# Vector Data I/O in Python

Reading data into Python is usually the first step of an analysis workflow. There are various different GIS data formats available such as Shapefile (https://en.wikipedia.org/wiki/Shapefile), GeoJSON (https://en.wikipedia.org/wiki/GeoJSON), KML (https://en.wikipedia.org/wiki/Keyhole_Markup_Language), and GPKG (https://en.wikipedia.org/wiki/GeoPackage). Geopandas (http://geopandas.org/io.html) is capable of reading data from all of these formats (plus many more).

This tutorial will show some typical examples how to read (and write) data from different sources. The main point in this section is to demonstrate the basic syntax for reading and writing data using short code snippets. You can find the example data sets in the data-folder. However, most of the example databases do not exists, but you can use and modify the example syntax according to your own setup.

## File formats

In geopandas, we use a generic function from_file() (http://geopandas.org/reference.html#geopandas.GeoDataFrame.to_file) for reading in different data formats. In the bacground, Geopandas uses fiona.open() (https://fiona.readthedocs.io/en/latest/fiona.html#fiona.open) when reading in data. Esri Shapefile is the default file format. For other file formats we need to specify which driver to use for reading in the data.

You can check supported through geopandas, or directly from fiona:

In [1]:
```
import geopandas as gpd

# Check supported format drivers
gpd.io.file.fiona.drvsupport.supported_drivers

# Same as:
#import fiona
#fiona.supported_drivers
```

Out[1]:
```
{'AeronavFAA': 'r',
 'ARCGEN': 'r',
 'BNA': 'rw',
 'DXF': 'rw',
 'CSV': 'raw',
 'OpenFileGDB': 'r',
 'FlatGeobuf': 'r',
 'ESRIJSON': 'r',
 'ESRI Shapefile': 'raw',
 'GeoJSON': 'raw',
 'GeoJSONSeq': 'rw',
 'GPKG': 'raw',
 'GML': 'rw',
 'OGR_GMT': 'rw',
 'GPX': 'rw',
 'GPSTrackMaker': 'rw',
 'Idrisi': 'r',
 'MapInfo File': 'raw',
 'DGN': 'raw',
 'PCIDSK': 'rw',
 'OGR_PDS': 'r',
 'S57': 'r',
 'SEGY': 'r',
 'SUA': 'r',
 'TopoJSON': 'r'}
```

## Read / write Shapefile

In [4]:
```
import geopandas as gpd

# Read file from Shapefile
fp = "data/finland_municipalities.shp"
data = gpd.read_file(fp)

# Write to Shapefile (just make a copy)
outfp = "temp/finland_municipalities.shp"
data.to_file(outfp)
```

In [ ]:
```
```

## Read / write GeoJSON

In [5]:
```
# Read file from GeoJSON
fp = "data/finland_municipalities.gjson"
data = gpd.read_file(fp, driver="GeoJSON")

# Write to GeoJSON (just make a copy)
outfp = "temp/finland_municipalities.gjson"
data.to_file(outfp, driver="GeoJSON")
```

