

Lec7



explain the code line by line and if there is error in the execution of the code then remove it by updating the code snippet and state the reason for the update



Could you please provide the specific code snippet you would like to have explained and updated? This will allow me to walk through it line by line and offer any necessary updates.



GeoParquet data analysis

This notebook is for the basic geospatial analysis using geoparquet file format. The input data can be downloaded from [nz-building-outlines.parquet](#).

```
! pip install pyarrow
```

```
! pip install geopandas==1.0.0
```

Reading and writing

```
import geopandas as gpd
```

```
gpd.__version__
```

```
'1.0.0'
```

```
# Load the geoparquet file
```

```
gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
```

```
gdf.head()
```

```
building_id  name  use suburb_locality  town_city  territorial_authority  capture_method
capture_source_group  capture_source_id  capture_source_name  capture_source_from
capture_source_to  last_modified  geometry
```

```
0  2292028  Unknown  Akitio  Tararua District  Feature Extraction  NZ Aerial
Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27 2016-04-21
2019-01-04  MULTIPOLYGON (((1890386.169 5501282.718, 18903...
```

```
1  2292051  Unknown  Marton  Marton  Rangitikei District  Feature Extraction  NZ
Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1796386.756 5560662.126, 17963...
```

```
2  2292290  Unknown  Rangataua  Ruapehu District  Feature Extraction  NZ Aerial
Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27 2016-04-21
2019-01-04  MULTIPOLYGON (((1811431.727 5632559.044, 18114...
```

```
3  2292353  Unknown  Durie Hill  Whanganui  Whanganui District  Feature Extraction
NZ Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1776387.292 5576785.712, 17763...
```

```
4  2292407  Unknown  Durie Hill  Whanganui  Whanganui District  Feature Extraction
NZ Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1776318.375 5576879.269, 17763...
```

```

len(gdf)
3320498
gdf.plot()
gdf.crs
<Projected CRS: EPSG:2193>
Name: NZGD2000 / New Zealand Transverse Mercator 2000
Axis Info [cartesian]:
- N[north]: Northing (metre)
- E[east]: Easting (metre)
Area of Use:
- name: New Zealand - North Island, South Island, Stewart Island - onshore.
- bounds: (166.37, -47.33, 178.63, -34.1)
Coordinate Operation:
- name: New Zealand Transverse Mercator 2000
- method: Transverse Mercator
Datum: New Zealand Geodetic Datum 2000
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
gdf.to_file('./data/nz-building-outlines.geojson', driver='GeoJSON')
gdf.to_file('./data/shp/nz-building-outlines.shp')
C:\Users\tek\AppData\Local\Temp\ipykernel_18796\1341848182.py:1: UserWarning: Column
names longer than 10 characters will be truncated when saved to ESRI Shapefile.
  gdf.to_file('./data/shp/nz-building-outlines.shp')
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/launched field name: 'building_id' to 'building_i'
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/launched field name: 'suburb_locality' to 'suburb_loc'
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/launched field name: 'territorial_authority' to 'territoria'
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/launched field name: 'capture_method' to 'capture_me'
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/launched field name: 'capture_source_group' to 'capture_so'
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/launched field name: 'capture_source_id' to 'capture__1'
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/launched field name: 'capture_source_name' to 'capture__2'
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/launched field name: 'capture_source_from' to 'capture__3'
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning: Field
capture__3 create as date field, though DateTime requested.
  ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:

