22.268764039073968],
       [90.3515625, 28.304380682962783],
       [87.5390625, 32.54681317351514],
```

(continues on next page)

(continued from previous page)

```
       [82.265625, 32.84267363195431]]]}}]}}


g1 = Skydipper.Geometry(attributes=atts)
g1
```

[5]:
```
Geometry 75857ff50bc4cbaa687bfdf5070ffdfa
```

[6]:
```
g1.map()
```

[6]:
```
<folium.folium.Map at 0x118e793c8>
```

1.8.7 From a Shapely object

Shapely objects are at the root of popular python geolibraries such as Geopandas. We can recieve those geometry objects and create a Geometry object (simultaneously registering it in a Vizzuality Geostore server).

[ ]:
```
import geopandas as gpd
```

[ ]:
```
%%writefile ./sample.geojson
{"features":[{"properties":null,"type":"Feature","geometry":{"type":"Polygon",
↪"coordinates":[[[-43.1343734264374,-8.07358087603511],[-43.1327533721924,-8.
↪08277985402466],[-43.1298887729645,-8.08181322762719],[-43.1103515625,-8.
↪07815914647929],[-43.1094932556152,-8.07799981079283],[-43.1094932556152,-8.
↪09641859926744],[-43.1103515625,-8.09645046495416],[-43.1187307834625,-8.
↪0967372560211],[-43.1186878681183,-8.10273857778317],[-43.1186771392822,-8.
↪10358831522616],[-43.1476235389709,-8.10358831522616],[-43.1477630138397,-8.
↪10273857778317],[-43.1505310535431,-8.08645513764317],[-43.1517112255096,-8.
↪08057041885644],[-43.1439757347107,-8.0795931648273],[-43.1448876857758,-8.
↪07574785969913],[-43.1343734264374,-8.07358087603511]]]}}],"crs":{},"type":
↪"FeatureCollection"}
```

[ ]:
```
df = gpd.read_file('./sample.geojson')
df
```

[ ]:
```
s = df.geometry[0]
print(f"Hello!  I am a {type(s)}")
s
```

[ ]:
```
g = Skydipper.Geometry(s=s)
```

[ ]:
```
g
```

1.9 Describe a Geometry

Return a title and textual description of a geometry in any language.

[22]:
```
g.map()
```

[22]:
```
<folium.folium.Map at 0x12174b4e0>
```

[23]:
```
%%time
g.describe()
```

```
Title: Area between Ukraine and Moldova, in Europe
CPU times: user 22.4 ms, sys: 3.46 ms, total: 25.8 ms
Wall time: 4.19 s
```

[24]:
```
g.describe(lang='es') # same description but this time in Spanish
```

```
Title: Área entre Moldavia y Ucrania, en Europa
```

Second example of geodescriber, but considering a larger area.

[9]:
```
atts={'geojson': {'type': 'FeatureCollection',
  'features': [{'type': 'Feature',
    'properties': {},
    'geometry': {'type': 'Polygon',
     'coordinates': [[[82.265625, 32.84267363195431],
       [77.34374999999999, 27.059125784374068],
       [85.4296875, 22.268764039073968],
       [90.3515625, 28.304380682962783],
       [87.5390625, 32.54681317351514],
       [82.265625, 32.84267363195431]]]}}]}}


g2 = Skydipper.Geometry(attributes=atts)
g2
```

[9]:
```
Geometry 75857ff50bc4cbaa687bfdf5070ffdfa
```

[26]:
```
g2.map()
```

[26]:
```
<folium.folium.Map at 0x121717dd8>
```

[10]:
```
g2.describe()
```

```
Title: Area between Nepal and Bangladesh, in Asia
```

After running describe, the results are avaiable as geometry.description

[28]:
```
g2.description
```

[28]:
```
{'title': 'Area near Nepal, Asia',
 'description': 'The region is made up of different habitats, including Central␣
↪Tibetan Plateau alpine steppe, and Upper Gangetic Plains moist deciduous forests.␣
↪This region contains some Intact Forest. The most common environmental conditions␣
↪of the area are polar tundra climate. The region is made up of several types of␣
↪biomes, including Montane Grasslands and Shrublands, and Tropical and Subtropical␣
↪Moist Broadleaf Forests. The location is predominantly land area. Area of 891,
↪255km² located in a mix of lowland and mountainous areas.',
 'lang': 'en'}
```

1.10 Simplify a Geostore

[11]:
```
g3 = g2.simplify(tolerance=10)
g3
```

[11]:
```
Geometry e0f2effe4ee070f6d91ddc6814df0b27
```

[12]:
```
g3.map()
```

[12]:
```
<folium.folium.Map at 0x119a6ec88>
```

1.11 Add a satellite image to the map of your Geometry

1.11.1 Single images

The Geometry().map() method supports the return of recent satellite imagery. We calculate the centroid of your geometry and return the best, most recent image that intersects with that point. This is simply to provide context to your geometry. For point geometries we return the best cloud-free image we can find within a specified date range.

[29]:
```
# Create a point

atts = {'geojson': {'type': 'FeatureCollection',
  'features': [{'type': 'Feature',
    'properties': {},
    'geometry': {'type': 'MultiPoint', 'coordinates': [[-4.29, 39.1097]]}}]}}

point = Skydipper.Geometry(attributes=atts)
point
```

[29]:
```
Geometry 09405592c74fc4f2443d8245ddc0a45e
```

[30]:
```
point.map(image=True, start='2018-01-01', end='2018-12-31')
```

[30]:
```
<folium.folium.Map at 0x12176a390>
```

1.11.2 Composite images

Whereas for polygon-type geoms we return a cloud-free composite image clipped to your geometry.

[31]:
```
# Create a polygon

atts={'geojson': {'type': 'FeatureCollection',
  'features': [{'type': 'Feature',
    'properties': {},
    'geometry': {'type': 'Polygon',
     'coordinates': [[[-0.87890625, 43.329173667843904],
       [-1.6149902343749998, 42.75104599038353],
       [-1.1865234375, 42.35854391749705],
       [-0.6427001953125, 42.755079545072135],
       [-0.45043945312499994, 42.9524020856897],
       [-0.87890625, 43.329173667843904]]]}}]}

g1 = Skydipper.Geometry(attributes=atts, server='https://production-api.
↪globalforestwatch.org')
g1
```

[31]:
```
Geometry 99e57dfb0832d621c485a7ca9e3a0160
```

[32]:
```
g1.map(image=True, instrument='sentinel')
```

[32]:
```
<folium.folium.Map at 0x12177d4e0>
```

1.12 Display a Layer and Geometry together

[33]:
```
atts={'geojson': {'type': 'FeatureCollection',
  'features': [{'type': 'Feature',
    'properties': {},
    'geometry': {'type': 'Polygon',
     'coordinates': [[[-48.131103515625, -0.6001172008725418],
       [-48.13934326171875, -0.9791088369866402],
       [-47.86331176757812, -0.9818550168696459],
       [-47.616119384765625, -0.8459165322899671],
       [-47.50213623046875, -0.7182123915862891],
       [-47.51861572265625, -0.5287095375108173],
       [-47.882537841796875, -0.5163504323777461],
       [-48.131103515625, -0.6001172008725418]]]}}]}}
g = Skydipper.Geometry(attributes=atts)
l = Skydipper.Layer(id_hash='e7070d5f-3d38-46b1-86eb-e98782da55dd')
l
```

[33]:
```
Layer e7070d5f-3d38-46b1-86eb-e98782da55dd 2005 Biodiversity Intactness (%)
```

[34]:
```
l.map(geometry=g)
```

[34]:
```
<folium.folium.Map at 0x12176a198>
```

1.13 Intersecting Raster data with Geometry objects

Any Layer or Dataset objects based on an Earth Engine raster can be intersected with Skydipper.Geometry objects.

[35]:
```
g
```

[35]:
```
Geometry 5208143898c498db610b9d2290dffb68
```

[38]:
```
l
```

[38]:
```
Layer e7070d5f-3d38-46b1-86eb-e98782da55dd 2005 Biodiversity Intactness (%)
```

Finally, you can call the intersect function on the dataset or layer object to see a dictionary of values.

[39]:
```
l.intersect(geometry=g)
```

[39]:
```
{'b1': {'count': 2176,
  'max': 1.0728240013122559,
  'mean': 0.9838019005727772,
  'min': 0.8430513739585876,
  'stdev': 0.037301282181494394,
  'sum': 2086.5743862748154}}
```

1.13.1 Layer Intersections

Intersections against layers work in the same manner. Here we show an intersect between an area along Brazil's coast and a Mangrove biomass density Layer.

1.14 ImageCollection: Search for satellite images

You can search for Landsat-8 and Sentinel-2 imagery using the ImageCollection module as follows:

[40]:
```
ic = Skydipper.ImageCollection(lon=28.3,
                               lat=-16.6,
                               start='2018-08-01',
                               end='2018-08-10') # n.b. lon/lat are temporarily␣
↪flipped
```

[41]:
```
ic
```

[41]:
```
[0. Image Sentinel-2A 2018-08-08 07:56:11Z,
 1. Image LANDSAT_8 2018-08-05 08:11:14Z,
 2. Image Sentinel-2B 2018-08-03 07:56:09Z]
```

[42]:
```
type(ic)
```

[42]:
```
Skydipper.imageCollection.ImageCollection
```

All types of pythonic list mainipulation are supported on the image collection results. Including iteration:

[43]:
```
for i in ic[0:2]:
    print(i)
```

```
Image COPERNICUS/S2/20180808T075611_20180808T081515_T35KPB
Image LANDSAT/LC08/C01/T1_RT_TOA/LC08_172071_20180805
```

. . . and subsetting to access individual Images.

1.15 Images: Individual satellite tiles

[44]:
```
i = ic[0]
i
```

[44]:
```
Image COPERNICUS/S2/20180808T075611_20180808T081515_T35KPB
```

[45]:
```
type(i)
```

[45]:
```
Skydipper.image.Image
```

You can access the Image attributes, which indicate the provinence of the tile.

[46]:
```
i.attributes
```

[46]:
```
{'provider': 'COPERNICUS/S2/20180808T075611_20180808T081515_T35KPB'}
```

1.15.1 Mapping Satellite Images

You can also display web-map tiles and the bounding-box of the satellite image.

[47]:
```
i.map()
```

[47]:
```
<folium.folium.Map at 0x12bdbadd8>
```

1.15.2 Classification of individual Satellite Images

[ ]:
```
#classified = i.classify()
#classified
```

[49]:
```
#classified.map()
```

1.15.3 Composite and classify satellite image collections

First grab a collection, specifying a point and time period.

[ ]:
```
ic = Skydipper.ImageCollection(lon=28.271979, lat=-16.457814, start='2018-06-01', end=
↪'2018-06-20')
ic
```

Next ask for a composite image based on your collection. By default it will be for Sentinel-2, but you may change this via an argument to Landsat.

[ ]:
```
i = ic.composite()
i
```

You can then map your composite image.

[ ]:
```
i.map()
```

You can take your composite image, and apply a land cover classifier based on our pre-trained models. These are Deepvel, and Segnet.

[ ]:
```
c = i.classify(model_type='segnet', version='v2')
c
```

[ ]:
```
c.map()
```

CHAPTER 2

Utilities

2.1 Generate Movie Map Tiles

- Step 1: create a visulised image collection
- Step 2: create a geometry that represents that area within which you want to generate tiles for
- Step 3: obtain credentials to write to a GCS bucket (including a JSON token)
- Step 4: run the movie tile code below

[ ]:
```
from Skydipper.utils import MovieMaker
import ee
ee.Initialize()
```

[ ]:
```
# example of creating a valid image collection
# Composite EVI collection Example based on https://developers.google.com/earth-
↪engine/tutorials/community/modis-ndvi-time-series-animation

col = ee.ImageCollection('MODIS/006/MOD13A2').select('NDVI') # MODIS NDVI Data
mask = ee.FeatureCollection('USDOS/LSIB_SIMPLE/2017')  # All world shapes

# Add day-of-year (DOY) property to each image.
def add_DOY_atts(img):
    doy = ee.Date(img.get('system:time_start')).getRelative('day', 'year')
    return img.set('doy', doy)

col = col.map(add_DOY_atts)

# Get a collection of distinct images by 'doy'.
distinctDOY = col.filterDate('2013-01-01', '2014-01-01')

# Define a filter that identifies which images from the complete
# collection match the DOY from the distinct DOY collection.
filter_img = ee.Filter.equals(leftField= 'doy', rightField= 'doy')
```

(continues on next page)

(continued from previous page)

```
# Define a join.
join = ee.Join.saveAll('doy_matches')

# Apply the join and convert the resulting FeatureCollection to an
# ImageCollection.
joinCol = ee.ImageCollection(join.apply(distinctDOY, col, filter_img))

# Apply median reduction among matching DOY collections.
def doyCol(img):
    tmp_i = ee.ImageCollection.fromImages(img.get('doy_matches'))
    return tmp_i.reduce(ee.Reducer.median())

comp = joinCol.map(doyCol)

# Define RGB visualization parameters.
visParams = {
  'min': 0.0,
  'max': 9000.0,
  'palette': [
    'FFFFFF', 'CE7E45', 'DF923D', 'F1B555', 'FCD163', '99B718', '74A901',
    '66A000', '529400', '3E8601', '207401', '056201', '004C00', '023B01',
    '012E01', '011D01', '011301'
  ],
}


# Create an RGB visualization images for use as animation frames.
def imgViz(img):
    return img.visualize(**visParams)#.clip(mask)

ic = comp.map(imgViz)
```

[ ]:
```
canary_area = [
    [-18.617344317591915,26.23725199497126],
    [-11.849766192591915,26.23725199497126],
    [-11.849766192591915,29.824435938587445],
    [-18.617344317591915,29.824435938587445],
    [-18.617344317591915,26.23725199497126]
]

area = ee.Geometry.Polygon(canary_area)
```

[ ]:
```
m = MovieMaker(area=area, zlist=[1, 2, 3, 4, 5, 6], ic=ic, bucket_name='skydipper_
↪materials', folder_path='movie-tiles/DTEST',
               privatekey_path="/Users/me/.privateKeys/key.json", report_status=True)
m.run()
```

[ ]:
```
# After the jobs have finished you can run this to correctly name the exported tiles
m.reNamer()
```

2.2 Construct Widgets

A demo of how to use Skydipper to construct widgets like those on our projects such as Global Forest Watch Dashboard pages.

2.2.1 Tree cover example

[ ]:
```
import matplotlib.pyplot as plt
%matplotlib inline
```

[ ]:
```
# Get a datatable (Hansen)
table = Skydipper.Table('a20e9c0e-8d7d-422f-90f5-3b9bca355aaf')
table
```

[ ]:
```
iso = 'BRA'
administration = 1

sql = f"""
        SELECT
            SUM(area_extent) as value,
            SUM(area_admin) as total_area
        FROM data
        WHERE iso = '{iso}'
        AND adm1 = {administration}
        AND thresh = 30
        AND polyname = 'admin'
"""

results = table.query(sql=sql)
results
```

[ ]:
```
sizes = [results.value[0], results.total_area[0] - results.value[0]]
colors = ['green','grey']
labels = ['Tree cover', 'Non-forest']

fig1, ax1 = plt.subplots()
ax1.pie(sizes, labels=labels, autopct='%1.1f%%',
        shadow=False, startangle=90, colors=colors)
ax1.axis('equal')
centre_circle = plt.Circle((0,0),0.75,color='black', fc='white',linewidth=0.5)
fig1 = plt.gcf()
fig1.gca().add_artist(centre_circle)
plt.suptitle('Tree cover extent')
plt.title(f'in {iso}/{administration}')
plt.show()
```

2.2.2 Tree cover loss example

[ ]:
```
sql = """
SELECT
    polyname, year_data.year as year,
    SUM(year_data.area_loss) as area
```

(continues on next page)

(continued from previous page)

```
FROM data
WHERE polyname = 'admin'
AND thresh= 30
GROUP BY polyname, iso, nested(year_data.year)
"""
global_loss = table.query(sql=sql)
global_loss.head()
```

[ ]:
```
iso='BRA'
loss_data = list(global_loss[global_loss['iso'] == f'{iso}']['area'])
years = list(global_loss[global_loss['iso'] == f'{iso}']['year'])

width = 0.66
fig, ax = plt.subplots()
rects1 = ax.bar(years, loss_data, width, color='#FE5A8D')

# add some text for labels, title and axes ticks
ax.set_ylabel('Loss extent (ha)')
ax.set_title(f'Loss by year in {iso}')
plt.show()
```

2.3 Creating a local backup of Data objects

Save a local backup of a collection to a specified path. This creates a folder containing a JSON for each dataset and it's associated Layers, Metadata and Vocabularies.

[ ]:
```
col = Skydipper.Collection(app=['gfw'], env='production')
```

[ ]:
```
path = './LMI-BACKUP'
```

[ ]:
```
col.save(path)
```

2.4 Load Data objects from local backup

You can also restore a previous version from local backup.

[ ]:
```
import os
```

[ ]:
```
files = os.listdir(path)[0:3]
files
```

[ ]:
```
ds_id = files[0].split('.json')[0]
ds_id
```

[ ]:
```
dataset = Skydipper.Dataset(ds_id)
```

[ ]:
```
dataset
```

[ ]:
```
backup = dataset.load(path)
```

[ ]:
```
backup
```

[ ]:
```

```

CHAPTER 3

Indices and tables

- genindex
- modindex
- search