## Read / write KML

In [6]:
```
# Enable KML driver
gpd.io.file.fiona.drvsupport.supported_drivers['KML'] = 'rw'

# Read file from KML
fp = "data/finland_municipalities.kml"
data = gpd.read_file(fp)

# Write to KML (just make a copy)
outfp = "temp/finland_municipalities.kml"
data.to_file(outfp, driver="KML")
```

```
---------------------------------------------------------------------------
DriverError                               Traceback (most recent call last)
<ipython-input-6-57e391911fd1> in <module>
      4 # Read file from KML
      5 fp = "data/finland_municipalities.kml"
----> 6 data = gpd.read_file(fp)
      7 
      8 # Write to KML (just make a copy)

/srv/conda/envs/notebook/lib/python3.8/site-packages/geopandas/io/file.py in _read_file(filename, bbox, mask, rows, **kwargs)
     94 
     95     with fiona_env():
---> 96         with reader(path_or_bytes, **kwargs) as features:
     97 
     98             # In a future Fiona release the crs attribute of features will

/srv/conda/envs/notebook/lib/python3.8/site-packages/fiona/env.py in wrapper(*args, **kwargs)
    396     def wrapper(*args, **kwargs):
    397         if local._env:
--> 398             return f(*args, **kwargs)
    399         else:
    400             if isinstance(args[0], str):

/srv/conda/envs/notebook/lib/python3.8/site-packages/fiona/__init__.py in open(fp, mode, driver, schema, crs, encoding, layer, vfs, enabled_drivers, crs_wkt, **kwargs)
    251 
    252         if mode in ('a', 'r'):
--> 253             c = Collection(path, mode, driver=driver, encoding=encoding,
    254                            layer=layer, enabled_drivers=enabled_drivers, **kwargs)
    255         elif mode == 'w':

/srv/conda/envs/notebook/lib/python3.8/site-packages/fiona/collection.py in __init__(self, path, mode, driver, schema, crs, encoding, layer, vsi, archive, enabled_drivers, crs_wkt, ignore_fields, ignore_geometry, **kwargs)
    161 
    162         if self.session is not None:
--> 163             self.guard_driver_mode()
    164 
    165         if self.mode in ("a", "w"):

/srv/conda/envs/notebook/lib/python3.8/site-packages/fiona/collection.py in guard_driver_mode(self)
    178         driver = self.session.get_driver()
    179         if driver not in supported_drivers:
--> 180             raise DriverError("unsupported driver: %r" % driver)
    181         if self.mode not in supported_drivers[driver]:
```

```
    182             raise DriverError("unsupported mode: %r" % self.m
ode)

DriverError: unsupported driver: 'LIBKML'
```

### Read / write Geopackage

In [ ]:
```
# Read file from Geopackage
fp = "data/finland_municipalities.gpkg"
data = gpd.read_file(fp)

# Write to Geopackage (just make a copy)
outfp = "temp/finland_municipalities.gpkg"
data.to_file(outfp, driver="GPKG")
```

### Read / write GeoDatabase

In [ ]:
```
# Read file from File Geodatabase
fp = "data/finland.gdb"
data = gpd.read_file(fp, driver="OpenFileGDB", layer='municipalities')

# Write to same FileGDB (just add a new layer) - requires additional p
ackage installations(?)
#outfp = "data/finland.gdb"
#data.to_file(outfp, driver="FileGDB", layer="municipalities_copy")
```

### Read / write MapInfo Tab

In [ ]:
```
# Read file from MapInfo Tab
fp = "data/finland_municipalities.tab"
data = gpd.read_file(fp, driver="MapInfo File")

# Write to same FileGDB (just add a new layer)
outfp = "temp/finland_municipalities.tab"
data.to_file(outfp, driver="MapInfo File")
```

## Databases

Example syntax for reading and writing data from/to databases.

### Read PostGIS database using psycopg2

In [ ]:
```
import geopandas as gpd
import psycopg2

# Create connection to database with psycopg2 module (update params ac
cording your db)
conn, cursor = psycopg2.connect(dbname='my_postgis_database',
                                user='my_usrname',
                                password='my_pwd',
                                host='123.22.432.16', port=5432)

# Specify sql query
sql = "SELECT * FROM MY_TABLE;"

# Read data from PostGIS
data = gpd.read_postgis(sql=sql, con=conn)
```

## Read / write PostGIS database using SqlAlchemy + GeoAlchemy

In [ ]:
```
from sqlalchemy.engine.url import URL
from sqlalchemy import create_engine
from sqlalchemy import MetaData
from sqlalchemy.orm import sessionmaker
from geoalchemy2 import WKTElement, Geometry

# Update with your db parameters
HOST = '123.234.345.16'
DB = 'my_database'
USER = 'my_user'
PORT = 5432
PWD = 'my_password'

# Database info
db_url = URL(drivername='postgresql+psycopg2', host=HOST, database=DB,
                   username=USER, port=PORT, password=PWD)

# Create engine
engine = create_engine(db_url)

# Init Metadata
meta = MetaData()

# Load table definitions from db
meta.reflect(engine)

# Create session
Session = sessionmaker(bind=engine)
session = Session()

# ========================
# Read data from PostGIS
# ========================

# Specify sql query
sql = "SELECT * FROM finland;"

# Pull the data
data = gpd.read_postgis(sql=sql, con=engine)

# Close session
session.close()

# =========================================
# Write data to PostGIS (make a copy table)
# =========================================

# Coordinate Reference System (srid)
crs = 4326

# Target table
target_table = 'finland_copy'

# Convert Shapely geometries to WKTElements into column 'geom' (defaul
t in PostGIS)
```

```
data['geom'] = data['geometry'].apply(lambda row: WKTElement(row.wkt,
srid=crs))

# Drop Shapely geometries
data = data.drop('geometry', axis=1)

# Write to PostGIS (overwrite if table exists, be careful with this! )
# Possible behavior: 'replace', 'append', 'fail'

data.to_sql(target_table, engine, if_exists='replace', index=False)
```