```

```

Normalized/laundered field name: 'capture_source_to' to 'capture__4'
ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning: Field
capture__4 create as date field, though DateTime requested.
ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning:
Normalized/laundered field name: 'last_modified' to 'last_modif'
ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning: Field
last_modif create as date field, though DateTime requested.
ogr_write(
c:\Users\tek\conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning: 2GB file
size limit reached for data/shp\nz-building-outlines.dbf. Going on, but might cause compatibility
issues with third party software
ogr_write(
gdf = gpd.read_file('data/nz-building-outlines.geojson')
gdf.head()
building_id  name  use suburb_locality  town_city  territorial_authority  capture_method
capture_source_group  capture_source_id  capture_source_name  capture_source_from
capture_source_to  last_modified  geometry
0  2292028  Unknown  Akitio  Tararua District  Feature Extraction  NZ Aerial
Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27 2016-04-21
2019-01-04  MULTIPOLYGON (((1890386.169 5501282.718, 18903...
1  2292051  Unknown  Marton  Marton Rangitikei District  Feature Extraction  NZ
Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1796386.756 5560662.126, 17963...
2  2292290  Unknown  Rangataua  Ruapehu District  Feature Extraction  NZ Aerial
Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27 2016-04-21
2019-01-04  MULTIPOLYGON (((1811431.727 5632559.044, 18114...
3  2292353  Unknown  Durie Hill  Whanganui Whanganui District  Feature Extraction
NZ Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1776387.292 5576785.712, 17763...
4  2292407  Unknown  Durie Hill  Whanganui Whanganui District  Feature Extraction
NZ Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1776318.375 5576879.269, 17763...
gdf = gpd.read_file('data/shp/nz-building-outlines.shp')
gdf.head()
building_i  name  use suburb_loc  town_city  territoria  capture_me  capture_so
capture__1  capture__2  capture__3  capture__4  last_modif  geometry
0  2292028 None  Unknown  Akitio  None  Tararua District  Feature Extraction  NZ
Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  POLYGON ((1890386.169 5501282.718, 1890380.894...
1  2292051 None  Unknown  Marton  Marton Rangitikei District  Feature Extraction
NZ Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  POLYGON ((1796386.756 5560662.126, 1796394.053...
2  2292290 None  Unknown  Rangataua  None  Ruapehu District  Feature Extraction
NZ Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  POLYGON ((1811431.727 5632559.044, 1811437.183...
3  2292353 None  Unknown  Durie Hill  Whanganui Whanganui District  Feature
Extraction  NZ Aerial Imagery 1042  Manawatu Whanganui 0.3m Rural Aerial Photos (2...
2015-12-27 2016-04-21 2019-01-04  POLYGON ((1776387.292 5576785.712, 1776394.41 ...

```

```

4  2292407 None    Unknown    Durie Hill    Whanganui    Whanganui District    Feature
Extraction    NZ Aerial Imagery    1042    Manawatu Whanganui 0.3m Rural Aerial Photos (2...
2015-12-27    2016-04-21    2019-01-04    POLYGON ((1776318.375 5576879.269, 1776321.881...
gdf.crs
<Projected CRS: EPSG:2193>
Name: NZGD2000 / New Zealand Transverse Mercator 2000
Axis Info [cartesian]:
- N[north]: Northing (metre)
- E[east]: Easting (metre)
Area of Use:
- name: New Zealand - North Island, South Island, Stewart Island - onshore.
- bounds: (166.37, -47.33, 178.63, -34.1)
Coordinate Operation:
- name: New Zealand Transverse Mercator 2000
- method: Transverse Mercator
Datum: New Zealand Geodetic Datum 2000
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
gdf.to_parquet('./data/output/nz-building-outlines.parquet')
from datetime import datetime

print('-'*50)
start_p = datetime.now()
gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
end_p = datetime.now()
print('Parquet data read time: ', end_p - start_p)

start_s = datetime.now()
gdf = gpd.read_file('./data/shp/nz-building-outlines.shp')
end_s = datetime.now()
print('Shapefile data read time: ', end_s - start_s)

start_g = datetime.now()
gdf = gpd.read_file('./data/nz-building-outlines.geojson')
end_g = datetime.now()
print('GeoJSON data read time: ', end_g - start_g)

print('-'*50)
-----
Parquet data read time: 0:00:06.995695
Shapefile data read time: 0:00:52.704218
GeoJSON data read time: 0:04:34.113218
-----

Geospatial analysis
# Load the geoparquet file
gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
gdf.head()
building_id  name    use suburb_locality  town_city  territorial_authority  capture_method
capture_source_group  capture_source_id  capture_source_name  capture_source_from
capture_source_to  last_modified  geometry
0  2292028    Unknown    Akitio    Tararua District    Feature Extraction    NZ Aerial

```

```
Imagery 1042    Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27 2016-04-21
2019-01-04  MULTIPOLYGON (((1890386.169 5501282.718, 18903...
1  2292051    Unknown    Marton    Marton    Rangitikei District    Feature Extraction    NZ
Aerial Imagery 1042    Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1796386.756 5560662.126, 17963...
2  2292290    Unknown    Rangataua    Ruapehu District    Feature Extraction    NZ Aerial
Imagery 1042    Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27 2016-04-21
2019-01-04  MULTIPOLYGON (((1811431.727 5632559.044, 18114...
3  2292353    Unknown    Durie Hill    Whanganui    Whanganui District    Feature Extraction
NZ Aerial Imagery 1042    Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1776387.292 5576785.712, 17763...
4  2292407    Unknown    Durie Hill    Whanganui    Whanganui District    Feature Extraction
NZ Aerial Imagery 1042    Manawatu Whanganui 0.3m Rural Aerial Photos (2... 2015-12-27
2016-04-21 2019-01-04  MULTIPOLYGON (((1776318.375 5576879.269, 17763...
# filter the data based on attribute
gdf_filtered = gdf[gdf['town_city'] == 'Marton']

# buffer geometry by 10 meters
gdf_filtered['geometry'] = gdf_filtered.buffer(10)

# load only the first 10000 rows
gdf_filtered = gdf_filtered.head(10000)
c:\Users\tek\conda\envs\gdal\Lib\site-packages\geopandas\geodataframe.py:1819:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
    super().__setitem__(key, value)
visualization
# plot the data
gdf_filtered.plot()
<Axes: >

# save the data
gdf_filtered.to_file('./data/output/marton-buildings-buffered.geojson', driver='GeoJSON')
```



Let's walk through your code line by line, explain each part, and address any potential issues or errors along the way.

Setup

```
python

! pip install pyarrow
! pip install geopandas==1.0.0
```

- These commands install the necessary packages: ``pyarrow`` for working with Parquet files and ``geopandas`` for geospatial data manipulation.
- **Issue:** It's recommended to use the latest version of ``geopandas``. Version 1.0.0 is quite old, and upgrading to a newer version could avoid compatibility issues with more recent formats and methods.

Update:

```
python

! pip install geopandas --upgrade
```

Importing Libraries

```
python

import geopandas as gpd
gpd.__version__
```

- Imports the ``geopandas`` library, which is used to work with geospatial data.
- Checks the version of ``geopandas`` installed.

Reading GeoParquet File

```
python

gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
gdf.head()
```

- This loads the GeoParquet file into a GeoDataFrame (``gdf``).
- ``gdf.head()`` displays the first few rows of the GeoDataFrame to inspect the data.

No issues here.

Checking the Length of the GeoDataFrame

```
python

len(gdf)
```

- Displays the number of rows in the ``gdf``. It shows that there are 3,320,498 rows in this dataset.

Plotting the Data

```
python

gdf.plot()
```

- This generates a simple plot of the geospatial data.

Potential issue: If the dataset is too large, plotting might take a long time or lead to performance issues. You could filter or sample the data before plotting.

Suggestion:

```
python

gdf.sample(1000).plot() # Plot a sample of 1000 rows
```

Checking the Coordinate Reference System (CRS)

```
python

gdf.crs
```

- Prints the Coordinate Reference System of the geospatial data, which is `EPSG:2193` (NZGD2000 / New Zealand Transverse Mercator 2000).

No issues here.

Saving Data to GeoJSON and Shapefile Formats

```
python

gdf.to_file('./data/nz-building-outlines.geojson', driver='GeoJSON')
gdf.to_file('./data/shp/nz-building-outlines.shp')
```

- This saves the data into GeoJSON and Shapefile formats.

Warnings: The warnings indicate that column names longer than 10 characters will be truncated when saving to a Shapefile (due to Shapefile format limitations). You can either shorten the column names manually or ignore this if it's not a concern.

Update: To suppress these warnings, you can use the `warn_bad_lines` option:

```
python

gdf.to_file('./data/shp/nz-building-outlines.shp', driver='ESRI Shapefile', warn_bad_
```

```
lines=False)
```

Comparing Read Times for Different Formats

python

```
from datetime import datetime

start_p = datetime.now()
gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
end_p = datetime.now()
print('Parquet data read time: ', end_p - start_p)

start_s = datetime.now()
gdf = gpd.read_file('./data/shp/nz-building-outlines.shp')
end_s = datetime.now()
print('Shapefile data read time: ', end_s - start_s)

start_g = datetime.now()
gdf = gpd.read_file('./data/nz-building-outlines.geojson')
end_g = datetime.now()
print('GeoJSON data read time: ', end_g - start_g)
```

- This block compares the time it takes to read data in different formats: Parquet, Shapefile, and GeoJSON.
- The output shows that Parquet is the fastest to read, while GeoJSON is the slowest.

Geospatial Analysis

python

```
gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
gdf_filtered = gdf[gdf['town_city'] == 'Marton']
```

- Reads the Parquet file again and filters the dataset to include only rows where `town_city` is 'Marton'.

python

```
gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

- Buffers the geometry by 10 meters.

Warning: This raises a `SettingWithCopyWarning` because you are trying to modify a copy of a slice from a DataFrame. This warning occurs because directly assigning values to a slice may lead to unintended behavior.

Update: To avoid this, use `.loc[]` to ensure you're working on a copy of the data:


```
python
```

```
gdf_filtered = gdf[gdf['town_city'] == 'Marton'].copy()
gdf_filtered.loc[:, 'geometry'] = gdf_filtered.buffer(10)
```

Visualization

```
python
```

```
gdf_filtered.plot()
```

- This plots the buffered geometries of the filtered dataset.

```
python
```

```
gdf_filtered.to_file('./data/output/marton-buildings-buffered.geojson', driver='GeoJSON')
```

- This saves the filtered and buffered data to a new GeoJSON file.

No issues here.

Conclusion

- The primary issues encountered were with outdated versions of `geopandas` and a `SettingWithCopyWarning`. By upgrading `geopandas` and using `.copy()` with `.loc[]`, these issues are resolved.

Let me know if you need further modifications or additional analysis!



missed some codes to explain



Let me go over the full code again and ensure that I cover each line and segment properly. I'll explain everything in more detail, including any issues or warnings.

1. Installing necessary libraries

```
python
```

```
! pip install pyarrow
! pip install geopandas==1.0.0
```

- These lines install necessary libraries for the code to run:
 - ``pyarrow`` is required to read and write Parquet files.
 - ``geopandas==1.0.0`` installs an older version of GeoPandas, which may lead to compatibility issues with modern datasets or code syntax. It's recommended to update GeoPandas to a newer version:

Update:

```
python

! pip install geopandas --upgrade
```

2. Importing and checking the version of GeoPandas

```
python

import geopandas as gpd
gpd.__version__
```

- Importing ``geopandas`` as ``gpd`` to work with geospatial data. This is essential for reading and writing geospatial data formats.
- ``gpd.__version__`` checks the current version of the installed ``geopandas``. In this case, it is '1.0.0', but you should consider upgrading to avoid any compatibility issues.

3. Reading the Parquet file

```
python

gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
gdf.head()
```

- This reads the Parquet file (``nz-building-outlines.parquet``) into a GeoDataFrame (``gdf``).
- ``gdf.head()`` displays the first 5 rows of the data, which allows you to inspect the structure of the dataset.

No issues here.

4. Checking the number of rows in the dataset

```
python

len(gdf)
```

- This line checks how many rows are in the dataset. The dataset contains **3,320,498 rows**.

5. Plotting the dataset

```
python

gdf.plot()
```

- This command generates a plot of the geometries in the dataset.

Potential issue: If the dataset is very large, plotting all rows might be slow. You can sample the data to reduce the number of rows plotted.

Suggested update:

```
python

gdf.sample(1000).plot() # Plot a random sample of 1000 rows
```

6. Checking the Coordinate Reference System (CRS)

```
python

gdf.crs
```

- This line prints the CRS (Coordinate Reference System) of the dataset. In this case, it's `EPSG:2193`, which is the New Zealand Transverse Mercator 2000 projection.

No issues here.

7. Saving the data to GeoJSON and Shapefile formats

```
python

gdf.to_file('./data/nz-building-outlines.geojson', driver='GeoJSON')
gdf.to_file('./data/shp/nz-building-outlines.shp')
```

- These lines save the dataset in GeoJSON and Shapefile formats.

Warnings: You might see warnings about column names being truncated in Shapefiles (since Shapefiles have a 10-character limit for field names). GeoJSON does not have this issue.

Update to suppress warnings:

```
python

gdf.to_file('./data/shp/nz-building-outlines.shp', driver='ESRI Shapefile', warn_bad_lines=False)
```

8. Reading the GeoJSON and Shapefile formats back into GeoDataFrames

python

```
gdf = gpd.read_file('data/nz-building-outlines.geojson')
gdf.head()
```

- This reads the previously saved GeoJSON file and displays the first 5 rows. You can confirm that the data is correctly written and read back.

python

```
gdf = gpd.read_file('data/shp/nz-building-outlines.shp')
gdf.head()
```

- This reads the Shapefile and similarly displays the first 5 rows.

No issues here.

9. Checking the CRS of the reloaded data

python

```
gdf.crs
```

- Once again, it prints the CRS of the data after reading from the file, ensuring it has not changed during the file writing and reading processes.

10. Saving the data back to Parquet

python

```
gdf.to_parquet('./data/output/nz-building-outlines.parquet')
```

- This saves the GeoDataFrame back to a Parquet file.

No issues here.

11. Comparing read times for different formats (Parquet, Shapefile, GeoJSON)

python

```
from datetime import datetime
print('- '*50)
start_p = datetime.now()
```

```
gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
end_p = datetime.now()
print('Parquet data read time: ', end_p - start_p)

start_s = datetime.now()
gdf = gpd.read_file('./data/shp/nz-building-outlines.shp')
end_s = datetime.now()
print('Shapefile data read time: ', end_s - start_s)

start_g = datetime.now()
gdf = gpd.read_file('./data/nz-building-outlines.geojson')
end_g = datetime.now()
print('GeoJSON data read time: ', end_g - start_g)

print('- '*50)
```

- This block measures how long it takes to read data from different file formats (Parquet, Shapefile, GeoJSON).
- Parquet is generally the fastest to read, followed by Shapefile, and then GeoJSON, which is the slowest.

No issues here.

12. Geospatial analysis (Filtering and Buffering)

python

```
gdf_filtered = gdf[gdf['town_city'] == 'Marton']
```

- This filters the dataset for buildings located in 'Marton'. The resulting GeoDataFrame (`gdf_filtered`) will contain only those rows where the `town_city` is 'Marton'.

python

```
gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

- Buffers the geometries in `gdf_filtered` by 10 meters.

Warning: This line produces a `SettingWithCopyWarning`, indicating that you're modifying a slice of the original data. This could cause issues, so it's better to explicitly create a copy of the data before modifying it.

Update:

python

```
gdf_filtered = gdf[gdf['town_city'] == 'Marton'].copy()
gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

13. Loading only the first 10,000 rows

```
python

gdf_filtered = gdf_filtered.head(10000)
```

- This limits the dataset to the first 10,000 rows. It's useful if you're working with a very large dataset and want to reduce memory usage.

14. Visualization

```
python

gdf_filtered.plot()
```

- This line generates a plot of the filtered and buffered geometries.

15. Saving the filtered and buffered data

```
python

gdf_filtered.to_file('./data/output/marton-buildings-buffered.geojson', driver='GeoJSON')
```

- This saves the filtered and buffered data to a GeoJSON file.

Conclusion

In this walk-through, the primary issues were related to:

- **Outdated versions** of GeoPandas (upgrading to the latest version can avoid issues).
- `SettingWithCopyWarning` when modifying the filtered data slice. This can be resolved by creating a copy before making modifications.

By upgrading the GeoPandas library and ensuring that proper data copies are made, the code should execute without warnings or errors. Let me know if you'd like further clarifications or additional details!



explain the concept



The code you've shared is performing **basic geospatial analysis** using a **GeoParquet file format**. Let me break down the concepts that are central to understanding this code:

1. GeoParquet and Geospatial Data

- **GeoParquet** is a file format that combines the efficient columnar storage of **Parquet** with geospatial support (geometry data). It is used for storing large amounts of geospatial data in an optimized way, enabling fast reading and writing.
- **Geospatial data** refers to data that includes information about the geographical location of objects, typically represented as coordinates (latitude/longitude or projected coordinates). Examples include polygons for building footprints, lines for roads, or points for specific locations.

2. GeoPandas

- **GeoPandas** is an extension of the popular Python data analysis library **pandas**, but with support for geospatial data. GeoPandas makes it easy to work with geometries (like points, lines, and polygons) and conduct spatial operations such as intersections, buffering, and filtering based on location.
- A **GeoDataFrame** in GeoPandas is like a regular pandas DataFrame, but with an additional column containing geometrical shapes.

3. CRS (Coordinate Reference System)

- **CRS** defines how the two-dimensional, projected map in the dataset relates to real locations on Earth. Different projections like **EPSG:2193** (New Zealand Transverse Mercator) are used for specific regions to convert latitude and longitude into a flat coordinate system for easier computations.
- In this example, the dataset uses **NZGD2000 (New Zealand Geodetic Datum 2000)** for mapping geospatial data onto New Zealand's territory.

4. Buffering in Geospatial Analysis

- **Buffering** creates a new geometry that represents all the points within a certain distance (buffer) from the original geometry. For example, buffering a polygon by 10 meters will create a new polygon that expands outward by 10 meters.

In the code:

```
python

gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

This line creates a 10-meter buffer around all building geometries in the `gdf_filtered` dataset. It's useful for spatial analysis, for example, determining which buildings are within a certain distance of a road or a river.

5. Filtering Geospatial Data

- The code filters the dataset based on an attribute using this line:

```
python
```

```
gdf_filtered = gdf[gdf['town_city'] == 'Marton']
```

This retrieves only the rows (or features) where the buildings are located in **Marton** (a town in New Zealand). Filtering by attributes like city, district, or other metadata is common in geospatial analysis to focus on specific areas of interest.

6. File Formats in Geospatial Analysis

- **Parquet:** A columnar storage file format, well-suited for large datasets. It offers fast read/write performance and is ideal for data stored in tables like in a DataFrame.
- **GeoJSON:** A geospatial data format that is easy to work with, especially for web mapping applications, but can be slower than Parquet due to its text-based nature.
- **Shapefile:** An older geospatial format with limitations like 10-character field name limits and a maximum size of 2 GB. It's still widely used but considered less efficient for larger datasets compared to Parquet or GeoJSON.

7. Plotting Geospatial Data

- **Plotting** is a way to visualize geospatial data. The `.plot()` method in GeoPandas is a simple way to quickly visualize the geometries in your GeoDataFrame. For example:

```
python
```

```
gdf.plot()
```

This will display the spatial footprints of buildings or other geometrical shapes on a 2D plane.

8. Performance Considerations in Geospatial Analysis

- The code compares the **read times** of different file formats: Parquet, Shapefile, and GeoJSON:
 - **Parquet** is the fastest to read, making it efficient for large datasets.
 - **Shapefile** is slower due to its older format and more complex structure.
 - **GeoJSON** is the slowest because it is a text-based format, meaning the data must be parsed and interpreted as text, which is inefficient for large datasets.

9. Warnings about Column Length in Shapefiles

- The warnings about truncating column names to 10 characters occur because **Shapefiles** have a strict limit on the number of characters in field names (max 10). This is a limitation of the Shapefile format, and to resolve it, column names can be manually shortened or handled in other formats like GeoJSON, which does not have this restriction.

10. SettingWithCopyWarning

- The warning `SettingWithCopyWarning` in pandas (or GeoPandas) happens when you try to modify a copy of a slice of a DataFrame. It's important to avoid this warning by explicitly copying the data before making changes to it. In this case, the code:


```
python
```

```
gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

should be changed to:

```
python
```

```
gdf_filtered = gdf[gdf['town_city'] == 'Marton'].copy()
gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

This ensures that you're working on a copy of the data, not a view, which avoids unexpected side effects.

Summary of the Concepts:

- **GeoParquet** is a modern, efficient way to store geospatial data, combining the benefits of Parquet and geospatial features.
- **GeoPandas** makes it easy to handle geospatial data in Python and allows operations like filtering, buffering, and plotting.
- **CRS** ensures that the geospatial data is correctly projected on the Earth's surface.
- **Buffering** is used to create zones around geometries, and **filtering** extracts specific data based on attributes.
- Different file formats (Parquet, GeoJSON, Shapefile) come with their own performance trade-offs, and choosing the right format is important for efficient data handling.

Let me know if you need further clarification on any part!



somecodes missing



Apologies for missing some parts of the code in my previous explanation. I'll now go through all the missing parts of the code and make sure every line is explained thoroughly.

Full Explanation of the Missing Code:

1. Saving Data to GeoParquet Format

```
python
```

```
gdf.to_parquet('./data/output/nz-building-outlines.parquet')
```

- This line writes the **GeoDataFrame** (``gdf``) back to a **GeoParquet file** (``nz-building-outlines.parquet``). This file format is efficient in terms of storage size and I/O performance compared to other formats such as Shapefile or GeoJSON.

No issues here.

2. Comparing Read Times for Different File Formats (Parquet, Shapefile, GeoJSON)

python

```
from datetime import datetime

start_p = datetime.now()
gdf = gpd.read_parquet('./data/nz-building-outlines.parquet')
end_p = datetime.now()
print('Parquet data read time: ', end_p - start_p)
```

- **Start:** Records the current time before reading the Parquet file.
- **Read Parquet file:** The ``read_parquet()`` function reads the GeoParquet file and loads it into the GeoDataFrame ``gdf``.
- **End:** Records the current time after the Parquet file has been read.
- **Print time:** The time difference (``end_p - start_p``) shows how long it took to read the Parquet file.

python

```
start_s = datetime.now()
gdf = gpd.read_file('./data/shp/nz-building-outlines.shp')
end_s = datetime.now()
print('Shapefile data read time: ', end_s - start_s)
```

- Similar to the above block, but this part reads the **Shapefile** using the ``read_file()`` method. Shapefiles are slower to read compared to Parquet due to format limitations.
- The code calculates the time taken to read the Shapefile and prints it.

python

```
start_g = datetime.now()
gdf = gpd.read_file('./data/nz-building-outlines.geojson')
end_g = datetime.now()
print('GeoJSON data read time: ', end_g - start_g)
```

- This part reads the **GeoJSON file** and calculates the time taken for this operation. Reading a GeoJSON file is typically slower than Parquet and Shapefile due to its text-based nature, which requires more processing.

Finally, this code block prints out the read times for **Parquet**, **Shapefile**, and **GeoJSON** formats. This comparison illustrates that **Parquet** is the fastest format for reading large datasets.

3. Filtering Data Based on an Attribute

```
python
```

```
gdf_filtered = gdf[gdf['town_city'] == 'Marton']
```

- **Filtering:** This line filters the GeoDataFrame `gdf` to include only rows where the `town_city` column is equal to `Marton`.
- This is useful for focusing on a subset of the data (in this case, only the buildings in the town of Marton).

4. Buffering the Geometry

```
python
```

```
gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

- **Buffering:** Adds a **10-meter buffer** around each geometry (which could be polygons representing buildings, etc.). The new geometries will be polygons that are expanded outward by 10 meters.
- **Warning:** The line will trigger a `SettingWithCopyWarning` because you're attempting to modify a slice of the original GeoDataFrame. To resolve this, make an explicit copy before modifying it:

Updated version:

```
python
```

```
gdf_filtered = gdf[gdf['town_city'] == 'Marton'].copy()
gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

5. Limiting the Data to the First 10,000 Rows

```
python
```

```
gdf_filtered = gdf_filtered.head(10000)
```

- This line limits the filtered and buffered GeoDataFrame (`gdf_filtered`) to the **first 10,000 rows**.
- This is useful for working with a smaller subset of data, especially if the dataset is too large to process efficiently in full.

6. Plotting the Filtered Data

```
python
```

```
gdf_filtered.plot()
```

- This line plots the geometries of the filtered and buffered dataset. It provides a visual representation of the buildings (or other geospatial objects) located in **Marton**, buffered by 10 meters.

Note: Depending on the system resources, plotting a large dataset can be slow. You might want to sample or reduce the dataset size if performance is an issue.

7. Saving the Filtered and Buffered Data as GeoJSON

python

```
gdf_filtered.to_file('./data/output/marton-buildings-buffered.geojson', driver='GeoJSON')
```

- This line saves the filtered and buffered data (buildings in Marton with 10-meter buffers) to a new **GeoJSON** file.
- The `driver='GeoJSON'` parameter specifies that the output format should be GeoJSON.

8. Warnings Regarding Shapefile Field Name Truncation

In earlier code, while saving data to a **Shapefile**, you encountered warnings like:

python

```
c:\Users\tek\.conda\envs\gdal\Lib\site-packages\pyogrio\raw.py:709: RuntimeWarning: Normalized/launched field name: 'building_id' to 'building_i'
  ogr_write(
```

These warnings are due to the **10-character limit** for field names in the Shapefile format. Since column names like `'building_id'` are longer than 10 characters, they are truncated when saved to a Shapefile.

You can ignore these warnings or shorten the column names manually before saving the file. Alternatively, consider using file formats like GeoJSON or Parquet that do not have this limitation.

Summary of the Full Code:

1. **Installation and Import:** Install necessary libraries and import `geopandas` for geospatial data handling.
2. **Reading Data:** The code reads geospatial data from different formats: Parquet, Shapefile, and GeoJSON.
3. **Filtering:** Data is filtered based on the `town_city` attribute to focus on buildings in **Marton**.
4. **Buffering:** The geometries are buffered by 10 meters, which is useful in many geospatial analyses like proximity analysis.
5. **Saving:** Data is saved to multiple formats: Parquet, GeoJSON, and Shapefile. The Shapefile format triggers warnings due to field name length limitations.
6. **Performance Comparison:** The code compares the read times of Parquet, Shapefile, and GeoJSON formats, demonstrating that Parquet is the most efficient for reading large geospatial datasets.

7. **Visualization:** Filtered and buffered data is plotted for visualization purposes.

This explanation now covers all the missing parts and gives you a complete understanding of the geospatial analysis using the GeoParquet file format.

Let me know if you need further clarification on any specific part of the code or concepts!



explain the concept



The overall **concept** behind the code you provided revolves around **geospatial data analysis**, specifically using **GeoParquet** and other geospatial file formats such as **Shapefile** and **GeoJSON**. Let's break down the key concepts involved:

1. Geospatial Data

Geospatial data refers to information that has an associated location. In this case, we are working with building outlines in New Zealand, which are represented as polygons. Each polygon has coordinates (latitude, longitude, or projected coordinates) that define its location and shape on the Earth's surface.

- **Geometries:** These are the spatial shapes in your data, like points, lines, or polygons. For example, a polygon can represent the outline of a building or a land plot.
- **Attributes:** Besides geometrical information, each feature (e.g., a building) might have additional properties such as ``building_id``, ``town_city``, and ``capture_method``, which describe non-spatial attributes.

2. GeoParquet

GeoParquet is a modern, efficient file format that allows for the storage of geospatial data in **Parquet format**. The Parquet format is highly efficient because it:

- Stores data in a **columnar** manner, which makes reading and writing large datasets faster.
- Is highly **compressed**, saving space when working with large geospatial datasets.
- Supports parallel processing, making it ideal for big data applications.
- Is flexible enough to store not only traditional tabular data (like in a DataFrame) but also geospatial features like **polygons** and **multipolygons**.

By using GeoParquet, you can handle large geospatial datasets quickly and efficiently compared to older formats like Shapefile or GeoJSON.

3. GeoPandas

GeoPandas is a Python library that extends **pandas** (used for handling tabular data) to support geospatial data. It adds functionality to:

- Work with geometries (points, lines, polygons).
- Perform spatial operations such as **buffering**, **intersection**, **spatial joins**, and more.

- Plot geospatial data on a map.
- Read and write data in multiple geospatial formats like **Shapefile**, **GeoJSON**, and **GeoParquet**.

GeoDataFrame: In GeoPandas, a **GeoDataFrame** is an extension of the pandas DataFrame that can hold geometry data. Each row represents a geospatial feature, and one column contains the **geometry** (polygon, multipolygon, etc.) defining the spatial location of the feature.

4. CRS (Coordinate Reference System)

A **Coordinate Reference System (CRS)** defines how the two-dimensional, projected map (like the polygons representing building outlines) relates to real-world locations on the Earth's surface. There are different types of projections and coordinate systems, and choosing the right one is critical to ensure accuracy.

- In the provided code, the CRS is ``EPSG:2193``, which is **New Zealand Transverse Mercator (NZGD2000)**, a projection that's commonly used in New Zealand.
- **CRS Conversion:** Different geospatial datasets might use different projections, and converting between them is an important part of geospatial analysis.

5. Spatial Operations: Filtering and Buffering

Filtering and **buffering** are fundamental spatial operations in geospatial analysis:

- **Filtering:** This operation selects features (rows in a GeoDataFrame) based on some attribute. In your code:

```
python

gdf_filtered = gdf[gdf['town_city'] == 'Marton']
```

This selects all buildings located in the town **Marton**.

- **Buffering:** Buffering creates a new geometry that is expanded outward from the original geometry by a specified distance. This is useful for proximity analysis, such as determining how close buildings are to certain features (roads, rivers, etc.). In your code:

```
python

gdf_filtered['geometry'] = gdf_filtered.buffer(10)
```

This creates a 10-meter buffer around each building, meaning that a new polygon is generated that is 10 meters larger than the original building outline.

6. File Formats for Geospatial Data

The code works with multiple geospatial file formats. Each has its own strengths and weaknesses:

- **GeoParquet:** Efficient for reading and writing large datasets and performing analytics on geospatial data.
- **Shapefile:** A traditional format that is widely used but has limitations (e.g., it truncates column names to 10 characters, can only handle files up to 2GB, and lacks built-in compression).
- **GeoJSON:** A text-based format that is easy to work with for web applications but slower and more space-consuming compared to binary formats like Parquet.

7. Performance Comparison

The code compares the performance of reading data from different file formats:

- **Parquet:** Fastest due to its optimized columnar structure.
- **Shapefile:** Slower than Parquet due to its older, more rigid structure.
- **GeoJSON:** The slowest to read, as it's a text-based format, meaning each feature must be parsed from text.

This performance comparison demonstrates that using modern formats like **Parquet** can significantly speed up geospatial data analysis tasks, especially when working with large datasets.

8. Handling Large Geospatial Data

The code deals with a dataset of over 3 million building outlines. When working with such large datasets, considerations like:

- **Storage efficiency:** Choosing file formats like Parquet to save storage space.
- **Read and write performance:** Using efficient formats (e.g., Parquet) to reduce processing time.
- **Sampling data:** Plotting or analyzing subsets of data (as done in the code) when the full dataset is too large to handle.

9. Visualization

- **gdf.plot()** generates simple maps showing the geospatial data.
- Visualization is crucial in geospatial analysis to visually inspect the data and results of spatial operations, such as the buffered buildings in the town of Marton.

10. Warnings and Field Name Limitations in Shapefiles

Shapefiles, being an older format, impose certain restrictions:

- **Field name truncation:** Shapefiles limit field names to 10 characters. When saving a GeoDataFrame to a Shapefile, long column names like ``capture_method`` get truncated (e.g., to ``capture_me``).
- **2GB file size limit:** Shapefiles have a maximum file size of 2GB, making them inefficient for very large datasets.

These limitations are important to understand when choosing file formats for geospatial analysis.

Summary of Concepts

- **GeoParquet:** A highly efficient file format for storing large geospatial datasets.
- **GeoPandas:** A powerful Python library for handling and analyzing geospatial data.

- **CRS:** A system that relates geospatial data to locations on Earth, ensuring correct interpretation of coordinates.
- **Filtering:** Selecting specific features from the dataset based on attributes.
- **Buffering:** Expanding geometries by a given distance to create buffer zones.
- **File Formats:** The strengths and limitations of formats like Parquet, Shapefile, and GeoJSON.
- **Performance:** The speed of reading different formats, highlighting Parquet's advantages in handling large datasets.
- **Visualization:** Plotting geospatial data for easy inspection and interpretation.

These concepts collectively form the foundation of geospatial analysis, enabling efficient handling of spatial data, spatial queries, and visualizing geospatial relationships.

ChatGPT can make mistakes. Check important info.