## Read / write Spatialite database

In [ ]:
```
import geopandas as gpd
import sqlite3
import shapely.wkb as swkb
from sqlalchemy import create_engine, event

# DB path
dbfp = 'L2_data/Finland.sqlite'

# Name for the table
tbl_name = 'finland'

# SRID (crs of your data)
srid = 4326

# Parse Geometry type of the input Data
gtype = data.geom_type.unique()
assert len(gtype) == 1, "Mixed Geometries! Cannot insert into SQLite t
able."
geom_type = gtype[0].upper()

# Initialize database engine
engine = create_engine('sqlite:///{db}'.format(db=dbfp), module=sqlit
e)

# Initialize table without geometries
geo = data.drop(['geometry'], axis=1)

with sqlite3.connect(dbfp) as conn:
    geo.to_sql(tbl_name, conn, if_exists='replace', index=False)

# Enable spatialite extension
with sqlite3.connect(dbfp) as conn:
    conn.enable_load_extension(True)
    conn.load_extension("mod_spatialite")
    conn.execute("SELECT InitSpatialMetaData(1);")
    # Add geometry column with specified CRS with defined geometry typ
ehaving two dimensions
    conn.execute(
        "SELECT AddGeometryColumn({table}, 'wkb_geometry',\
        {srid}, {geom_type}, 2);".format(table=tbl_name, srid=srid, ge
om_type=geom_type)
    )

# Convert Shapely geometries into well-known-binary format
data['geometry'] = data['geometry'].apply(lambda geom: swkb.dumps(geo
m))

# Push to database (overwrite if table exists)
data.to_sql(tbl_name, engine, if_exists='replace', index=False)
```

## Read Web Feature Service (WFS)

This script was used to generate input data for this tutorial (FileGDB and tab were created separately). Source: Statistics finland WFS.

In [ ]:
```
import geopandas as gpd
import requests
import geojson
from pyproj import CRS

# Specify the url for the backend.
#Here we are using data from Statistics Finland: https://www.stat.fi/o
rg/avoindata/paikkatietoaineistot_en.html. (CC BY 4.0)
url = 'http://geo.stat.fi/geoserver/tilastointialueet/wfs'

# Specify parameters (read data in json format).
params = dict(service='WFS', version='2.0.0', request='GetFeature',
         typeName='tilastointialueet:kunta4500k', outputFormat='json')

# Fetch data from WFS using requests
r = requests.get(url, params=params)

# Create GeoDataFrame from geojson and set coordinate reference system
data = gpd.GeoDataFrame.from_features(geojson.loads(r.content),  crs="
EPSG:3067")
```

In [ ]:
```
data.head()
```

In [ ]:
```
# Prepare data for writing to various file formats
data = data.drop(columns=["bbox"])
```

In [ ]:
```
# Check crs
data.crs
```

In [ ]:
```
# filename
layer_name = "finland_municipalities"

# enable writing kml
gpd.io.file.fiona.drvsupport.supported_drivers['KML'] = 'rw'

# drivers and extensions for different file formats
drivers = {'ESRI Shapefile': 'shp',
           'GeoJSON': 'gjson',
           'KML': 'kml',
           'GPKG': 'gpkg',
          }

# Write layer to different file formats
for driver, extension in drivers.items():

    # Create file path and file name
    file_name = "data/{0}.{1}".format(layer_name, extension)

    # Write data using correct dricer
    data.to_file(file_name, driver=driver)
    print("Created file", file_name)
```

In [ ]: