Open GIS for the Developing World

Term Project and Lab Assignments

Authored by Dr. Joseph Holler at Middlebury College, 2015.
Contact: joseph@josephholler.com

This work was made possible in part by the GIS labs and support staff of the Middlebury College Department of Geography, the Middlebury College Faculty Research Assistant Fund, the GIS Teaching Fellows Program, and the research assistance of Hayden Shea.

Reference:
Holler, J. (2015). *Open GIS for the Developing World: Term Project and Lab Assignments*. Retrieved from http://www.josephholler.com

Table of Contents

Lab 1: Thematic Mapping in QGIS

Purpose:

The goal for the first week is to create two global thematic maps of the digital divide. In doing so, you will become acquainted with the QGIS software environment, create your own SpatiaLite database, download data, and prepare it and import it to your database, and use your database and QGIS to join data and create maps.

Data:

- Natural Earth: http://www.naturalearthdata.com/
- World Bank: http://data.worldbank.org/indicator

References:

- QGIS 2.6 documentation: http://www.qgis.org/en/docs/index.html#26
- SpatiaLite: http://www.gaia-gis.it/gaia-sins/ and the QSpatiaLite plugin: https://code.google.com/p/qspatialite/
- GDAL: http://www.gdal.org/ and its CSV importer: http://www.gdal.org/drv_csv.html
- SQLite: http://www.sqlite.org/ and its data types: https://www.sqlite.org/datatype3.html
- How to use Natural Earth SpatiaLite database with QGIS: http://gis.stackexchange.com/questions/78172/how-do-i-use-the-natural-earth-sqlite-db-with-qgis

Deliverables:

- Two global thematic maps of variables indicative of the digital divide – one of the percentage of internet users and one of your choice.

Procedure:

`Copy lab data and launch QGIS`	• Create a folder for this lab in your personal splinter folder. • Start QGIS 2.6 Brighton Desktop. • Save the QGIS project to your folder for this lab. • Save frequently, e.g. after each set of instructions! • You might want to minimize distractions in the GUI… if so, go to *View* -> *Panels* and select only Layers and Browser. Layers is similar to the Table of Contents in ArcGIS, and Browser is similar to ArcCatalog. • Similarly go to *View* -> *Toolbars* and select only attributes, file, help, plugins, and map navigation.
`Set up a new SpatiaLite database for storing geographic data`	• SpatiaLite is an extension of SQLite (a simple relational database) which is capable of storing and analyzing spatial data, much like a geodatabase with vector data. • In the **Browser** panel, right-click the **SpatiaLite** SpatiaLite tree and **create database**. Save the new database to your lab folder. • The database should appear as a new connection under SpatiaLite. • Go to **Database** -> **DBManager** -> **DBManager** and verify that your database is listed under SpatiaLite. You'll see that it already includes over 20 systems tables to manage the database.

Load geographic data into a QGIS project from a database

- **Note:** There are several different ways to add data to a QGIS project, and they are not all equally useful. If data fails to import with one method, try another… In preparing this lab I've noticed that first importing data to QGIS as layers has been more reliable than other methods.
- The geographic data for this lab is provided by the Natural Earth project, and open source one stop shop for basic global data. It can be downloaded as individual shapefiles, as a geodatabase, or as an SQLite database. I've chosen the open source version: SQLite, and saved it in our courses folder on Splinter. You may copy it, download it, or simply use it where it is.
- Go to **Layer** -> **Add Layer** -> **Add Vector Layer**
    - Keep the **source type** as `file` and set the **dataset** to the `Natural_Earth_Vector` database. Wait a moment…
- A **Select Layers to Add** dialogue pops up. Hold the **Ctrl** key and select the following layers to add:
    - `ne_110m_admin_0_countries_lakes`
    - `ne_110m_lakes`
    - `ne_110m_ocean`
- Once added, the layer geometries should display in QGIS.
- Re-open **DBManager** and go to the **Import Layer/file** tool 
- Re-connect to your database by expanding its tree and viewing a table.
- One at a time, select an **input** layer using the drop-down menu. For each, change the **Table** name under **Output Table** to something shorter and easier, e.g. `countries110`, `lakes110`, and `ocean110`.
    - ***Note*:** Table names and Field names in databases should always start with a letter and contain only letters and numbers. They should not be identical to any SQL keywords: www.sqlite.org/lang_keywords.html or SpatiaLite function names: http://www.gaia-gis.it/gaia-sins/spatialite-sql-4.2.0.html
    - `ne_110m_admin_0_countries_lakes` did not import correctly! This version of QGIS is not handling a layer with both polygons and multipolygons correctly. You can fix this: right-click the countries layer and save it as an ESRI Shapefile: `countries.shp`. Then import that new layer with the new table name, `countries`.
    - You may delete the broken `countries110` table by right-clicking it.
- Changes often do not appear automatically, so make frequent use of the **refresh** tool 
- Each of the layers you added should appear with a geometry icon (e.g. polygons  or points ), and should have the preview tab enabled.
- Nice! You've just set up your first SpatiaLite database!
- In the QGIS **Layers** panel, remove all the layers that you added from `Natural_Earth_Vector`. Then add the layers from your own SpatiaLite database: expand the database in the Browser panel, right clicking table names, and **Add Layer**.

Download World Bank data and prepare it for use with GIS

- Go to the World Bank Data indicators site: http://data.worldbank.org/indicator
- Find the `Internet Users (per 100 people)` indicator
- **Download Data** as a **CSV** file (comma-separated values).
- Extract or copy the data to your lab folder.
- Open the `it.net.user.p2_Indicator_en_csv_v2.csv` file in Microsoft Excel.
- You can do this by dragging the file onto an Excel window, or by using Excel's **From Text** import tool on the **Data** tab. If you use the from text tool, indicate that the file is **delimited** with **commas**.
- Delete the first two rows before the column names.
- We only need the `Country Name`, `Country Code`, and `2013` data columns. Delete the other columns.
  - **Tip**: don't just erase what's in a row or column. Delete the entire row or column by selecting the row number or column letter, right-clicking, and delete. The column headings should not be in row 1.
- Edit column headers to conform to good database field names.
  - "Country Name" may be `CountryName`
  - "Country Code" may be `CountryCode`
  - "2013" may be `InternetUsers`
- Save the spreadsheet as `internet.csv`.

Load tabular attribute data into QGIS and a SpatiaLite database

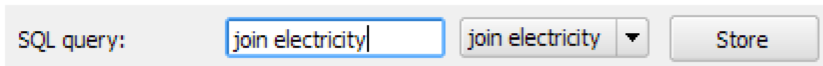
- Return to **QGIS** and go to **Layer** -> **Add Layer** -> **Add Delimited Text Layer**
  - Set the **File Name** by browsing to `internet.csv`
  - Set the **File Format** to *CSV (comma separated values)*
  - For **Record Options**, make sure you've checked **First Record has field names**, and do not discard any header lines.
  - Set the **Geometry definition** to *no geometry (attribute table only)*
  - If the preview of rows looks good, **OK**!
- Look at the `internet` layer properties (right click `internet` -> **properties**) and check the **Fields** to make sure that the `InternetUsers` field has imported as a numerical data type: **double**.
- Now you're ready to add the table to your database! Re-open **DBManager** and go to the **Import Layer/file** tool 
- Select the `internet` layer using the drop-down menu and import it to a table with the same name, `internet`.
- Refresh the database and look for your new table. Does it look ok?
- Also check the table's **info** to make sure the data types are correct:
  - `pk` is **INT**
  - `CountryName` is **TEXT**
  - `CountryCode` is **TEXT**
  - `InternetUsers` is **REAL**

Join Attribute Data to Geographic Features

- When joining tables, you need one or more join fields that can accurately match the rows to be joined. In this case, you can use the `admin0_a3_is` field in the `countries` table and `CountryCode` in the `internet` table.
- A join can be accomplished with an SQL query. Open the **SQL Window**  and enter the following query:

```
SELECT *
FROM countries LEFT OUTER JOIN internet
ON adm0_a3_is = CountryCode
```

- If you want to save and re-use queries, enter a name for **SQL Query** and **store**:



- Now **Execute**! The result is a temporary table with `internet` data appended to the end of the `countries` table, using `admin0_a3_is` and `CountryCode` as the join fields. You'll notice that any country with no matching row in the internet data has **NULL** values for the `CountryName`, `CountryCode`, and `InternetUsers` fields. Any country that had a blank cell in the World Bank table also has a **NULL** value for `InternetUsers`.
- Let's see the results on the map!
- Check **load as new layer**.
  - The **column with unique integer values** is asking for a *primary key*. In this case it is: `PKUID`.
  - The **geometry column** is asking for the column storing feature geometries. In this case it is: `Geometry`.
  - Enter a **layer name,** e.g. `Internet Users`
  - Ok, **Load Now**!
- A new `QueryLayer` should appear on the map. If you open its attribute table, it should be complete with the internet use data.

Create a Map of Internet Users per 100 People

- Go to the layer **properties** and **style**, and use the **Graduated** option to classify the data with up to 7 classes.
  - Set the **column** to `InternetUsers`
  - Try a variety of classification **modes** and use **classify** and **apply** to see their influence on the map.
- To help your decision, here's a summary of classification methods in QGIS:
  - **Equal Interval** divides the range from the *minimum* to the *maximum* into equal parts. For example, if your *min* is 0 and your *max* is 50 and you choose 2 classes, the first will be 0-25 and the second will be 25-50.
  - **Pretty Breaks** this classification is like an equal interval classification, modified to round to more legible numbers.
  - **Quantile (Equal Count)** ranks all of your records from smallest to greatest and divides the dataset into the number of groups you specify, with equal number of records in each group. E.g. Quantile

classification of 60 records into 3 classes will place the smallest 20 records in one class, next 20 records in a second class, and largest 20 records in a final class.

- o **Natural Breaks (Jenks)** ranks the data and breaks it into the number of classes you specify. Then, it iteratively moves one record from the class with the most variability toward the class with the least variability. It continues to bump records from high variability classes toward low variability classes until the procedure no longer improves the classification. The result is that classes form around natural clusters in the data, with natural "breaks" in between classes. It was George Jenks' idea.
- o **Standard Deviation** calculates the mean (average) and the standard deviation (a measure of variability) of the dataset. Starting at the mean, it creates classes with range sizes based on the standard deviation, adding classes below and above the mean until the whole dataset is covered. Thus, this classification groups data in relation to its difference from the mean.

`Represent the absence of data`

- You'll notice that any geometry with NULL internet values has disappeared! Let's add a second layer to represent those countries and territories with missing internet user data.
- Back in **DBManger**'s **SQL Window**, modify your query to select all the records with no internet data by adding a WHERE clause:

```
SELECT *
FROM countries LEFT OUTER JOIN internet
ON adm0_a3_is = CountryCode
WHERE InternetUsers IS NULL
```

- With this new layer, represent countries with no data available with a symbol clearly distinct from the colors you chose for the internet data.

`Compose a Map Layout and save it as a pdf`

- Thematic maps of data by enumeration units (e.g. countries) should usually use a map projection that preserves equal areas. Change the coordinate system of the map by going to **Project** -> **Project Properties**. Enable "on the fly" CRS transformation and filter the CRS list by the keyword "world". The Mollweide, Robinson, or World Sinusodial projections are good options for global thematic maps. It may look ridiculous until you zoom/pan to an area where no polygons are wrapping around the edges of the projection, but it seems to render just fine when you export a map as an image or pdf.
- Go to **File** -> **new print composer**
- Hereafter, go to **File** -> **Composer Manager** to access your composition
- Make an ANSI A letter-size landscape orientation layout.
- See QGIS Documentation for the Print Composer to familiarize yourself with the QGIS icons and tools. In particular:
  - o  allows you to add a new map frame
  - o  allows you to move data within the map frame

-  allows you to select, resize, and move items
- It seems that the easiest way to adjust scale is by selecting the map and editing the scale ratio's denominator in the **scale** field of its **item properties**. For a global map on letter size paper, try something around `130,000,000`

- The map should include the map data, title, and credits (your name, date, the data sources, the projection)
- Scale bar and north arrow are inappropriate, because in keeping areas equal, distance and north direction are inconsistent. You may add a scale ratio.
- Save the composition, and export your map as a pdf.

A challenge: Prepare your own map from a second World Bank indicator

- You should now be able to produce a second map of another World Bank indicator. Choose a second indicator that helps supplement internet users as an indicator of access to digital technology in developing countries. The indicator should be normalized (e.g. it is already a *rate*, *percentage*, *ratio*, or *density*).
- I suggest using the **duplicate** tool in the **composer manager** to save yourself some time—you can then simply adjust the data, title and legend.
- When you're finished, upload your two maps to the assignment for Lab 1 on Moodle.

PreLab 2: Structured Query Language for troubleshooting Joins

Purpose:

Some of you noticed imperfections in the completeness and cardinality of the joins we used in lab 1. That is, sometimes joins were **incomplete**, with some rows from the target table not matching any rows in the join table, or some rows in the join table not matching any rows in the target table. Other times joins had **one to many cardinality** problems, with duplicate records matching in the joining tables. This PreLab will follow up on those questions with the same two datasets: World Bank internet use data, and Natural Earth 110m countries.

These types of problems can be extremely frustrating to diagnose for large datasets. Fortunately, SQL queries can be of great help, and this pre-lab will expand your knowledge of SQL selection queries while demonstrating some strategies for diagnosing problematic joins.

Data:

- Natural Earth: http://www.naturalearthdata.com/
- World Bank: http://data.worldbank.org/indicator

References:

- SQLite: http://www.sqlite.org/
- Definitive Guide to SQLite: http://link.springer.com/978-1-4302-3226-1

Deliverables:

- Complete the quiz on Moodle by 8am. The quiz questions follow the procedure sequentially.

Procedure:

`Open Lab1 in QGIS`	• Open your lab 1 QGIS project.
	• Open **DB Manager** and re-connect to the database you created in lab 1
	• Verify that you have a SQLite database containing:
	  o a table of `countries` from Natural Earth containing 177 rows
	  o a table of `internet` user data from the World Bank containing 248 rows.
	• Open the **SQL window**

`Simplify Query output by organizing columns`	• In Lab 1, we used an asterisk * to select *all* columns from each query.
	`SELECT *`
	`FROM countries`
	• It's also possible to specify which columns you want to select, and in what order. Replace the asterisk with specific column names, separated by commas as follows:
	`SELECT sovereignt, admin, sov_a3, adm0_a3`
	`FROM countries`
	• Try a few more selections of specified columns on your own…

Check for unique values

- To check for uniqueness of values in a column, you can add a **GROUP BY** clause in order to aggregate groups of rows that have identical values:

```
SELECT sovereignt, admin, sov_a3, adm0_a3
FROM countries
GROUP BY adm0_a3
```

- How many rows are there now? If you have the same number of rows, then you don't have a uniqueness problem. If you have fewer rows, then you have duplicates.
- Now try this with the `sov_a3` field in place of `adm0_a3.` How many unique `sov_a3` values are there? How many unique **`CountryCode`** values are there in the **`internet`** table?

Which values are duplicated, and how many duplicates are there?

- If you are using a `GROUP BY` clause, you can also add an *aggregate function* (http://www.sqlite.org/sessions/lang_aggfunc.html) to the selected columns in order to summarize all the rows in each group.
- The `count()` aggregate function is most useful for counting the number of duplicate values. Where the count is 1, the value must be unique. Where the count is greater than 1, the value has duplicates! Try adding the `count()` function to your query:

```
SELECT sovereignt, admin, sov_a3, adm0_a3, count()
FROM countries
GROUP BY sov_a3
```

Sort the results of a query

- It would be nice to sort the results of the query to place the high counts at the top of the table. This can be accomplished by adding an `ORDER BY` clause with the DESCending option:

```
SELECT sovereignt, admin, sov_a3, adm0_a3, count()
FROM countries
GROUP BY sov_a3
ORDER BY count() DESC
```

- Which country has the most duplicates in the `sov_a3` column?

Select the results of a GROUP BY selection according to the results of an aggregate function for each group

- Even easier, you can select out only those groups where the count() is greater than one, displaying only duplicates. This can be accomplished by adding a `HAVING` clause to the `GROUP BY` aggregation. The only difference between `HAVING` and `WHERE` is that `WHERE` will select out records before they are grouped, and `HAVING` waits to calculate any aggregate functions, and then selects out records.

```
SELECT sovereignt, admin, sov_a3, adm0_a3, count()
FROM countries
GROUP BY sov_a3
HAVING count() > 1
```

- Which countries are listed as having duplicates?

How complete is each join? For example, which countries do not have any matching rows in the World Bank internet table?

- This selection is the same as the method you used to map the areas missing data in Lab 1. Add a `WHERE` condition to the `JOIN` so that you can find all the `null` values.

```
SELECT sovereignt, admin, sov_a3, adm0_a3,
internetusers13
FROM countries LEFT OUTER JOIN internet
ON adm0_a3 = CountryCode
WHERE internetUsers13 IS NULL
```

- Try to modify the selection to test a join on `sov_a3`.
- Can you also use this method to find the World Bank records that have no matching row in the countries features?

Taking only the rows that do join from internet to countries, are any missing internet use data?

- Thus far all of the joins have been `LEFT OUTER JOIN`, which keep *all* of the original target table rows. It's also possible to join and keep *only the matching rows*. This is an `INNER JOIN`.
- Modify the previous selection in order to find any countries that *do* match to World Bank records but *do not* contain any valid internet use data.

```
SELECT sovereignt, admin, sov_a3, adm0_a3,
internetusers13
FROM countries INNER JOIN internet
ON adm0_a3 = CountryCode
WHERE internetUsers13 IS NULL
```

Are there any one to many cardinality problems? That is, do any rows from the World Bank internet table join to multiple rows in the Countries table?

- In order to answer this type of question, you'll have to combine a JOIN clause and a GROUP BY clause in the same query. That will allow you to group the results of a join and count how many duplicate results the join produced.
- You'll continue to use an `INNER JOIN`, since you're only concerned about duplication in the rows that successfully joined (the others would turn out `NULL` data anyway).

```
SELECT sovereignt, admin, sov_a3, adm0_a3,
internetusers13, count()
FROM countries INNER JOIN internet
ON adm0_a3 = CountryCode
GROUP BY adm0_a3
ORDER BY count() DESC
```

- The query above tested the join on `adm0_a3`. Now try to test cardinality for a join using `sov_a3` on your own.

Prepare to inspect a dataset's metadata in Tuesday's lab.

- In preparation for lab, please read some background information on error, metadata, and standards:
  - Longley's subsection 11.2.1 on Object-level metadata, including boxes 11.2 and 11.3.
  - Bolstad's chapter on Data Standards and Accuracy (pages 561 to 569)
  - Ken Foote's Error, Accuracy, and Precision module on the Geographer's Craft: http://www.colorado.edu/geography/gcraft/notes/error/error_f.html

Lab 2: Investigating Metadata

Purpose:

This week's challenge is to assemble population data for Tanzania, describe its metadata, and assess its accuracy. To avoid unnecessarily taxing the servers of our data providers, I have given you the .zip downloads and links to the sources.

Data:

- Tanzania Census Data pdf reports: http://www.nbs.go.tz/
- Tanzania GIS Maps: http://www.nbs.go.tz/nbs/index.php?option=com_content&view=article&id=383
- GeoHive: GeoHive: http://www.geohive.com/cntry/tanzania.aspx

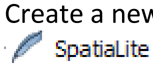
References:

- Error, Accuracy, and Precision by Ken Foote and Donald J. Huebner: http://www.colorado.edu/geography/gcraft/notes/error/error_f.html

Deliverables:

- Document metadata for district-level Tanzania population data from the 2002 and 2012 censuses.

Procedure:

`Set up folders and databases for analyzing Tanzanian Census data`	• Copy the `TZpop` folder from Splinter to your personal Splinter drive. We'll use this data for the next few labs. It contains a folder, sources containing unmodified downloads from the Tanzania Bureau of Statistics. • Start a new QGIS project and save it to your `TZpop` folder. • Create a new SpatiaLite database, `TZpop.sqlite` (right-click the **SpatiaLite**  tree and **create database**).
`Copying Data from a table in a Webpage`	• Find a table organized by administrative areas for which you have geometries that can be mapped(e.g. a polygon shapefile). • Keep in mind that you'll want good database column names, and no extraneous formatting, like commas or decimal points in the numbers. Any non-numerical data in a number column will force the whole column to be imported as `text`. • Tanzania's Census General Report tabulates 2012 census results by region, and then by ward in a separate table for each district. This would be very annoying to map district data, since none of the tables are a direct match to our district geometries. • Fortunately, GEOHIVE has organized the data more pleasantly: http://www.geohive.com/cntry/tanzania.aspx • Select the whole **Expanded administrative units** table and copy. • Open Excel and paste, with the option to *match destination formatting*.

Adjust the data into a regular table format for import into an attribute table or database

- Clean up the column headers to reflect good SQL column names, like so:

Dcode	Dname	capital	regionArea	pop02	pop12
1	Dodoma	Dodoma	41,311	1,692,025	2,083,588
1	Kondoa DC	Kondoa			269,704
2	Mpwapwa DC	Mpwapwa			305,056
3	Kongwa DC	Kongwa			309,973
4	Chamwino DC	Chamwino			330,543

- The Regions are included here as rows, but ideally we want separate columns identifying, for each district, the region that it belongs to. Regions are easily identified since they have a `regionArea`. Each district below a region belongs to that region. Let's make separate column for the region code and region name and move the code and name to the new columns as follows:

Rcode	Rname	Dcode	Dname	capital	regionArea	pop02	pop12
1	Dodoma	1	Dodoma	Dodoma	41,311	1,692,025	2,083,588
1	Dodoma	1	Kondoa DC	Kondoa			269,704
1	Dodoma	2	Mpwapwa DC	Mpwapwa			305,056
1	Dodoma	3	Kongwa DC	Kongwa			309,973
1	Dodoma	4	Chamwino DC	Chamwino			330,543
1	Dodoma	5	Dodoma MC	-			410,956
1	Dodoma	6	Bahi DC	Bahi			221,645
1	Dodoma	7	Chemba DC	Chemba			235,711
2	Arusha	2	Arusha	Arusha	37,576	1,288,088	1,694,310
2	Arusha	1	Monduli DC	Monduli			158,929
…							

- Now cut all the rows containing a region rather than a district and paste them into a separate sheet. Then delete the `regionArea` column since it applied only to regions.
- Hint: this is really easy if you sort the data by `regionArea`!

`Clean up the data formatting so that columns and data types will import into QGIS correctly`	

Clean up the data formatting so that columns and data types will import into QGIS correctly

- The best way to transfer tabular data between programs is a **CSV** file which uses commas to delineate each column. The commas in our population figures are therefore going to be a problem!
- To remove commas from the numbers, select the column with your population data, then right-click and **format cells**. Change to a **Number** format with **0 decimal** places.
- Extraneous white space doesn't seem to be a problem here, but in some data sets you'll have white space surrounding text fields, making it hard to use them for joins. If this happens to you, the `trim()` text function in Excel removes all extraneous whitespace from around text. Just add a new column and use this function to clean up any "invisible" white space surrounding your district names.
- Save the excel workbook.

Check your results

- The `districts` sheet should contain **169** districts, and if it's sorted by `Rcode` and `Dcode`, the beginning should look like this:

Rcode	Rname	Dcode	Dname	capital	pop02	pop12
1	Dodoma	1	Kondoa DC	Kondoa		269704
1	Dodoma	2	Mpwapwa DC	Mpwapwa		305056
1	Dodoma	3	Kongwa DC	Kongwa		309973
1	Dodoma	4	Chamwino DC	Chamwino		330543
1	Dodoma	5	Dodoma MC	-		410956
1	Dodoma	6	Bahi DC	Bahi		221645
1	Dodoma	7	Chemba DC	Chemba		235711
2	Arusha	1	Monduli DC	Monduli		158929
…						

- The `regions` sheet should contain **30** regions, and if it's sorted by `Rcode`, the beginning should look like this:

Rcode	Rname	capital	regionArea	pop02	pop12
1	Dodoma	Dodoma	41311	1692025	2083588
2	Arusha	Arusha	37576	1288088	1694310
3	Kilimanjaro	Moshi	13250	1376702	1640087
4	Tanga	Tanga	26677	1636280	2045205
…					

Save the datasheets as CSV files and import them into QGIS

- Save the `districts` worksheet as a **CSV (Comma delimited) (*.csv)** file, `Districts12.csv`.
  - If you start file names (and subsequent table names) with capital letters, they will conveniently sort to the beginning of the database's list of tables.
- Import the `Districts12.csv` file into QGIS and change the text **Encoding** option so that the spaces in district names import properly. *System* works fine.
- Likewise, save the `regions` worksheet as `Regions12.csv` and import it into QGIS.

Copying data from a table in a PDF document	see below

Copying data from a table in a PDF document

- Open 2002popcensus.pdf and skip to Annex Table 1.A Population by district: 1988 and 2002 on page 15.
- Copy each page of this data table and paste it into a single text document with a simple text editor (e.g. Notepad). Be careful not to include the page numbers.
- This plain text can be imported to QGIS with the **Add delimited text layer** tool or into Excel by using the **From Text** data import tool by assuming that *columns* are delimited by *spaces* rather than *commas*, but there are a few problems:
  1. The first row should contain good database column names, separated by spaces.
  2. Some district names contain spaces, e.g. the Dodoma Urban district in Dodoma or the North A district in Ungunja North. This can be fixed by placing the name in quotes. The beginning of my text file looks like this:

```
Dname pop88 pop02 annualGrowth
1 Dodoma
1,235,327 1,692,025 2.2
"Dodoma Urban" 202,665 322,811 3.3
Kongwa 163,446 248,656 3
Mpwapwa 176,051 253,602 2.6
Kondoa 340,267 428,090 1.6
"Dodoma Rural" 352,898 438,866 1.6
2 Arusha
744,135 1,288,088 3.9
Arusha 132,861 281,608 5.4
```

  3. Four district names span two rows because the district name was long. These were: Sumbwanga Urban, Sumbawanga Rural, Shinyanga Urban, and Shinyanga Rural.
  4. Regions, page numbers, and some other random data are mixed into the table as their own rows. Copy the region code and region name to new columns for each district. Then use an Excel's **sort** tool to group them together and delete them (don't forget Kisarawe—it's in Pwani but falls on the next page of the pdf).
  5. You may also delete Tanzania Zanzibar and Tanzania Mainland.
  6. The pop02 values contain commas. You can fix this by importing the text to Excel (Go to the Data tab, and From Text tool) and using the number formatting options. Please leave pop88 as it is for now, to see how these will import into QGIS and how to later fix them in a database.
- ***Check***: You should have **129** districts with population data for the year 2002.
- Save your districts 2002 data as districts02.csv and import it into QGIS.

Import the data into SpatiaLite

- Import the districts12, regions12, and districts02 tables from QGIS into your TZpop SpatiaLite database.
- Also import the districts.shp and regions.shp shapefiles contained in GIS_Maps.zip to QGIS and into your SpatiaLite database.
- To refresh updates to a database schema in DB Manager (e.g. adding/re-naming tables, adding columns), right-click the database name and **re-connect**.

`Assess the Data`	• According to the criteria we discuss in lab, describe the metadata for the `Districts` shapefile, `Regions` shapefile, GeoHive 2012 data, and Annex Table 1.A Population by district: 1988 and 2002 table.
	o Include assessment of how well the tabular data for 2002 and 2012 districts will join to the geographic data, including rows that do not match or that have cardinality problems.
	o Include assessment of internal consistency.
	o Use additional SQLite aggregate functions (see https://www.sqlite.org/lang_aggfunc.html) to provide descriptive statistics: `avg(x)`, `max(x)`, `min(x)`, and `sum(x)`, where `x` is a column name.
	o For example, to find descriptive statistics of the 2012 population (column `pop12` of table `districts12`), I used the following SQL:
	`SELECT count(), min(pop12), max(pop12), avg(pop12), sum(pop12)`
	`FROM Districts12`
	o To find the descriptive statistics for each region in 2012, I included the region name in the `SELECT`, and grouped by the region code:
	`SELECT Rname, count(), min(pop12), max(pop12), avg(pop12), sum(pop12)`
	`FROM Districts12`
	`GROUP BY Rcode`
	• Write up your metadata in a word processor, and upload it to the Moodle assignment for Lab 2. The metadata should cover district data for 2012 from GeoHive, district data for 2002 from 2002popcensus.pdf, and districts.shp from the National Bureau of Statistics.

`Components of Metadata to look for`	• In class, we agreed to look for the following components of metadata: o Coordinate system / projection o Spatial Scale (cell size, representative fraction, or linear accuracy) o Spatial extent o Temporal Extent o Data hierarchy o Methods / Lineage o Author / publisher / source URL o Primary Key o Joins, join fields o Cardinality issues (completeness, duplicates) o Attributes fields: names, descriptions, data types, descriptive statistics (count, min, max, avg, sum), any bad/missing/null data

PreLab 3: Cleaning Data to Make Joins Work

Purpose:

Many of you noticed discrepancies in the formatting of District names between various sources of data for Tanzania. Discrepancies between formatting, spelling, and data types are by no means unique to developing countries: even attribute data from the U.S. Census or Environmental Protection Agency often has to be cleaned up before it will join properly to geographic features. This PreLab will teach you several techniques for diagnosing and fixing problems with a Join.

Data:

- `ZanzPop.sqlite` is a SpatiaLite database. It contains data that should be identical to what you produced in Lab 2, except it is limited to the Zanzibar semi-autonomous region, composed of Unguja and Pemba islands. It contains the following tables:
  - `Zanz02`, containing attribute data for the 2002 census gleaned from the 2002 Census Analytical Report.
  - `Zanz12`, containing attribute data for the 2012 census gleaned from the GeoHive website.
  - `ZanzDistricts`, containing geographic data for the 2012 census, downloaded from the National Bureau of Statistics.

References:

- SQLite language guide https://www.sqlite.org/lang.html
- SQLite core functions https://www.sqlite.org/lang_corefunc.html
- Chapters 3 and 4 of the *Definitive Guide to SQLite*.

Deliverables:

- Please complete the PreLab 3 Quiz on Moodle by 8am Monday morning.

Procedure:

`Set up a new project`	• Copy the `ZanzPop.sqlite` database from Splinter, start a new QGIS project, save the project, and connect to the `ZanzPop.sqlite` database.
`Insert a new column into a table.`	• Your task is going to be to edit the attribute data of `Zanz02` and `Zanz12` in order to join that attribute data to `ZanzDistricts` properly. Rather than edit the district names directly, let's create a new column for the district names and edit the new column separately. This will allow you to: o a) not accidentally lose or corrupt any information, and o b) safely try new SQL statements to edit data o c) preserve a record of the original district names, in case the names have to be changed for the join • To alter the database schema by adding columns to a table, use an ALTER TABLE statement: **`ALTER TABLE`** `Zanz12` **`ADD COLUMN`** `joinDist text` o `Zanz12` is the name of the table to be altered o `joinDist` is the name of the column to be added o `text` is the data type of the column to be added o Re-connect the database to see the new `joinDist` column.

Create a VIEW to preview how Zanz12 joins to ZanzDistricts.

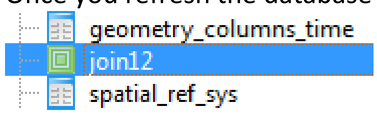
- Rather than iteratively editing data and joining the data to see if the edits worked, you can save a *view* of a *select* query, which will appear like its own table in your database and will update as the tables involved in the query are modified.
- First, create a selection to join `Zanz12` to `ZanzDistricts` on `joinDist` and `district_n`, displaying only the useful columns:

```
SELECT pk_0, geom, district_n, district, pop12,
joinDist
FROM ZanzDistricts LEFT OUTER JOIN Zanz12
ON district_n = JoinDist
```

- If that selection looks good, modify it by starting with:

```
CREATE VIEW join12 AS
SELECT pk_0, geom, district_n, district, pop12,
joinDist
FROM ZanzDistricts LEFT OUTER JOIN Zanz12
ON district_n = JoinDist
```

- Once you refresh the database , an additional view, `join12`, should appear:



- So far, all of the 2012 census data is `NULL` in the `join12` view, but we'll get there…

The beginning of the district field in Zanz12 seems to match the district_n field of Districts. The LIKE operator in conjunction with wildcard characters has the flexibility to match portions of text.

- Try to use the `LIKE` operator in a `SELECT` statement to add flexibility in matching text. First, try to select the Kati district from `Districts12`:

```
SELECT *
FROM Zanz12
WHERE District = 'Kati'
```

- This didn't match any rows, because with the equals operator, the entire text string must match exactly. Try matching the whole name:

```
SELECT *
FROM Zanz12
WHERE District = 'Kati District'
```

- This still didn't work! The *space* character you are typing is not identical to the *space* character as it was downloaded from GeoHive. Remember the mismatched character set problem? No worries… the `LIKE` operator gives you more flexibility…
- You can also concatenate text together with the **||** operator and add a wildcard underscore character _ which will allow a query to replace _ with any other character:

```
SELECT *
FROM Zanz12
WHERE District LIKE 'Kati' || '_' || 'District'
```

  - The above query means to select everything from `Zanz12` where the District equates to `'Kati'` followed by any single character, followed by `'District'`

While the underscore _ wildcard replaces one character, the percent % wildcard replaces any number of characters:

```
SELECT *
FROM Zanz12
WHERE District LIKE 'Kati' || '%'
```

- The statement means to select everything from Zanz12 where the District equates to 'Kati' followed by any number of other characters.

- You can even select all the districts starting with 'K':

```
SELECT *
FROM Zanz12
WHERE District LIKE 'K' || '%'
```

- Can you start to see how LIKE and wildcards can help select and match records when their text is not consistent?
- Could we also use LIKE to improve the results of joining Zanz12 to ZanzDistricts?

```
SELECT *
FROM ZanzDistricts LEFT OUTER JOIN Zanz12
ON district LIKE district_n || '%'
```

- Now that we know we can match subsets of text strings in a join, can we use this information to update the JoinDist column?

Updating a single value in a record

- First of all, let's manually update the joinDist record for one value.

```
UPDATE Zanz12
SET joinDist = 'Chake Chake'
WHERE district LIKE 'Chake' || '%'
```

- The statement means to update the Zanz12 table's joinDist column to equal 'Chake Chake', but only for the rows meeting the condition where the district starts with 'Chake'.

- Look at your Zanz12 table. Chake Chake should be entered in the joinDist column for the appropriate record.
- Look at your join12 view. The Chake Chake record should be joined correctly, since you've edited the joinDist column to match correctly.

Update all the values in a column according to the results of a selection query

- All of the Zanz12 problems can be solved by matching the ZanzDistricts district names to the beginning of the Zanz12 names. Can you update Zanz12's joinDist column with the text from ZanzDistricts's district_n column wherever district_n is like the beginning of Zanz12's district?
- Yes you can! Set joinDist equal to the results of a SELECT query, finding district_n where it is like the beginning of district.

```
UPDATE Zanz12
SET joinDist =
      (SELECT district_n
      FROM ZanzDistricts
      WHERE district LIKE district_n || '%')
```

- Did it work? Look at the `Zanz12` table and the `join12` view to check.
- **Note**: Order is important for the `LIKE` operator. The left hand side of the `LIKE` operator should always contain a string or a text column, while the right hand side of the `LIKE` operator can contain combinations of strings and wildcard characters.

Start working on the 2002 data…

- Can you add a `joinDist` column of type text to `Zanz02`?
- Can you set up a `VIEW` of a join from `Zanz02` to `ZanzDistricts`, similar to the `join12` view?
- It looks like the errors in the 2002 data are not very consistent… so let's use a different strategy here. First, copy all the data from the `District` column to the `joinDist` column with an UPDATE clause:

```
UPDATE Zanz02 SET joinDist = District
```

- Check the `join02` view. `Wete`, `Micheweni`, and `Mkoani` should already be matching correctly. The others need some work.

Translate Swahili to English using the replace() function

- The only pattern I can observe in the `Zanz02` data is that several Swahili words are used in place of their English equivalents in the `ZanzDistricts` table. If you know these translations, you can use a `replace()` function in SQL like a *Find and Replace* tool in a word processor. I've outlined the translations below:

Swahili	English
Kaskazini	North
Kusini	South
Mjini	Urban
Kati	Central
Magharibi	West

- The `replace(X,Y,Z)` function takes three arguments: `X` refers to the string that is to be changed, `Y` refers to the string that is to be found in `X`, and `Z` refers to the string that will replace `Y` in `X`. Therefore, try translating `South` into `Kusini` for every record in `JoinDist`:

```
UPDATE Zanz02
SET joinDist = replace(joinDist, 'South', 'Kusini')
```

- Try to also replace `Kaskazini` with `North` with the same technique.
- Did your `replace()` function work in the `Zanz02` table? What about the `join02` view?
- The situation is improving, but maybe it's tedious to use SQL to edit rows one by one. SQL was much more useful when the same solution could be applied to numerous rows at the same time.

`Add the Zanz02 table to QGIS to edit its attribute table manually`

- You've probably noticed that it's impossible to manually edit any data in DB Manager without writing a query. Since there's little consistency in the 2002 Zanzibar data errors, let's find a way to manually edit the records.
- Right-click the **`Zanz02`** table and **Add to Canvas**.
- Minimize DB Manager and look at the QGIS map. A Z**anz02** table should be included in the **Layers** panel.
- Open the Zanz02 attribute table in QGIS
- Start an editing session with the **toggle editing** button:
- Now you can edit any of the `joinDist` records in the table. Double check your `join02` view in DB Manager to be sure you fix each of the remaining problems. DB Manager will not update with the new edits until you **save** them in the Attribute Table in QGIS and then **refresh** DB Manager.
- When you are finished, **save** and **toggle editing** back off .
- Return to DB Manager and refresh the `join02` view to confirm that your edits worked. If not, go back to edit again.
- **Warning**! Don't try this with `SELECT` statements or with `VIEWS`. A `VIEW` is intended for viewing data, not for editing it.

`Don't forget…`

- Remember to double-check cardinality *completeness* and *duplicates* before you call your join a success.

`Thinking more broadly about fixing data for joins`

- In your own GIS projects, data will likely have its own unique problems different from this example. As a general rule of thumb, if you see a *pattern* in any of the data problems, you can probably exploit that pattern to automate a solution in one way or another.
- There are numerous other text functions available in SQLite: https://www.sqlite.org/lang_corefunc.html and they include `trim()` for removing characters from the beginning & end of a string,, `lower()` and `upper()` for converting to all lower case or upper case, and `substr()` for returning a subset of a string based on position and length of the subset.
- If you need to remove a string of characters from a column all together (e.g. remove commas from a column of type string that accidentally is storing numbers), use a `replace()` function to replace the string of characters with an empty string: e.g.: `replace(popWithCommas , ',' , '').`
- If your data has a problem of mismatched data types (e.g. you end up needing to convert string to integer or integer to string in order to join), follow a two-step fix:
    - A) use `ALTER TABLE` to create a new column of the proper data type
    - B) use `UPDATE` to set the new column equal to the old column. This process should automatically convert data types for you. This is called type-casting. See https://www.sqlite.org/lang_expr.html#castexpr on how this works, and also how you can do it inside SQL statements.
- Try removing the commas from `Pop88` in `Zanz02` and converting `Pop88` to an integer in a new column…

Open Source Coordinate Systems	• In the next lab, we will customize and project coordinate systems, so start getting familiar with these by looking at what you already have. • The coordinate system for the `ZanzDistricts geom` column is stored in the `geometry_columns` systems table, in the form of an `SRID` code. The majority of `SRID` codes are maintained by the International Association of Oil & Gas Producers (the EPSG authority) at http://www.epsg-registry.org/ • Look up the **SRID** for our data using the **retrieve by code option** on the EPSG website.. What coordinate system is it using? • In preparation for Monday's lecture and lab, look over the Cartographic projection procedures for the UNIX environment, a guide to the Proj.4 open source projections library: http://trac.osgeo.org/proj/. Try to find some projection systems that you've seen before.

Lab 3: Projections in QGIS and SpatiaLite

Purpose:

The goal of this lab is to create a population density map for Tanzania in 2012. Before you calculate the *area* column required for a density map. This will require checking the spatial data's coordinate system, making sure that it is correctly defined, and transforming to an equal area projection customized for Tanzania.

Data:

- Use the data you prepared in Lab 2, originating from GeoHive and the National Bureau of Statistics.

References:

- SpatiaLite functions reference: http://www.gaia-gis.it/gaia-sins/spatialite-sql-4.2.0.html
- SpatiaLite new geometry columns: http://www.gaia-gis.it/gaia-sins/spatialite-tutorial-2.3.1.html#t5
- Proj.4 reference manual: ftp://ftp.remotesensing.org/proj/OF90-284.pdf
- EPSG Geodetic Parameter Registry: http://www.epsg-registry.org/
- Spatial Reference: http://spatialreference.org/

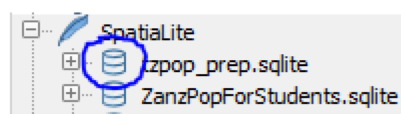
Deliverables:

- A population density map for Tanzania in 2012.

Procedure:

`Set up a new QGIS project with the Tanzania GIS Maps data`

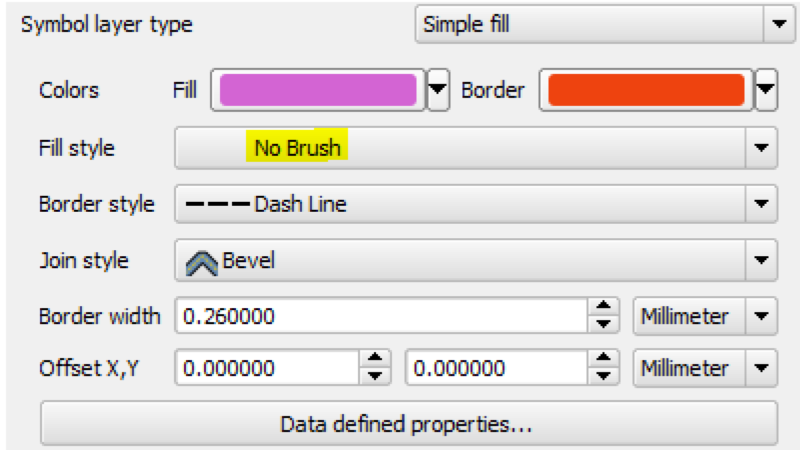
- Create a new QGIS project for Lab3.
- Connect to the SpatiaLite database you created for Lab2. The database should contain:
  - Tanzanian Districts geometries with 169 records
  - Tanzania 2002 census data table with 129 records
  - Tanzania 2012 census data table with 169 records
- You may disconnect any other database. In the QGIS Browser panel's SpatiaLite tree, right-click any other database in the Browser panel and *delete.* This only deletes the *connection* to the database, not the database itself.



- From your SpatiaLite database, add the `Districts` geometry to the map.
- From `GIS_Maps.zip`, extract the `Regions` and `Water_Body` shapefiles and add them to the map.
- Check the Coordinate Reference Systems of each of the three layers. This is found in the layer properties General section. What coordinate system are they using, and what is its SRID?

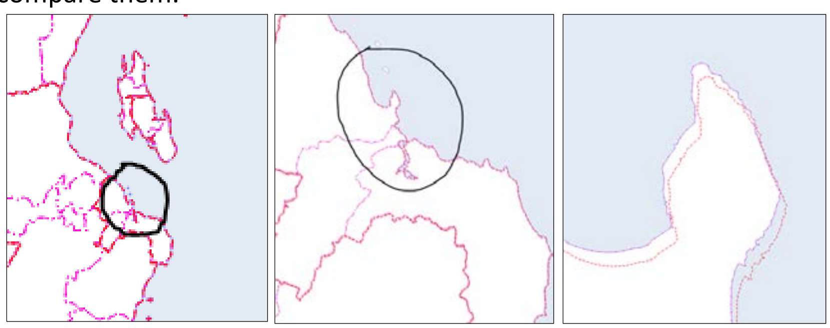
Compare the internal spatial consistency of these data layers

- Let's check the spatial consistency of these layers. Change their backgrounds to *nobrush* or make them mostly transparent and change their borders to distinctive patterns or colors.



  - Much spatial data is distributed with no defined coordinate system, or incorrectly defined coordinate systems—how can you check this? Let's first verify systems against each other for internal consistency.

- Zoom in to the Msasani peninsula in Dar es Salaam coastal area, where the data quality should be good and there's enough detail in the borders to compare them.



- The regions, shown here in Red, do not align with the water body or districts layer, but which is correct?

Add an Open Street Map base layer to the map for comparison

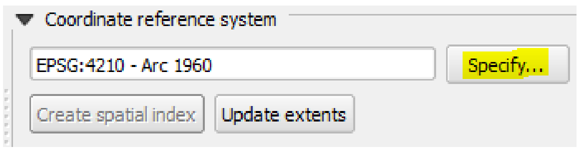
- Let's check these against an Open Street Map base layer.S
- Go to **Web** -> **OpenLayers Plugin** -> **Open Street Map** -> **Open Street Map**
  - If the OpenLayers Plugin menu is not there, install it by going to **Plugins** -> **Manage and Install Plugins**.
- Change project coordinate reference system to match the Open Street Map (OSM) data.
  - First, change the **project properties' CRS** options to **enable 'on the fly' CRS transformation**.
  - Then, set the CRS to the **WGS 1984 Pseudo Mercator** system used by web mapping services like OSM. Its SRID is 3857.
- Reorder your map layers to move the OSM data to the bottom.

- It seems like `Regions` is correctly defined in the Arc 1960 geographic coordinate system, but `Districts` and `Water_Body` are off by about 50-100 meters E/W and 250-300 meters N/S.
- Errors on this magnitude are often due to a mis-defined datum (geographic coordinate system) or failure to apply a geographic transformation when transforming data from one datum to another.

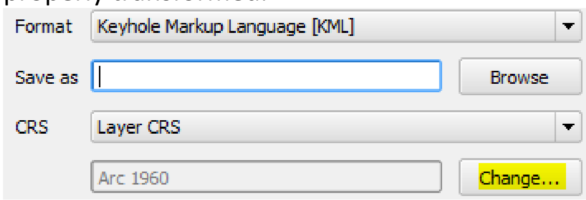
```
Advice and warnings
regarding
projections in QGIS
```

- Projection on the Fly does not alter any geographic data. It simply projects all data layers into the coordinate reference system (CRS) you have selected in the Project Properties. This information is displayed in the status bar at the bottom of the map. In the example below, I'm using 3857, WGS 1984 Pseudo Mercator.

The CRS Status button  is a shortcut to change the Project CRS.
- **Warning**: projection on the Fly is for ***visualization only*** and often requires you to refresh the canvas in order to display properly. This is true in any GIS, but especially in QGIS. There are so many surprising ways in which projection on the fly can muck up your analysis…
- You can also **specify** the CRS for any layer in the layer properties general section. This *does not* transform the coordinates. It simply *redefines* the CRS for a layer. It will change the way QGIS interprets the coordinates, but it will not transform the coordinates themselves.



- If you want to *transform* a layer from one CRS to another CRS, right click the layer, **Save As…** and **change** the CRS. Through the process of saving a new version of the layer and selecting a new CRS, the coordinates will be properly transformed.



Try to fix the Water_Body shapefile in QGIS	• Is it possible that someone accidentally specified `Water_Body` to Arc1960 from a more common system, WGS 1984, without transforming it? • Try **specifying** `Water_Body` to the WGS 1984 geographic coordinate system (SRID: `4326`) in its layer properties. • That attempt *doubled* the spatial error! Could it be that `Water_Body` was at some point *specified* from Arc 1960 to WGS 1984 without a proper transformation, and then *transformed* from WGS 1984 back to Arc 1960? How can you fix that mess? Reverse those errors step by step… • **1) Specify** `Water_Body` back to Arc 1960 (SRID: `4210`) • **2) Transform** `Water_Body` to a shapefile in WGS 1984 by using **Save As…** I named mine `WaterWGS1984`. • The new `WaterWGS1984` appears in the same location as `Water_Body` on the map thanks to projection on the fly, but its coordinates have been transformed from Arc 1960 to WGS 1984. • **3)** Now **specify** `WaterWGS1984` to Arc 1960 in its layer properties. • `WaterWGS1984` should now line up with OSM and the Regions!
Import Water and Regions to SpatiaLite	• **Import** the `WaterWGS1984` layer to your SpatiaLite database, naming the new table `Water`. Use the import vector layer's drop-down menu to select a *layer*, as opposed to browsing to the *shapefile*. The shapefile may not have been updated with the specified Arc1960 SRID yet. • Check the `geometry_columns` table to see if it has the correct `SRID`. If so, remove the shapefile water layers from the map and add your database version to the map. • Likewise, import the `Regions` layer, naming the new table `Regions`.
Take a breather…	• You just fixed the `Water_Body` shapefile's projection to match `Regions` and OSM by first transforming and then specifying its coordinate system. Districts, however has already been imported to SpatiaLite. This next section will walk you through the same conceptual steps of coordinate system transformations and specifications, but this time in a database.

Inspect the Districts geometries in SpatiaLite using DB Manager.	• Before we try to fix `District`, let's look at the SRID and coordinates stored for the Kinondoni district in Dar es Salaam. This is the district we've zoomed into on the map. Open DB Manager and use the `Srid()` and `AsText()` functions to unpack the geometry blob:
	`SELECT Srid(geom), AsText(geom)`
	`FROM districts`
	`WHERE district_n = 'Kinondoni'`
	• The result is SRID `4210` and a very long string of geographic coordinates, starting like this:
	`MULTIPOLYGON(((39.122661 -6.563863, 39.123254 -6.563952,`
	o To see just how many coordinate pairs make up this polygon, copy the results of your query and paste them into Notepad.

Inspect the Districts coordinate system metadata in SpatiaLite

- To see the database's registry of geometries and their metadata (including coordinate systems), look at the `geometry_columns` table. The first five columns look like this:

f_table_name	f_geometry_column	geometry_type	coord_dimension	srid
districts	geom	6	2	4210
regions	geom	6	2	4210
water	geom	6	2	4210

  - `F_table_name` refers to the table name
  - `f_geometry_column` refers to the column name in which the geometry is stored
  - `geometry_type` refers to the type of geometry, where 6 is apparently `MULTIPOLYGON`
  - `coord_dimensions` refers to whether the data is 2D or 3D
  - SRID refers to the coordinate system ID, usually an EPSG code.

Transform coordinates in a SpatiaLite database by adding a new geometry column and updating the column with a transform function.

- Add a `BLOB` column to the Districts table for storing new geometries:

```
ALTER TABLE Districts ADD COLUMN newgeom BLOB
```

- Calculate the new geometry by **transforming** the old geometry:

```
UPDATE districts
SET newgeom = Transform(geom, 4326)
```

  - The `Transform()` function took 2 parameters:
    1. an existing geometry column
    2. the SRID of the new coordinate system (WGS 1984 in this case)

- Let's re-examine the `newgeom` SRID and geometry for the `Kinondoni` district using the `Srid()` and `AsText()` functions. My results show that the `transform` function has indeed transformed the SRID to 4326 and the geometry to:
`MULTIPOLYGON(((39.123532 -6.565993, 39.124125 -6.566081`

Specify the SRID of geometries and define the spatial metadata for geometries

- Specify, or re-define, the coordinate system of each record in `newgeom` using the `SetSrid()` function:

```
UPDATE Districts
SET newgeom = SetSrid(newgeom, 4210)
```

  - The `SetSrid` function took 2 parameters:
    1. an existing geometry column
    2. the SRID of the new coordinate system (ARC 1960 in this case)

- Let's re-examine the new SRID and geometry for the Kinondoni district. My results show that the `SetSrid()` function has changed the SRID to `4210`, but not transformed the coordinates:
`MULTIPOLYGON(((39.123532 -6.565993, 39.124125 -6.566081,`

Register the new geometry column in the database's spatial metadata tables	• Thus far our `newgeom` column has not appeared in the `geometry_columns` metadata table, and it cannot be used as geometry by the database yet. Use the `RecoverGeometryColumn()` function to register the new geometry: **SELECT RecoverGeometryColumn**('Districts', 'newgeom', 4210, 'MULTIPOLYGON') o This function took five parameters in the following order: 1. table name in single quotes (a string) 2. geometry column name in quotes (a string) 3. coordinate system SRID to be used by the new geometry (Arc 1960 in this case) 4. geometry data type in quotes. Reference all the geometry types here: http://www.gaia-gis.it/gaia-sins/spatialite-cookbook/html/new-geom.html o This function returned a single value in a table. If it returns `1`, the function worked, but if it returns `0`, it did not work. Boolean logic…
Check your results and discard old geometry metadata	• Refresh the whole database. It should now look like there are two `District` polygon geometry tables. These refer to the same attribute data, but for geometry one is using `geom` and the other is using `newgeom`. The table info and `geometry_columns` table should indicate that both `geom` and `newgeom` use the Arc 1960 coordinate system. • Try adding your `newgeom` version of Districts to the map canvas in QGIS. Does it line up? I hope so! • If this is all good, let's delete the metadata for the old geometry column to avoid confusion. Do this with the `DiscardGeometryColumn` function: **SELECT DiscardGeometryColumn**('Districts', 'geom') o This function took five parameters in the following order: 1. table name in single quotes (a string) 2. geometry column name in quotes (a string)
Take a breather…	• You fixed this data to accurately match OSM, and the data uses the Arc 1960 geographic coordinate system. But, you know that geographic coordinate systems are no good for calculating areas, and we'd like to use area to find population density. Therefore…
Custom Projection Considerations	• Let's make a custom projection for Tanzania that is projected from the Arc 1960 geographic coordinate system our data is already using. • We want an *equal area* projection so that area calculations will be accurate. • Tanzania is a large country with a fairly compact shape, in the sense that it is about as wide from east to west as it is tall from north to south. Therefore, we can use an *azimuthal* projection.

Look up example projections and your study area's spatial extents in QGIS	• Before we try to make our own custom projection, let's examine the parameters for some familiar projections.
	• Go to project properties CRS an enable on the fly projection.
	• Look up Arc 1960 and copy its Proj.4 text to notepad. Arc 1960's Proj.4 is: `+proj=longlat +ellps=clrk80 +towgs84=-160,-6,-302,0,0,0,0 +no_defs`
	• Likewise, look up and copy **Arc 1960 UTM 37S** (the zone covering Dar es Salaam), and any **Lambert Azimuthal Equal Area** projection.
	• You will also need to know the extent of your study area in Latitude and Longitude geographic coordinates, and maximum linear dimensions of your study area in meters along the N/S axis and E/W axis.
	  o For the Lat/Lon extent, you have a layer on the map in Arc 1960 geographic coordinates already. Check the layer properties metadata, scroll all the way to 'properties' of the metadata, and find the extent there.
	  o To estimate the linear dimensions, use projection on the fly to put the map in the Arc 1960 UTM 37S coordinate system and use the measure tool to estimate distance in meters across Tanzania along the N/S and E/W axes.

Create a custom projection in QGIS	• In notepad, write a custom Proj.4 projection • In QGIS, go to **Settings -> Custom CRS** • Click **Add new CRS** • Enter a name, e.g. `Tanzania Azimuthal Equal Area` • Copy your Proj.4 parameters from Notepad to the **Parameters** text box. • Click **OK** to create the projection. • Try using your custom projection as the coordinate reference system in the Project Properties.

Create a custom projection in SpatiaLite

- Take a look at the `spatial_ref_sys` table. This table is automatically created with new SpatiaLite databases, and it contains a pretty extensive list of map projections.
- To create a custom projection in SpatiaLite, you have to add it to the `spatial_ref_sys` table, using SQL for inserting a row into a table.:

```
INSERT INTO spatial_ref_sys  VALUES(110000, 'gg328',
110000, 'Tanzania Azimuthal Equal Area', '+proj=laea
+lat_0=-6.4 +lon_0=35 +x_0=800000 +y_0=800000
+ellps=clrk80 +towgs84=-160,-6,-302,0,0,0,0 +units=m
+no_defs', 'Undefined')
```

  - **Caution!** The Proj.4 text string should have a space between each parameter, but *no line breaks*. If you copy the SQL code from above, make sure there are no line breaks in it!
  - The VALUES must come in exactly the same order as they appear in the table, from top to bottom here:

Column	Data Type	Description
srid	INTEGER	Choose a unique SRID out of the range of other SRID's.
auth_name	TEXT	Any string referring to you...
auth_srid	INTEGER	This should be identical to the SRID
ref_sys_name	TEXT	Text name of the projection
proj4text	TEXT	Proj.4 parameters
srtext	TEXT	This is for a WKT (well-known text) representation of the projection. You don't need it inside of SpatiaLite and QGIS, so write 'undefined'.

Add a geometry column to Districts for your custom coordinate system

- Previously, you added a geometry column by first creating a `BLOB` column, then filling in the geometries with `Transform()` and `SetSrid()` functions, and finally registering the metadata with the `RecoverGeometryColumn()` function.
- If you know that you'll never need to change the SRID of a geometry column, you can shortcut some steps by simultaneously creating the column and registering the metadata with the `AddGeometryColumn()` function:

```
SELECT AddGeometryColumn('Districts', 'TZlaea',
110000, 'MULTIPOLYGON')
```

  - This `AddGeometryColumn()` function took the same four parameters as the `RecoverGeometryColumn()` function, in this order:
    1. table name in single quotes (a string)
    2. geometry column name in quotes (a string)
    3. coordinate system SRID to be used by the new geometry (your Tanzania Lambert Azimuthal Equal Area projection in this case)
    4. geometry data type in quotes. Reference all the geometry types here: http://www.gaia-gis.it/gaia-sins/spatialite-cookbook/html/new-geom.html

`Transform and Calculate Area`	• Transform the geometries with an UPDATE statement and `Transform` function, taking your correct `newgeom` as the input geometries.
	  o Let's examine the new `TZlaea` geometry for the Kinondoni district. My results show that the `transform` function has indeed transformed the coordinates to SRID `110000`, and the coordinates are now much larger numbers representing *meters* measured from the origin of the projection, rather than *latitude* and *longitude* geographic coordinates: `MULTIPOLYGON(((1255945.586377 779815.685393, 1256011.052779 779805.363902`
	• Add a column named `area_sqkm` of type REAL to the `Districts` table.
	• Calculate area with an area function:
	  `UPDATE` `Districts` **`SET`** `area_sqkm =` **`Area`**`(TZlaea)/1000000`
	  o We have divided by `1,000,000` because I know that `Area()` will use the units of the coordinate system (meters) and I want to convert to square kilometers.

	• **Warning**! You may not like to see two versions of the districts table, but *do not delete one*. They refer to different geometry columns in *the same data table*. If you delete one, you delete them all. To remove a duplicate geometry from your database view, use the `discardgeometry()` function.

`Make a Map!`	• Using the SQL skills you already know, can you now:
	  o Create a new column in `Districts12` for population density, named `density12`
	  o Calculate population density in people per square kilometer
	  o Join `Districts12` to `Districts` to make a map in QGIS?
	• When you're finished, upload the map to Moodle.
	• The next page has some additional notes, not necessary for this lab, but important for managing projections in SpatiaLite and QGIS on your own.

Cautionary Note about SRIDs and Geometry Columns	• Once an SRID is registered for a column with `RecoverGeometryColumn` or `AddGeometryColumn`, it really sticks, preventing you from using a `Transfrom` function or `SetSrid` function on the column. • Therefore, if you're setting up a new geometry column and you know that you might need to transform or redefine the coordinate system used for that column, use the `BLOB` data type at first. Then use `RecoverGeometryColumn` once you no longer need to change the SRID. • If you know the SRID for the column will never change, you can simply set up a new geometry column with the `AddGeometryColumn` function. • If you have a geometry column that is already registered and it's necessary to change it, first `DiscardGeometryColumn`, then `Transform` and/or `SetSrid`, and finally `RecoverGeometryColumn`.
Problems with transferring data in custom projections between QGIS and SpatiaLite databases	• If you ever want to transfer spatial data from SpatiaLite into QGIS and back into SpatiaLite again with a custom projection, you'll have to force-add your custom projection into the local QGIS system. At this time, it's only possible to do this on your own private computer: • Connect to the `srs` SQLite database located in the QGIS program files. You don't have access to change `srs.db` in the lab computers, but it's located at: `c:/program files/QGIS Brighton/apps/QGIS/resources/srs.db` • Run an SQL `INSERT` statement customized for your projection, much like you did for the SpatiaLite `spatial_ref_sys` table: **INSERT INTO** tbl_srs **VALUES**(110000, '+proj=laea +lat_0=-6.4 +lon_0=35 +x_0=800000 +y_0=800000 +ellps=clrk80 +towgs84=-160,-6,-302,0,0,0,0 +units=m +no_defs', 'Undefined') o The fields and data types of the `tbl_srs` table entered above are: 1. srs_id, integer 2. projection_acronym, text 3. ellipsoid_acronym, text 4. parameters, text 5. srid, integer 6. auth_name, text 7. auth_id, integer 8. is_geo, integer 9. deprecated, Boolean integer 10. noupdate, Boolean integer o The `srid`, `auth_name`, and `auth_id` must be identical to the SRID for your custom projection as it is entered into the `spatial_ref_sys` table. • Once you insert that row into the `srs.db` database, your projection is fully searchable and usable like any other projection in QGIS.

Lab 4: Spatial Data Accuracy & Integrity

Purpose:

This week's challenge is to assess the spatial accuracy of data, including coordinate systems and topological relationships, for population data layers from Tanzania.

Data:

- Tanzania 2012 District Data from the National Bureau of Statistics

References:

- Error, Accuracy, and Precision by Ken Foote and Donald J. Huebner
- Bolstad GIS Fundamentals: Data Standards and Accuracy
- Bolstad GIS Fundamentals: Digitizing and Editing
- QGIS Manual: Topological Editing and Topology Checker
- QGIS Training Manual Module 6, Lesson 2: Feature Topology
- http://grass.osgeo.org/grass64/manuals/v.in.ogr.html
- http://grass.osgeo.org/grass64/manuals/v.clean.html

Deliverables:

- Show the instructor a map for which Tanzanian districts have no geometry errors.
- Create a GitHub account and give the instructor your user name.

Procedure:

`Clean up your DB first by selecting and renaming the columns you want, and registering any new geometry columns`

- Is your database full of random names and extra tables and columns by now? Let's make a fresh version of the Districts table and discard the old.
- You might want to back up your database before you try this.
- Use a CREATE TABLE statement, much like a CREATE VIEW statement, to copy data from another table, select only the columns you want, and rename any column. For my example, I have a table `Districts` with the following fields: `pk, geom, district_c, district_n, newgeom, TZlaea, area_sqkm.` I want to create a new table with just one geometry column, named geom, with the data from `TZlaea`.
- First preview the `SELECT`:

```
SELECT pk, TZlaea AS geom, district_c, district_n,
area_sqkm
FROM Districts
```

  - The AS keyword allowed me to re-name `TZlaea` to `geom.`

- If it looks good, create a table for it:

```
CREATE TABLE DistrictsG AS
SELECT pk, TZlaea as geom, district_c, district_n,
area_sqkm from Districts
```

Register new geometry metadata records and delete old ones as necessary	• The table looks like just an attribute table so far, but you still need to register its geometry metadata with `recovergeometrycolumn()`: • **SELECT** `recovergeometrycolumn('DistrictsG', 'geom', 110000, 'MULTIPOLYGON')`Before deleting the old Districts table, delete any of its associated geometry metadata with `discardgeometrycolumn()`, e.g.: `SELECT discardgeometrycolumn('Districts', 'newgeom')` • **Note**: cleaning up attribute tables with no geometries is much simpler, since you don't need to `discardgeometrycolumn()` or `recovergeometrycolumn()`.
Remove the old geometry metadata and the old tables	• Let's also check how much disk space the database is using, by looking at the `.sqlite` file's size. Mine is presently 34.8 megabytes, or 35,729 kilobytes. • Now delete the Districts table, either by right-clicking and selecting Delete, or using SQL: **DROP TABLE** `Districts` • The table is deleted from the database, but the size is still 34.8 megabytes! • This is where `VACUUM` is handy: `VACUUM` o That's it, but be patient. It's making a fresh copy of the database, so it can take a little while. o Now my database is just 16.2 megabytes!
Check Geometries. Do the Districts suffer from undershoots, overshoots, dangling nodes, rat's nests and slivers? Let's hope not, but do an initial check with Geometry Checker.	• Add `DistrictsG` to the map canvas from your spatialite database. • Save `DistrictsG` as a shapefile in a new folder, `editGeom`, and name it `districtsGR.shp`. Keep the same coordinate system. • Set **the Project CRS** properties to your custom projection (one of the more recent ***Generated CRS** projections) and *disable projection on the fly.* o The tools in this lab do not work well if projection on the fly is enabled because they acquire the extent from the map canvas and apply it without transforming the coordinates • Go to **Vector** -> **Geometry Tools** -> **Check Geometry Validity** • Select your `DistrictsGR` Layer • Check "**save errors location**" and save as a new shapefile, `geom_errors_1.shp` o Click OK and wait… o How many geometry errors were there? • Zoom into some of the error points on the map. • We have some problems…

Identify topological errors
In addition to geometry errors, there may be overlaps, duplicates, or gaps between polygons.

- Enable the **Topology Checker** and **GRASS** plugins by going to **Plugins** -> **Manage and Install Plugins**.
- Go to **Vector** -> **Topology Checker** -> **Topology Checker**
- Click Configure
- Add four rules for the `DistrictsGR` layer:
  - Must not overlap
  - Must not have gaps
  - Must not have invalid geometries
  - Must not have duplicates
- Zoom into a small area of the map and **Validate Extent**. It may take some time to run… especially for the whole map at once.
  - The data layers and the data frame must be in the same coordinate system for this to work!
- I zoomed in to a few districts around Mount Kilimanjaro, and despite that small area, found over 500 errors!

GRASS Can Help:
GRASS is another open-source GIS software package, increasingly integrated into QGIS with the GRASS plugin and its toolbar, and with the addition of GRASS functions to the Processing Toolbox's Advanced Interface.

- To date, the best automated open source solution for sanitizing bad geometries is to import the data into GRASS, an open source GIS software that builds all of its vector datasets using topology. GRASS will automatically try to fix your geometry errors as it imports shapefiles to a GRASS mapset topology.
- First, we need to create a new Database, Location, and Mapset to get started with GRASS.
- Go to **View** -> **Toolbars** -> **GRASS**
- GRASS establishes a folder as a **database**, inside of which it creates a **location** with a CRS and extent, and finally inside of a location it creates a **mapset**.
- Zoom the map to the extent of the districts layer.
- Create a **new mapset**.
- Direct GRASS to a folder for its dataset; I suggest making a `tzdataset` folder inside your `editGeom` folder.
- Create a new location: `tzlocation`
- Set the **projection** to the ***Generated CRS** from the `districtsG` geometry.
- Set the default GRASS region to the extent of Tanzania.
  - In QGIS, zoom to the extent of the DistrictsGR layer
  - Copy this to the default GRASS region with **Set current QGIS extent**
- Name the mapset: `tzmapset` Next-> Finish!
- **Display the current GRASS region** and makes sure it encompasses all of Tanzania. You may have to zoom out to see it.
- If there is any problem, **edit the current GRASS region** to encompass all of Tanzania.

With a GRASS mapset started, you can now import shapefiles into GRASS

- Open GRASS tools.
- Under the Modules List, find `V.in.ogr.qgis`
- This import tool contains two options to drastically reduce the number of topology errors:
    - *Snapping threshold* will snap together any vertices within a given threshold. Even if we use a very small threshold (5 meters), this will fix a large number of errors where there were slight gaps or overlaps in polygon boundaries. It will also simplify boundaries if they have multiple verticies within 5 meters.
    - *Minimum size* will merge any very small polygons (splinters) with an adjacent polygon. The small polygon will be merged with the polygon it shares its longest boundary with.

- Use the following settings for `V.in.ogr.qgis`
    - Loaded Layer: `DistrictsGR`
    - Name for output: `DistrictsGrass`
    - **Show advanced options** ->
    - Snapping Threshold: `5` (meters)
    - Minimum Size (in meters squared): `150`
    - Do not override dataset projection (they should be the same anyway)
    - **Run**, and **wait** until the view output button becomes active (it takes some time after reaching 100%)

- Add GRASS vector layers to the map from your `tzdataset` Gisdbase, `tzlocation` Location, `PERMANENT` mapset, `DistrictsGrass` Map name.
- One by one, add the following layers to the map and inspect them.
    - `Topo node` represents the underlying topology's nodes, the intersections between lines
    - `Topo line` represents the underlying topology's edges, each beginning and ending at a single node
    - `Topo point` appears on the map like points, but represents the centroids of polygons formed by the topology's lines.
    - `Layer 0` represents areas where there were gaps, and these polygons have filled them. Zoom in to see a few of these.
    - `Layer 2` represents polygons removed from the `DistrictsGR` because multiple polygons had overlapped. Zoom in to a few of these, highlight the `DistrictsGR` layer, and use the identify tool to highlight the `DistrictGR` polygons surrounding any of the `Layer 2` polygons.
    - `Layer 1` represents `DistrictsGR` as fixed by GRASS.

- Save `Layer 1` (it may appear as `DistrictsGrass 1`) as a new shapefile: `DistrictsGrass.shp`
- Remove all layers but `DistrictsGrass` and `geom_errors_1` from the map.
- Dissolve DistrictsGRASS.shp on the `District_N` field (**Vector** -> **GeoProcessing Tools** -> **Dissolve**) and save as `DistrictsDisslove.shp`

Did GRASS help?

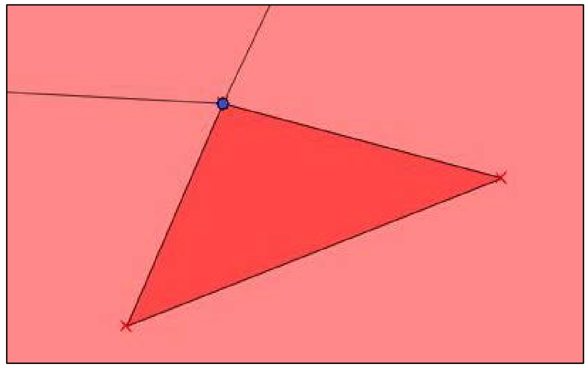
- Run **Check Geometry validity** again on `DistrictsDissolve`, saving the errors as `geom_errors_2.shp`.
- Compare `geom_errors_1` and `geom_errors_2`.
  - o Where has GRASS improved the errors, and where has it created more?
- Run **Topology Checker** again, this time for the whole extent. Remove the rules for `DistrictsGR` and set rules for overlaps, gaps, invalid geometries, and duplicates for `DistrictsDissolve` layer.
  - o Do you notice any difference from before GRASS?

Topological Editing: Finally, let's learn how to manually edit the remainder of these problems, with support of snapping and topological editing.

- Go to **Settings** -> **Snapping Options**
  - o Check the `DistrictsDissolve` layer, and set each to a tolerance of `10 pixels`.
  - o Check **Avoid Int** for each, which prevents polygons from overlapping each other.
  - o Check **Enable Topological Editing** so that QGIS can enforce topological rules while you edit.
- Save a backup copy of `DistrictsDissolve`, just in case editing doesn't go well.
- Try to fix up all of the geometry errors in the districts.
- Turn on the **Digitizing** and **Advanced Editing** toolbars:
  - o View -> Toolbars -> Digitizing
  - o View -> Toolbars -> Advanced Editing
- Review the topological editing resources linked from the beginning of the lab. For quick reference, here are some useful editing tools:
  - o  Toggle editing turns editing on and off for the selected layer.
  - o  Undo an edit.
  - o  Is for selecting a feature
  - o  Deselects all features
  - o  Deletes the selected feature(s)
  - o  Selects, moves, inserts and deletes nodes/vertices. Click to select, click and drag to move, double-click to insert, and select and press *delete* on the keyboard to delete.
  - o  Reshape Feature allows you to add an area to a selected polygon (click inside polygon, trace a region to add to the polygon outside, and right-click inside the polygon) or remove an area from a polygon (click outside polygon, trace a region to remove inside the polygon, and right-click outside the polygon).
  - o  Delete ring removes a "hole" from inside a polygon.

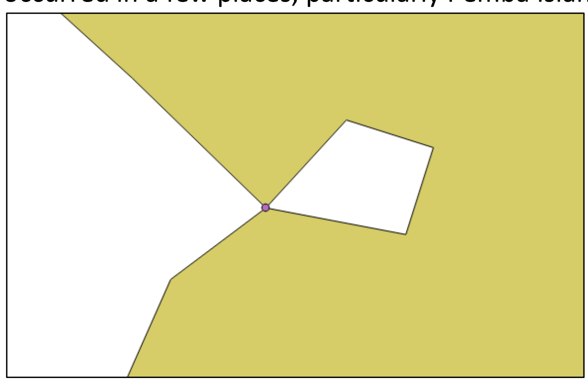
The Most Annoying Case:

- In a situation like the one shown below, there are likely 2 overlapping spurious polygons or polygon parts. It will require deletion of the spurious polygons and editing the remaining nodes/vertices to close the gap in the hole. Where the polygons intersect (here, a blue dot representing a geometry error), there are likely two nodes, and one should be deleted.



Another Annoying Case:

- Sometimes if a very sinuous boundary came within the snapping distance (we used 5 meters) to itself, GRASS snapped one side to the other. This has occurred in a few places, particularly Pemba Island.



Final Thoughts to think about

- One test for slivers and holes in a data layer is simply to **dissolve** the whole thing and see if there are any cracks. There are still cracks in your dataset, and you could find them by looking for the gaps in the topology checker.
  - If they're a hole inside a polygon, just delete the hole.
  - If they're a gap between polygons, use the Reshape Feature to fill the gap.
- Some lakes overlap with some land areas. In Northern Tanzania, around Lake Victoria, the lake covers legitimate land area. In Western Tanzania, around lake Tanganyika and Lake Malawi/Nyasa, the true lake shape covers up district areas that are not actually land. How could you use the vector geoprocessing tools and the editing tools to fix these problems?

Show your results to the professor

- Hopefully, you have no more geometry errors!

Reimporting to SpatiaLite requires some tricks	• SpatiaLite creates a primary key column for each table it imports, and tries to name it pk by default. However, if your shapefile was previously exported from SpatiaLite, then it already has a pk column, and the conflict causes DB Manager to crash. There are two solutions to this:
	  o 1) delete the pk column from the shapefile before you import it
	  o 2) while using the import layer tool in DB Manager, specify the primary key to use `cat` as a primary key, since it is of type `integer` and all the values are unique.
	• When you import your work back into SpatiaLite, it is going to frustrate you and fail to match your custom projection to the one already in SpatiaLite. Don't worry…
	• Import your edited districts with default SRID settings, and then hack your data back to the correct projection:
	  o Discard the geometry column
	  o Use `SetSrid()` to reset all the geometry data
	  o If there's a mixture of single part and multi part features, cast them all to multi-part e.g.
	`update distGR0 set geom = casttomulti(geom)`
	  o Recover the geometry column
	  o E.g. :
	`select discardgeometrycolumn('distGR0','geom');`
	`update distGR0 set geom = setsrid(geom, 110000);`
	`select recovergeometrycolumn('distGR0', 'geom', 110000, 'MULTIPOLYGON')  **OR JUST POLYGON IF THEY'RE ALL POLYGONS`

Create a GitHub Account	• Please create a GitHub account and let me know what your user name is, so that I can invite you to `github.com/GIS4DEV`.

Shortcut for cleaning geometry and topology errors with GRASS	• QGIS now provides an **Advanced interface** to its **Processing Toolbox,** and this interface allows users to run operations from other open source software packages. Internally, the advanced interface (formerly a plugin called *sextante*) converts your data layer into the format used by the other open source software, runs an operation for you, and then converts the data back to whatever form you choose to save it in.
	  o If the toolbox is not seen, go to **Processing** -> **Toolbox**
	  o If the advanced tools are not seen, switch the toolbox from **Simplified Interface** to **Advanced Interface** with the menu at the bottom.
	• GRASS uses its own vector data model based on topologies, so if you run any GRASS tools on a shapefile in QGIS's advanced interface, the data will be converted from a shapefile to a GRASS topology, the GRASS function will then run, and then the output will be exported and saved.
	• Therefore, if you run the GRASS `v.extract` tool and set its advanced parameters of **v.in.ogr snap tolerance** and **v.in.ogr min area,** you'll get an output cleaned with snapping and minimum polygon size. The only disadvantage is that you don't get the layers for gaps and overlaps.

PreLab 5: How Dissolve Really Works

Purpose:

The goal of this pre lab is to demonstrate how Dissolves really work in GIS, and how to accomplish a dissolve in SpatiaLite.

References:

- SpatiaLite Functions: http://www.gaia-gis.it/gaia-sins/spatialite-sql-4.1.0.html especially "SQL functions that implement spatial operators" (vector analysis and overlays).

Procedure:

Prepare the Districts02 2002 data table to join to the Districts geometry table (complete)	• Copy `PreLab5.sqlite` from the splinter drive.
	• Start a new QGIS project and connect to the `PreLab5.sqlite` database.
	• The database has 4 data tables:
	  o `Zanz12` includes attribute data from the 2002 and 2012 censuses
	  o `ZanzDistricts` includes geometry data for districts for the 2012 census
	  o `ZanzRegions` includes geometry data for regions for the 2012 census

A simple GROUP BY clause	• The GROUP BY clause will aggregate all the rows in a selection containing identical values for any columns specified by the GROUP BY clause. For example: **SELECT *** **FROM** Zanz12 **GROUP BY** Dcode • This results in only two rows, since there were only two unique values in the `Dcode` column. The `Dcode` column shows all the unique values in `Dcode`, while the other columns are uselessly showing what appear to be the last values for each group (where one group has `Dcode = 1` and the other group has `Dcode = 2`).

GROUP BY clause with aggregate functions to summarize the group's values

- It's much more useful to use aggregate functions to summarize numerical columns, e.g.:

```
SELECT Dcode, count(), sum(pop02), min(pop02),
max(pop02)
FROM Zanz12
GROUP BY Dcode
```

  - There are still two rows, but now `count()` shows you how many districts were aggregated to each group, and `sum()`, `min()`, and `max()` show you the sum, minimum, and maximum of population for the districts in each group.

Dcode	count()	sum(pop02)	min(pop02)	max(pop02)
1	5	515800	62391	*184204*
2	5	465954	31853	*205870*

  - Do the results make sense? Sort Zanz12 by Dcode and pop02 to see:

```
SELECT Dcode, Dname, pop02
FROM Zanz12
ORDER BY Dcode, pop02
```

Dcode	Dname	pop02
1	Kati District	62391
1	Chake Chake District	82998
1	Kaskazini A District	84147
1	Wete District	102060
1	Magharibi District	*184204*
2	Kusini District	31853
2	Kaskazini B District	52492
2	Micheweni District	83266
2	Mkoani District	92473
2	Mjini District	*205870*

Using meaningful columns for Grouping

- The previous example has no real-world application, since there were 5 regions and each had a district with dcode = 1 and with dcode = 2. It's be more useful to group by Rname in order to summarize the districts of a region:

```
SELECT Rname, count(), sum(pop02), min(pop02),
max(pop02)
FROM Zanz12
GROUP BY Rname
```

Rname	count()	sum(pop02)	min(pop02)	max(pop02)
Kaskazini Pemba	2	**185326**	83266	*102060*
Kaskazini Unguja	2	**136639**	52492	*84147*
Kusini Pemba	2	175471	82998	92473
Kusini Unguja	2	94244	31853	62391
Mjini Magharibi Unguja	2	390074	184204	205870

  - Do these results make sense? Zanz12 is already sorted by Rname, so check the first region, Kaskazini Pemba. It had two districts, Wete and Micheweni. In 2002, Wete had 102060 people, and Micheweni had *83266*. The sum of these is **185326.**
  - Likewise, Kaskazini Unguja had two creatively named districts: A and B. A had *84147* people and B had 52492 people, for a total of **136639**.
- You could also group by Rcode and achieve the same results. You don't necessarily have to include a grouping column in the output.

Using UNION to combine geometries

- The classic explanation of *dissolve* is that it erases boundaries between polygons sharing the same values for the dissolve fields. This is true, but it's an oversimplification.
- In truth there is a *union* going on behind the scenes, whereby points can be combined into multipoint features, lines can be combined into polyline features, and polygons can be combined into multipolygons (and erase the boundaries between adjacent polygons). Think of *union* as an aggregate function that takes the *sum* of all the geometries in a group.

- The SpatiaLite function for *union* is `gunion()`. Try a `gunion()` on all the districts:

```
SELECT gunion(geom), pk_0,
count()
FROM ZanzDistricts
```

  - If you load the results as a new layer, you'll see that you have one multipolygon feature encompassing both islands, which were formerly 10 different districts (see figure at right).
  - There are some cracks in these polygons, since this data was not cleaned with GRASS yet and it still contains topological errors.

Prepare a table with all the necessary columns for *dissolve*

- If our goal is to dissolve districts into regions with their population, then we'll first need a table that contains:
  - An integer primary key
  - District geometries
  - District populations in 2002 and 2012
  - A *dissolve field* indicating which region each district should be grouped into
- You'll have to join columns from Zanz12 to ZanzDistricts in order to compile all this information into one table. Preview such a join with a SELECT statement:

```
SELECT ZanzDistricts.pk AS pk, ZanzDistricts.geom AS
geom, Zanz12.Rname AS Rname, Zanz12.Dname AS Dname,
Zanz12.pop02 AS pop02, Zanz12.pop12 AS pop12 FROM
ZanzDistricts LEFT OUTER JOIN Zanz12
ON ZanzDistricts.district_n = Zanz12.joinDist
```

- The results appear like this:

pk	geom	Rname	Dname	pop02	pop12
1		Kusini Pemba	Chake Chake District	82998	97249
2		Kaskazini Unguja	Kaskazini A District	84147	105780
3		Kusini Unguja	Kati District	62391	76346
4		Mjini Magharibi Unguja	Magharibi District	184204	370645
5		Kaskazini Pemba	Wete District	102060	107916
6		Kaskazini Unguja	Kaskazini B District	52492	81675
7		Kusini Unguja	Kusini District	31853	39242
8		Kaskazini Pemba	Micheweni District	83266	103816
9		Mjini Magharibi Unguja	Mjini District	205870	223033
10		Kusini Pemba	Mkoani District	92473	97867

Use GROUP BY to aggregate rows together into groups; combined with aggregate functions to summarize attributes of groups and UNION to summarize geometries of groups

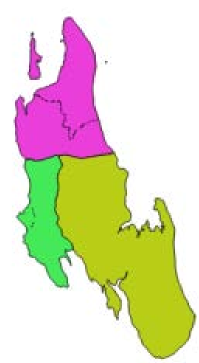
- Now add a GROUP BY clause to group districts by their Region Rname. Use the sum() aggregate function to sum the pop02 and pop12 values of the districts in each region. Use the gunion() spatial function to aggregate the geometries of the districts in each region:

```
SELECT ZanzDistricts.pk AS pk,
gunion(ZanzDistricts.geom) AS geom, Zanz12.Rname AS
Rname, sum(Zanz12.pop02) AS pop02, sum(Zanz12.pop12)
AS pop12
FROM ZanzDistricts LEFT OUTER JOIN Zanz12
ON ZanzDistricts.district_n = Zanz12.joinDist
GROUP BY Rname
```

- If you add the results to the map, the attribute table will look like this:

pk	Rname	pop02	pop12
8	Kaskazini Pemba	185326	211732
6	Kaskazini Unguja	136639	187455
10	Kusini Pemba	175471	195116
7	Kusini Unguja	94244	115588
9	Mjini Magharibi Unguja	390074	593678

- If you symbolize by categories and assign each region a random color, the resulting layer should look like the figure to the right. Districts have been aggregated into their respective regions.



Type-cast the gunion() results to uniform multipolygons	• One inconvenience in using `gunion()` to aggregate polygons is that it creates a `POLYGON` if all the input polygons were adjacent to each other, or it creates a `MULTIPOLYGON`, if any of the input polygons were disconnected (e.g. islands). • Add an `astext()` function to the results of `gunion()` to see this problem:

```
SELECT ZanzDistricts.pk AS pk,
astext(gunion(ZanzDistricts.geom)) AS geom,
Zanz12.Rname AS Rname, sum(Zanz12.pop02) AS pop02,
sum(Zanz12.pop12) AS pop12
FROM ZanzDistricts LEFT OUTER JOIN Zanz12
ON ZanzDistricts.district_n = Zanz12.joinDist
GROUP BY Rname
```

- This can be solved by casting all the `gunion()` results to `MULTIPOLYGONS` with a `casttomulti()` function. Also remove the `astext()` function!

```
SELECT ZanzDistricts.pk AS pk,
casttomulti(gunion(ZanzDistricts.geom)) AS geom,
Zanz12.Rname AS Rname, sum(Zanz12.pop02) AS pop02,
sum(Zanz12.pop12) AS pop12
FROM ZanzDistricts LEFT OUTER JOIN Zanz12
ON ZanzDistricts.district_n = Zanz12.joinDist
GROUP BY Rname
```

Create a new table to store your dissolve	• The selection is ready to be made permanent with a CREATE TABLE statement:

```
CREATE TABLE DissolveRegions AS
SELECT ZanzDistricts.pk AS pk,
casttomulti(gunion(ZanzDistricts.geom)) AS geom,
Zanz12.Rname AS Rname, sum(Zanz12.pop02) AS pop02,
sum(Zanz12.pop12) AS pop12
FROM ZanzDistricts LEFT OUTER JOIN Zanz12
ON ZanzDistricts.district_n = Zanz12.joinDist
GROUP BY Rname
```

- Register the geometry columns, and your dissolved Regions will be fully functional!

```
SELECT recovergeometrycolumn('DissolveRegions',
'geom', 110000, 'MULTIPOLYGON')
```

Lab 5: Change in Population over Time

Purpose:

This week's challenge is to calculate total change and percent change of population for the Districts of Tanzania. In order to do so, you'll have to investigate and record all of the district boundary changes, use a spatial join to determine which districts split, and learn how to dissolve and summarize fields using SQL.

References:

- Changes in Tanzania's Districts: http://www.statoids.com/ytz.html
- SpatiaLite Functions: http://www.gaia-gis.it/gaia-sins/spatialite-sql-4.1.0.html especially "sql functions that test spatial relationships" (spatial joins) and "SQL functions that implement spatial operators" (vector analysis and overlays).

Deliverables:

- Create two maps of percent population change in Tanzania. One map should use the boundaries of 2002 districts, and the other should use the boundaries of the 2012 regions.

Procedure:

`Prepare the Districts02 2002 data table to join to the Districts geometry table (complete)`	• Download `TZlab5.sqlite` from the splinter drive and connect to it in a new QGIS project. This database has already completed all the sections marked `(complete)`. • If you haven't yet imported your data from Lab 4 into the database, do so now, following advice at https://github.com/GIS4DEV/Q-and-A/wiki#why-wont-my-shapefile-layer-import-from-qgis-to-spatialite-using-db-manager • Add a `Djoin` column to `Districts02` to and calculate and edit its values to match the `district_n` column of the `districts` table wherever possible. For this, follow similar procedures as PreLab 3: Cleaning Data to Make Joins Work. • This work only gets you so far, because many countries change the boundaries of enumeration areas between censuses, often subdividing some units to form smaller units. In order to map change between two time periods, you must have geographic containers, or geometries, that match exactly. This will require you to aggregate any areas that have been subdivided back to their original whole. • In the case of Tanzania, several of the 2002 districts were subdivided before the 2012 census. So, we have to aggregate some 2012 districts back into their 2002 parents. To do so, you'll need to do additional research to find ancillary data and information on these changes.

Set up the database with a change table and views to track the changes from 2002 to 2012 (complete)	• Create a new table that will store the relationship between 2002 and 2012 districts.
	`CREATE TABLE Dchange`
	`AS SELECT district_n AS Dname02, district_n AS Dname12`
	`FROM Districts`
	
	• Create two views to preview how mapping 2002 Districts will go:
	`CREATE VIEW LOJdchange_districts02`
	`AS SELECT Dname02, Dname12, Djoin`
	`FROM Dchange LEFT OUTER JOIN Districts02`
	`ON Dname02 = Djoin`
	`ORDER BY Djoin, Dname02`
	
	`CREATE VIEW LOJdistricts02_Districts`
	`AS SELECT Dname, Djoin, district_n`
	`FROM Districts02 LEFT OUTER JOIN Districts`
	`ON Djoin = district_n`
	`ORDER BY district_n`

Edit the Dchange table to translate from 2012 Districts to 2002 Districts (complete)	• Add Dchange to the map canvas and open the attribute table for editing o Edit Dname02 to reflect changes made after 2002 according to Statoids: http://www.statoids.com/ytz.html. o For each District in 2012 that has been split from a 2002 district, edit it's 2002 district value to reflect the district it has been split off from. o Mkinga was split from Muheza, therefore I'll find the Mkinga row, and change its Dist02 value to Muheza. o If any urban district has *null* results in the LOJdchange_districts02 preview, it was probably split off from a greater urban/rural counterpart in 2002. Edit its Dname02 to match its greater 2002 District counterpart if the results are in the join preview o The exception to the previous pattern is Arusha, where the 2012 Arusha Urban was simply Arusha in 2002, the 2012 Arusha was split from the 2002 Arumeru, and the 2012 Meru was also split from the 2002 Arumeru. • There are still eight 2012 districts for which we haven't been able to match to a 2002 district! The original census data can tell us which Regions they are in, but how can we figure out which Districts they came from?

Prepare the 2002 Wards ancillary data for a spatial join to the 2012 district geometries	• Add `TZwards.shp` to the database and name the table `Wards02`
	  o `Wards02` provides ancillary data at a lower level of the census hierarchy. You'll need to find the center points (centroids) of each ward to determine which 2012 district each 2002 ward falls within. Centroids are better for comparison than boundaries, because of digitization errors and spatial scale discrepancies between layers.
	• Add a new geometry column to `Wards02` to store centroids:
	`SELECT addgeometrycolumn('Wards02', 'Center', 110000, 'POINT')`
	• Calculate the centroids and then transform them to the custom Tanzania Lambert Azimuthal Equal Area projection. You can do this all at once by using the results of a `centroid()` function as the input parameter for the `transform()` function:
	`UPDATE Wards02 SET Center = transform(centroid(geom),110000)`

Use a spatial join to append a 2012 district name to each of the 2002 wards, according to which 2012 district contains the 2002 ward.	• Let's preview how a spatial join using the `contains()` relationship will work. In this case, we'll join rows if the `Districts.geom` polygons contain the `Wards02.Center` points.
	`SELECT *`
	`FROM districts LEFT OUTER JOIN Wards02 ON contains(Districts.geom, Wards02.Center)`
	• That seemed to work, but the results could be presented in a more useful way if we select only the important rows and if we count how many 2002 wards of a given 2002 district name fell within each 2012 district. This will require adding a group by clause to find each unique combination of 2002 district names and 2012 district names, and a `count()` aggregate function to count the matching pairs.
	`SELECT Districts.district_n AS District2012, Wards02.region AS WardRegion, Wards02.district AS WardDist, count()`
	`FROM districts LEFT OUTER JOIN Wards02 ON contains(Districts.geom, Wards02.Center)`
	`GROUP BY District2012, WardDist`
	`ORDER BY District2012`
	• Look up all 2012 districts with unknown 2002 counterparts (i.e. rows that have *NULL* results when joining `Districts02` census data to the `Dchange` table in the `LOJdchange_districts02` view) and enter in their 2002 district name, matching spelling and formatting of the `Districts02.djoin` column.
	• Only `Dodoma` and `Arumeru` should remain unmatched, because although both a `Dodoma` and an `Arumeru` district existed in 2002, there are no districts by that name in 2012.
	• Remember that in our database, the 2012 rural districts have dropped the `Rural` portion of their name.

Using SQL and SpatiaLite functions to DISSOLVE

- The Dchange table should now contain a full set of relations between 2002 districts and 2012 districts. If you join the Dchange table to the Districts table, it can be used to group the 2012 districts into their corresponding 2002 districts, using a GROUP BY clause.

```
SELECT pk, Dname02, sum(area_sqkm)
FROM Districts LEFT OUTER JOIN Dchange
ON district_n = Dname12
GROUP BY Dname02
```

- A *Dissolve* is really a database GROUP BY that also includes a spatial aggregation, *Union*, using the gunion() function. Try this with the Districts table, summarizing the 2012 district boundaries by their 2002 district parents:

```
SELECT pk, gunion(geom), Dname02, sum(area_sqkm)
FROM Districts LEFT OUTER JOIN Dchange
ON district_n = Dname12
GROUP BY Dname02
```

- There may be a slight problem of geometry type mismatches from this union though. Check the astext() results:

```
SELECT pk, astext(gunion(geom)), Dname02,
sum(area_sqkm)
FROM Districts LEFT OUTER JOIN Dchange
ON district_n = Dname12
GROUP BY Dname02
```

- The results are a mixture of polygons and multipolygons! You can preview a fix for this by casting to multi-part features:

```
SELECT pk, astext(casttomulti(gunion(geom))) AS geom,
Dname02, sum(area_sqkm) AS area_sqkm
FROM Districts LEFT OUTER JOIN Dchange
ON district_n = Dname12
GROUP BY Dname02
```

- You can now make this dissolve permanent in a new table. Remove the astext() function so that you get valid geometries:

```
CREATE TABLE Districts02g
AS select pk, casttomulti(gunion(geom)) AS geom,
Dname02, sum(area_sqkm) AS area_sqkm
FROM Districts LEFT OUTER JOIN Dchange
ON district_n = Dname12
GROUP BY Dname02
```

  - In this query I have taken care to re-name all the columns to good database column names, and you should do the same any time you create a new table with a SELECT query.

- Now recover the geometry column for the Districts02g table. The srid is 110000

Test how well the dissolve worked	• Can you use a spatial join to find all of the 2002 Wards that have a *different* district name than the results of our dissolve to 2002 districts? • This question will require a spatial join of `districts02g` and `wards02`, finding where the `Districts02g` table's `Dname02` is `NOT LIKE` the `Wards02` table's `district` name: `SELECT Wards02.pk AS pk, Wards02.geom AS geom, Districts02g.Dname02 AS District2012, Wards02.district AS Ward02District FROM Wards02 LEFT OUTER JOIN Districts02g ON contains(Districts02g.geom, Wards02.Center) WHERE WardDist NOT LIKE District2012 || '%'` • If you load the results of this query to the map and overlay it on the `districts02g` districts, you'll immediately see any wards with potential discrepancies between 2002 and 2012. o Most of the discrepancies seem to be isolated errors in digitizing ward boundaries (this shapefile contains a *lot* of error), or discrepancies in spelling or language (e.g. Missungwi vs. Misungwi and the Swahili names vs. English names on the Zanzibar Islands). o Two discrepancies are of greater concern: the urban districts of Arusha and of Lindi both seem to have expanded from 2002 to 2012, growing to encompass some formerly rural wards. There are two potential solutions to these problems: o A) aggregate the urban districts and their surrounding rural districts into one polygon and sum the populations of each. B) research the populations of the wards that have changed from rural to urban and re-allocate those population figures accordingly. From the 2002 figures, subtract the population of those wards from the rural district and add them to the urban district.

Create a new pop02 column in Districts02g and update it with data from Districts02	• Add a column to the `Districts02g` table, `pop02`, of type `INTEGER`
	• You want to copy the `pop02` data from `Districts02` to `districts02g`, so let's preview what a join of those tables would look like:
	`SELECT * FROM Districts02g LEFT OUTER JOIN Districts02 ON Districts02.Djoin = Districts02g.Dname02`
	• In the query above, and in the next few queries, I've coded some elements in color as follows:
	o **Red** indicates the target table
	o **Blue** indicates the join table
	o **Purple** indicates the join condition
	• A SELECT is only temporary, whereas UPDATE will permanently transfer the data, so re-arrange the SELECT into an UPDATE:
	`UPDATE Districts02g SET District02g.pop02 = (SELECT Districts02.pop02 FROM Districts02 WHERE Districts02.Djoin = Districts02g.Dname02)`

Create a new pop12 column in Districts02g and update it with data from Districts02 and Districts12

- Add a column to the districts02g table, pop12, of type INTEGER
- Copying pop12 from Districts12 to Districts02g is tricky because it has to include a summation of any 2012 district that was part of the 2002 district. The results of that summation will be considered the join table. First, preview the aggregation of 2012 districts into 2002 districts:

```
SELECT Dname02, sum(Districts12.pop12)
FROM Districts12 LEFT OUTER JOIN Dchange
ON Djoin = Dname12
GROUP BY Dname02
```

- Second, preview a join of that aggregate 2012 data to the Districts02g table:

```
SELECT *
FROM Districts02g LEFT OUTER JOIN
(SELECT Dname02, sum(Districts12.pop12) FROM
Districts12 LEFT OUTER JOIN Dchange
ON Djoin = Dname12
GROUP BY Dname02) AS d12
ON Districts02g.Dname02 = d12.Dname02
```

  - This query has replaced a simple table with a whole SELECT query (in gray) and named those results d12

- Now use UPDATE to copy the pop12 data from Districts12 to Districts02g:

```
UPDATE Districts02g SET pop12 =
(SELECT sum(Districts12.pop12)
FROM Districts12 LEFT OUTER JOIN Dchange
ON Districts12.Djoin = Dchange.Dname12
WHERE Districts02g.Dname02 = Dchange.Dname02
GROUP BY Dname02)
```

Calculate total population change and percent population change

- The data is ready to calculate the total change:

  Total Change = $Time_2 - Time_1$

- And to calculate percent change*****:

  Percent Change = $((Time_2 - Time_1) / Time_1) * 100$

  - *Remember that percent change will be of the REAL data type, but your inputs here are INTEGERs. See problems with integer division below …

- In order to update the database's metadata for your table, use the UpdateLayerStatistics() function:

```
SELECT UpdateLayerStatistics('Districts02g')
```

Problems with Integer Division

- SQLite will use *integer* division if the numerator and denominator are both integers, and this will result in a truncated integer. This is not useful when the possible answers are all between 0 and 1!
- If you want to calculate the percentage of males using 'male' and 'total' in the tz_wards_2002 table, the following statement will return all 0's:

```
SELECT male / total
FROM wards02
```

- However, if you cast one of the variables to a double data format (using decimals), it will work:

```
SELECT cast(male AS REAL)/total
FROM wards02
```

- Or, if you include any real number in the formula, it will use real division:

```
SELECT male * 1.0 / total
FROM wards02
```

The next challenges:

- Can you confirm that the total population of Tanzania before aggregating to 2002 districts matches the total population after aggregating, for both census years?
- Can you check internal consistency between regions and districts, checking if the population of each region equals the sum of the populations of districts in that region?
  - Hint: use a `SELECT` with a `GROUP BY` clause and a `sum()` aggregate function to check if, when you add all the districts in each region together, the sum of population for the region is equal to the population of the region given by your data sources.
- Please make a map of percent population change from 2002 to 2012 according to the 2002 district boundaries.
- Please make a map of percent population change from 2002 to 2012 according to the 2002 region boundaries.
  - The 2002 Biharamulo district in the 2002 Kagera Region was divided into the 2012 Chato and Biharamulo districts. Chato has become part of the new Geita Region, while Biharamulo remains in Kagera Region. Therefore a 2012 Regions map will be accurate everywhere except Kagera and Geita.
  - In order to make this map, you need to dissolve districts into regions. To do so, you'll need a column in the districts02g table that indicates which region each district should be grouped into. Therefore, add a column for `Rname` of type `TEXT` and then use either a *spatial join* to the 2012 `Regions` geometry table ***-or-*** a normal attribute *join* to the `Districts02` table (which contains an `Rname` column).

Lab 6: Finding, Documenting, Georeferencing Satellite Data

Purpose:

This week's challenge is to find satellite images, document their metadata, and georeference a historic aerial photo.

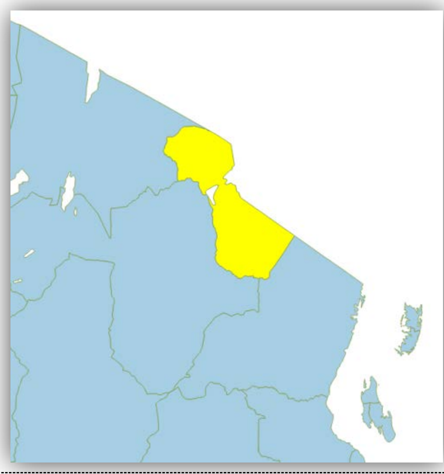
References:

- LANDSAT Science (NASA): http://landsat.gsfc.nasa.gov/
- LANDSAT Mission (USGS): http://landsat.usgs.gov/
- LANDSAT Processing Details (USGS): http://landsat.usgs.gov/Landsat_Processing_Details.php
- LANDSAT Data Dictionary (USGS): https://lta.cr.usgs.gov/landsat_dictionary.html
- GLOVIS: http://glovis.usgs.gov/

Deliverables:

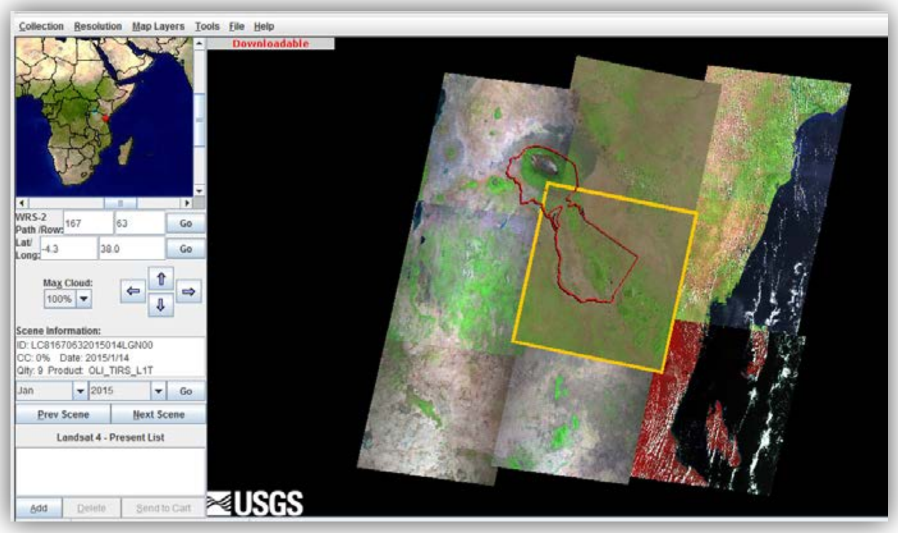
- Metadata documentation for 2002 and 2012 satellite images.
- Check off map with georeferenced aerial photo.

Procedure:

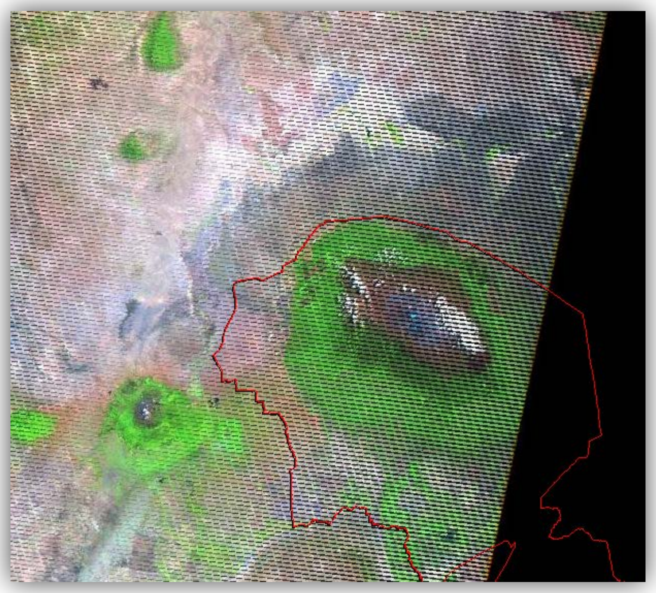
Before searching for satellite imagery, know your Area of Interest	• Open any version of Tanzanian regions. • Select Kilimanjaro Region • Right-click the Regions layer and *save as* • Choose to *save only selected features* • Save a shapefile in the WGS 1984 coordinate system (EPSG 4326) (web interfaces for downloading data use geographic coordinate systems)	

Search for available images, first finding the correct path and row

- Go to the LANDSAT Search and Download Page and open the GLOVIS browser: http://glovis.usgs.gov/
- Load your area of interest by going to Map Layers -> Read Shapefile
- Select the shapefile you saved of Kilimanjaro Region.
- GLOVIS should zoom to the area of interest as shown below:



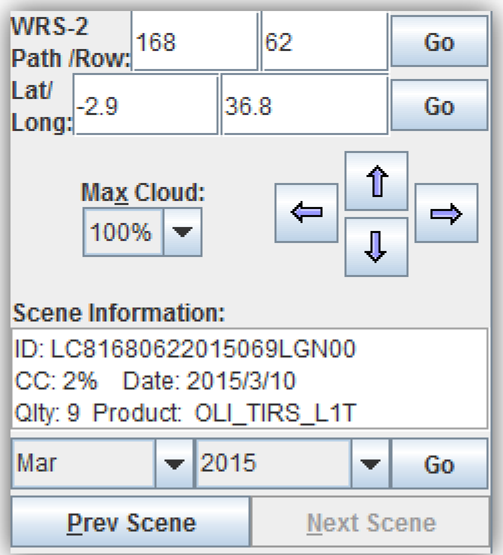
Go to *Resolution -> 240m* to zoom the image to one path and row.



- GLOVIS will zoom to an image like the one above, where the black lines are caused by the SLC (Scan Line Corrector) failure in Landsat 7, and a narrow path down the center of the image is error-free.
- Once you know the Path/Row location, you can start searching for image(s) at the best possible times.

Search for available images at your required location, focusing on time period

- Search results are limited to the selected *collection*. Go to *Collection* -> *LANDSAT* archive. *Landsat 4 – Present* should be selected, corresponding with all Thematic Mapper imagery, starting in 1982.
- With the Path/Row and Collection set, you can use the Prev Scene and Next Scene buttons to cycle through images for different times at the same location. You can also jump to a specified month and year, or exclude images with a maximum cloud cover percentage.



- Try to find the above scene, captured this weekend.
- ID refers to the scene's unique ID.
- CC indicates the percent cloud cover (March is the beginning of the long rain season).
- Date indicates the date the image was acquired.
- Qlty indicates the image quality, rated from 0 to 9 where 9 is best (based on spacecraft, sensor, and data storage conditions).
- Product indicates the sensor (ETM+) and image correction processing (L1G).

Searching for useful images	• Your task is to find images correlating as best as possible with the census data: years 2002 and 2012 in this case. You'll have to make tough decisions and compromises between using images from the same season, using high quality images, and using images that may not match the target year(s) exactly.
	• First, look at all the images available for 2012.
	• Once you find the best image(s) for 2012, use the *Add* button to add their scene IDs to the Scene List.
	• Once scenes are in the Scene List, the Scene List can be:
	  o Saved, by going to File -> Save All Scene Lists
	  o Metadata and preview images saved, by going to File -> Download Scene List Browse & MetaData (giving a .jpg of the scene and a .meta file that can be opened with notepad)
	  o Metadata or browse image previewed by going to Tools -> Scene List, selecting one scene, and clicking *Show Metadata* or *Show Browse*.
	  o *Sent to Cart*, wherein you'll have to choose to *order* the images or download *LandsatLook* (but this is just a natural color image, without the non-visible bands).
	  o *****Accessing a cart will require you to register, and orders may take up to a few days. Use a valid email address when you register, so that you can receive the download information!
	• Considering that to compare inter-annual change, images should be taken in the same seasons, what images would you choose to analyze land cover change from 2002 to 2012? Consider that we are most interested in land cover change on the inhabited slopes of Mt Kilimanjaro and surrounding plains.

Documenting Metadata	• Download the browse and metadata for your 2002 and 2012 images (either download as part of your scene list, or browse to the image and *download visible browse and metadata*)
	• In a Word Document, compile an image of the scenes and metadata for it, including:
	  o Satellite
	  o Sensor type(s)
	  o Scene ID
	  o Percent cloud cover
	  o Location (path and row)
	  o Acquisition date and time
	  o Image quality
	  o Spatial reference system (datum and projection)
	  o Correction: Has the image been radiometrically corrected, geometrically corrected, and/or orthographically corrected?
	  o Resolution of your image's satellite and sensor: a) spectral resolution, b) spatial resolution, c) radiometric resolution, d) temporal resolution

Georeferencing Images in QGIS

Georeferencing is the process by which you can transform images or grids into a projected coordinate system. It's most necessary with historical aerial photos or scanned maps, but remote sensing scientists also use it to ensure that all of their imagery lines up precisely. For this lab, you'll try georeferencing a historical photo from Mount Kilimanjaro in 1962.

- Open `0051markup.tif` with Windows Photo Viewer. This tif image of a 1962 aerial photograph has no geographic reference information. I have circled a few identifiable features in the image, which can be used as *control points* to georeference this image. Control points should be locations that are identifiable and relatively easy to precisely locate.
- Open QGIS and zoom in to Moshi District of Kilimanjaro Region.
- From `splinter/gg328/satellite/ikonos`, add `po_323693_pan_0000000.tif` to the QGIS map
- From `splinter/gg328/satellite/topographic,` add the 56_2 and 56_4 topo sheet `.tif` images to the map.
- Go to Raster -> Georeferencer -> Georeferencer
-  Once the Georeferencer window is open, open a raster in it: `0051.tif`.
- Always assign the raster the coordinate system of the project you are using to georeference. In this case, the Ikonos image is in `WGS 84 / UTM Zone 37S` (EPSG: `32737`). Double-check to see that the QGIS project is also using EPSG: `32737` for the `CRS.`
-  Start adding control points, first adding a point to the raster image, and secondly adding the matching coordinates from the map canvas.
- Once 6 or more points are registered, observe the *residuals* of each point. The residual refers to the remaining error after transforming the point. Low residuals indicate minimal error.
- Go to Settings -> Transformation Settings.
- Choose a Transformation Method:
  - **Polynomial 1** is ok if the data only needs to be rotated and scaled.
  - **Polynomial 2** allows a 2nd order equation for curvature on one axis.
  - **Polynomial 3** allows curvature on two axes, but can lead to 'overfitting' the transformation.
- Save an output name for your new image.
- Based on your transformation method choice, have the residuals changed?
- If one or more control points have particularly high residuals, they may be incorrectly digitized.
-  When you're satisfied with your control points and transformation method, start georeferencing.
- If QGIS asks you about the CRS again, keep the reference system consistent with the data you have (EPSG: `32737`) and see how it looks on the map.

View a variety of remote sensing and topographic data sources to evaluate change over time	• Look at change over time in this study area. Suppose that the general understanding of human-environment interactions in the area is that population growth and globalization since independence (1961) have led to deforestation of both the inhabited hillsides and the forest reserve on the higher slopes? Is there evidence in support of this? Against it? • The data available to you includes: o 1962 aerial photo you georeferenced o 1964 Topographic map of Mweka based on 1962 aerial photos o 1990 Topographic map of Moshi based on 1982 and 1983 aerial photos o 8.21.1964 Declassified spy satellite data o 2.17.1993 Landsat TM image from Landsat 5 o 2.2.2001 Ikonos image (blue, red, green, infrared, and panchromatic) o 12.22.2008 Worldview-1 panchromatic image o Modis data for all of Africa – a composite of data from multiple time periods, compiled into one cloud-free image. This was downloaded prior to 2009.

Lab 7: Visualizing Earth with Composites, Ratios, and Classes

Purpose:

This week's challenge is to visualize Landsat satellite images to visualize the Earth's surface. We'll first look at one band of electromagnetic energy at a time and practice modifying its *nodata* background settings and *stretch enhancements*. We will *stack* the bands representing red, green, and blue light into a *true-color RGB composite* image, and then make some common *false-color composites* to highlight different features in the landscape. We'll use some simple *band math*, specifically *band ratios*, to calculate an index of vegetation. Finally, we'll use an *unsupervised classification* method (clustering) to transform the continuous reflectance data from all the Landsat satellite bands into a set of discrete classes which we will then identify and label.

Data:

- Landsat 8: 2/5/2014
- Some topographic maps and high-resolution satellite images are available for reference in gg328/satellite. Please do not copy these images, but feel free to load them into QGIS straight from Splinter.
- Several research papers and maps are available for your reference, in the lab07/Kilimanjaro folder.

References:

- SAGA GIS: http://www.saga-gis.org/ (System for Automated Geoscientific Analysis)
- 7-zip file compression software: http://www.7-zip.org/
- Landsat band designations: http://landsat.usgs.gov/band_designations_landsat_satellites.php
- Landsat 7 compositor: http://compositor.gsfc.nasa.gov/pdfs/Landsat_7_Compositor.pdf
- Band combinations for Landsat 8: http://blogs.esri.com/esri/arcgis/2013/07/24/band-combinations-for-landsat-8/
- Landsat 8 bands: http://landsat.gsfc.nasa.gov/?page_id=5377
- Normalized Difference and Band Ratios, including NDVI: http://geology.wlu.edu/harbor/geol260/lecture_notes/Notes_rs_ratios.html

Deliverables:

- RGB composites of natural color, urban areas, and vegetation
- Normalized Difference Vegetation Index
- Classified map of land cover based on unsupervised cluster analysis

Procedure:

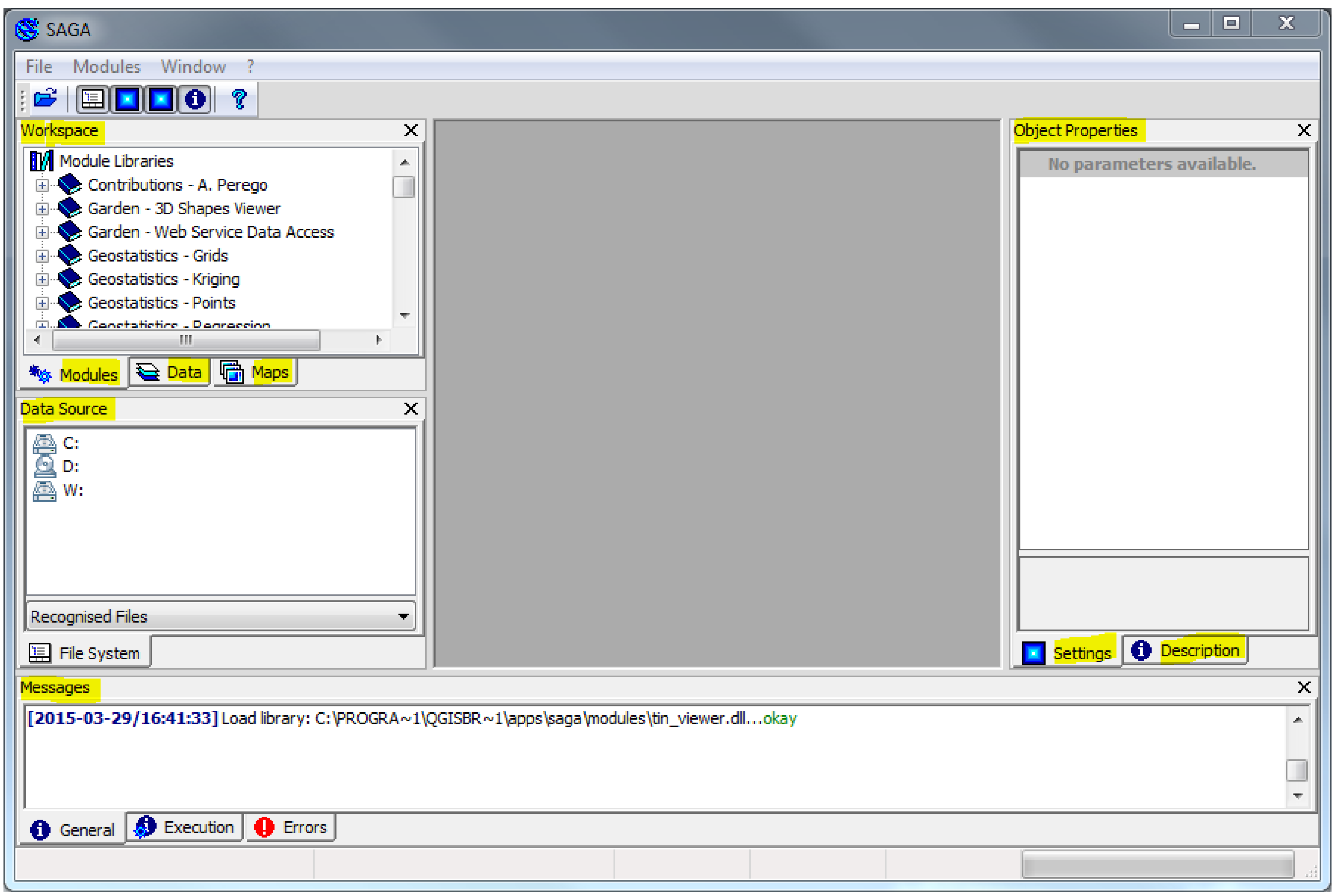
<table>
<tr>
<td><code>Unzip Landsat image data using 7-zip</code></td>
<td>

- Landsat images download as `.tar.gz` files. This is a `tar` archive compressed inside a `gz` archive. Use the **7-zip** open-source file compression software to *first* open the `tar` file from inside the `gz` file, and *then* extract the contents of the `tar.`
    - Open **7-zip** and open `LC81680622014034LGN00.tar.gz`
    - Then open `LC81680622014034LGN00.tar`
    - **Select all** the files and **Extract**. I like to add a folder to the extract location to contain all the images, e.g. `L82014`

</td>
</tr>
</table>

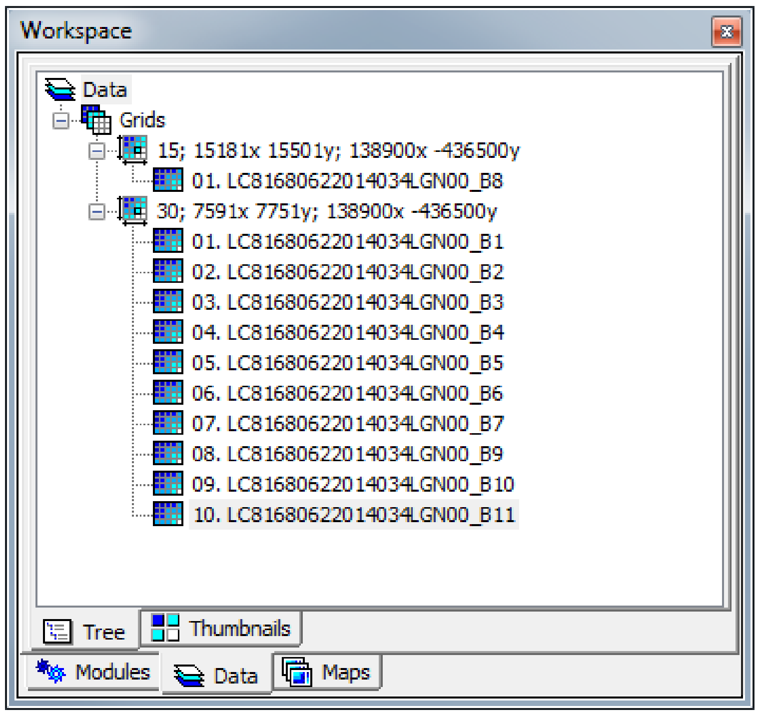
Open SAGA GIS (2.0.8)

- Go to the Start menu, browse to QGIS Brighton, and start SAGA GIS version 2.0.8.
- The default view launches with four **windows** shown below:
  - The **Workspace** window has three tabs. **Modules** are analogous to tools or processes. **Data** lists all raster grids, vector data, and data tables loaded for the current project. It is analogous to the Layers panel in QGIS. **Maps** lists all the map representations of data loaded for the current project. These are analogous to map compositions in QGIS
  - The **Data Source** window is analogous to the Browser panel in QGIS.
  - The **Messages** window is analogous to the results windows and status bars of QGIS.
  - The **Object Properties** window, with its **settings** and **description** tabs, serves many purposes. For data layers, it is analogous to the properties dialogue and metadata windows. For modules, it is analogous to a tool's graphical user interface and help documentation.
- You can add or remove **Windows** with the **Window** menu.



Load data, and save the project

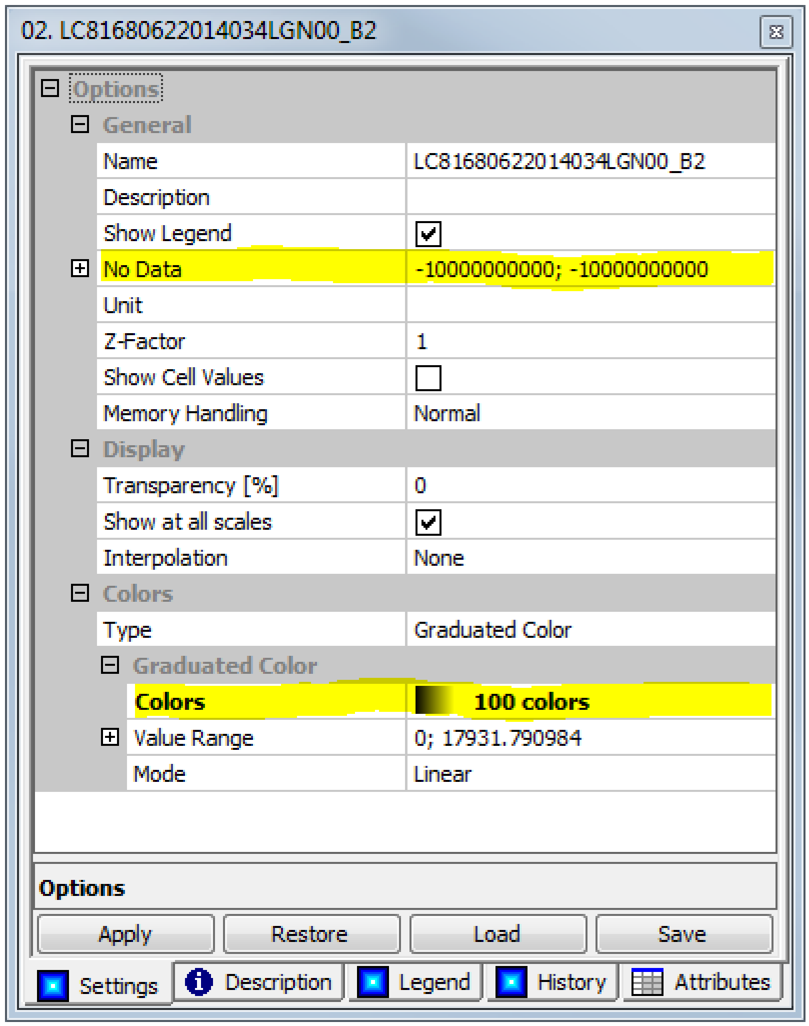
- In the File System tab, browse to your Landsat 8 image from 2014.
- Double-click each of the `.tif` files for Landsat 8 to load them, except for BQA (a quality assessment image). Be patient, SAGA converts them as they're added...
  - You can also drag and drop files onto the SAGA window to load them.
- In the **Workspace**'s **Data** tab, each grid is grouped with grids according to their grid systems (the coordinate system, cell size, and extent).
  - This is significant: SAGA will only run many algorithms on images with identical grid systems.
  - Notice that B8 is added to a separate grid, because it's the *panchromatic* band and has a higher spatial resolution than the other bands. The thermal bands fit in the 30m grid because they have been re-sampled to 30m resolution (from 100m).



- Once images `B1` through `B10` have all been added, go to **File** -> **Project** -> **Save Project As**, and save in a new folder (I called mine `lab7saga`)
  - Any time you save or close a project, a *"close and save modified data sets…"* dialogue asks you to save any data that is not yet permanently saved. SAGA only keeps data in temporary memory until you save it.
  - Check **save all**, and **okay**. SAGA starts saving everything in the same folder as your project and in its own SAGA grid `(.sgrd)` format.

Set the NoData range and colors for grid layers

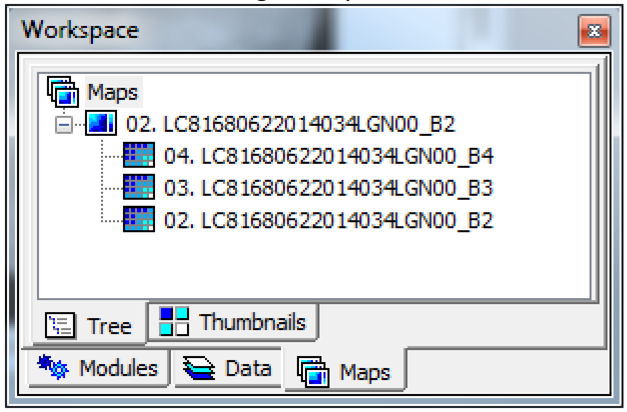
- Highlight any of your Landsat 8 grid layers
- Look at the **Object Properties** window, **Settings** tab, shown below.
- Under the Settings tab, you'll see that **No Data** is set to `-10000000000; -10000000000`.
- These Landsat images use `0` for **No Data**, therefore change the **No Data** range to `-10000000000; 0`, hit your keyboard's enter key, and **apply**.
    - o Set the *nodata* values for every image that you use.



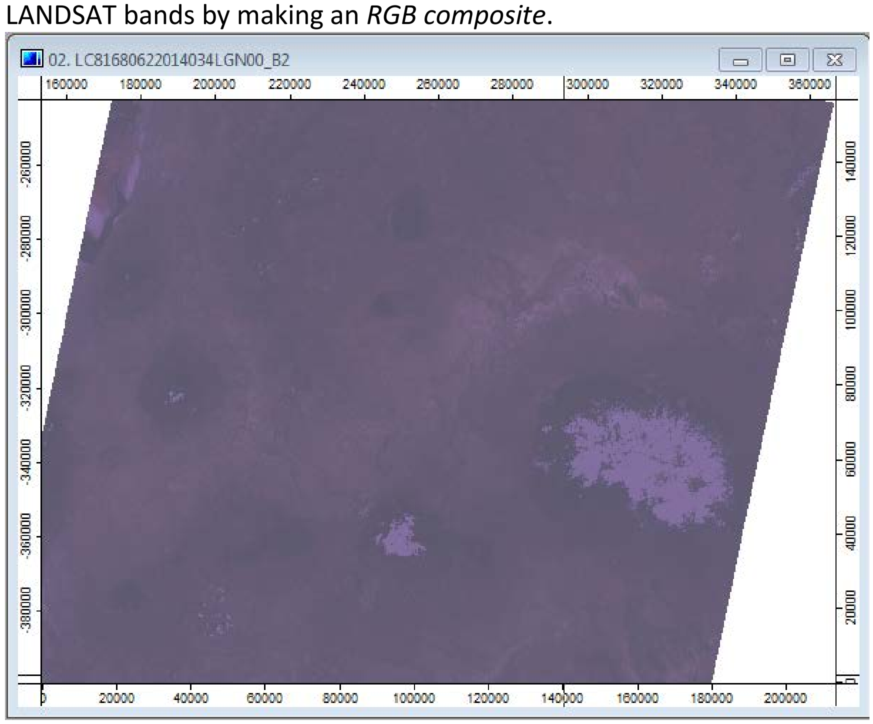
- Before we make a map, let's change the colors used by the blue, green, and red bands to their respective colors.
    - o Select the B2 band.
    - o Select the **Colors** row and use the ellipsis **…** button to select new colors
    - o Use the **Presets** button to select the `black > blue` preset.
    - o Also change the **transparency** to 66%
    - o Also set `B3` to green and `B4` to red

Make a true color map by displaying the blue, green, and red bands with their respective colors, and with transparency.

- Start composing a map by double-clicking the B2 grid (blue light) in the data tab.
- Double click the B3 and B4 grids (green and red light) as well. When SAGA asks for your map selection, add them to the same map as the B2 image.
- Switch to the **Workspace** window **Maps** tab, and you should see all three images under the same map.
- You can click and drag the layers to reorder them…

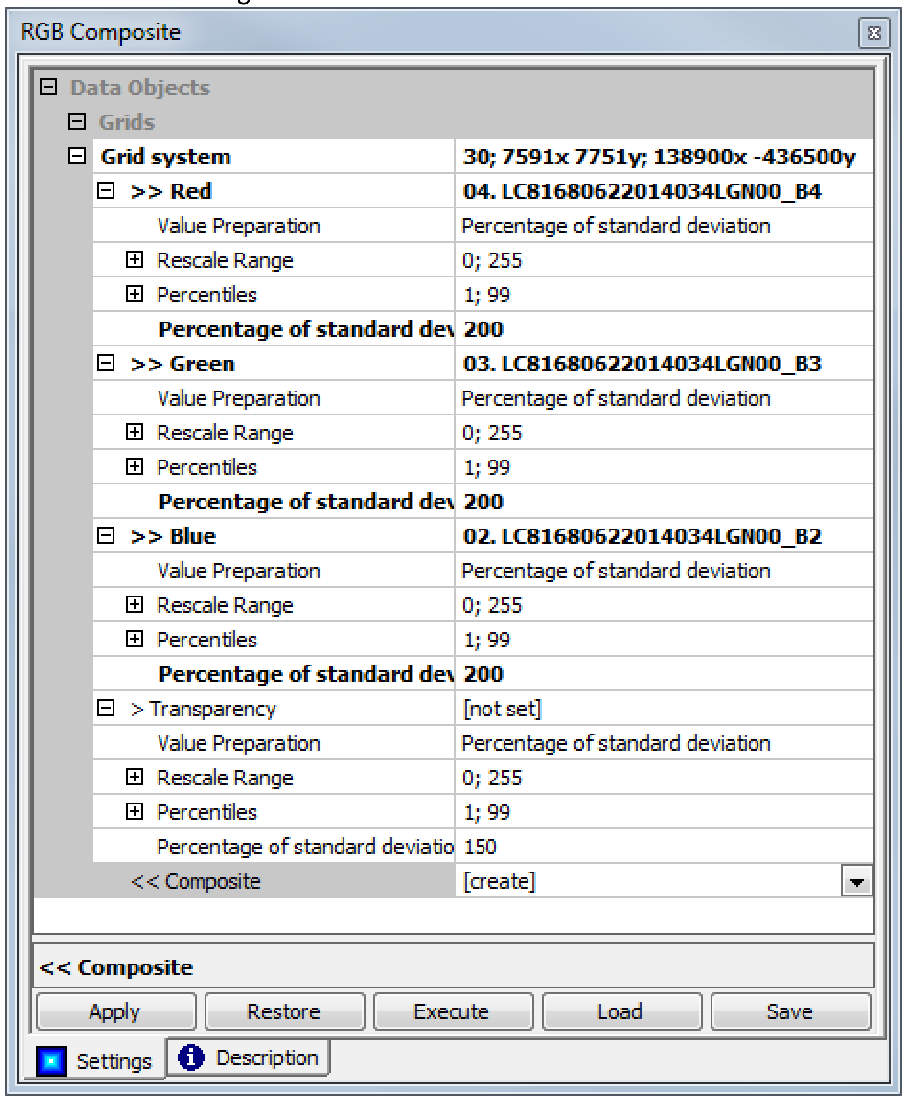


- If you set the NoData, colors, and transparency correctly in the previous section, you should have an approximation of a true color image as seen below. This map still doesn't look very good, but the purpose of making it was to illustrate the concept of combining multiple LANDSAT bands into a single map. You can make a much-improved combination of any three LANDSAT bands by making an *RGB composite*.

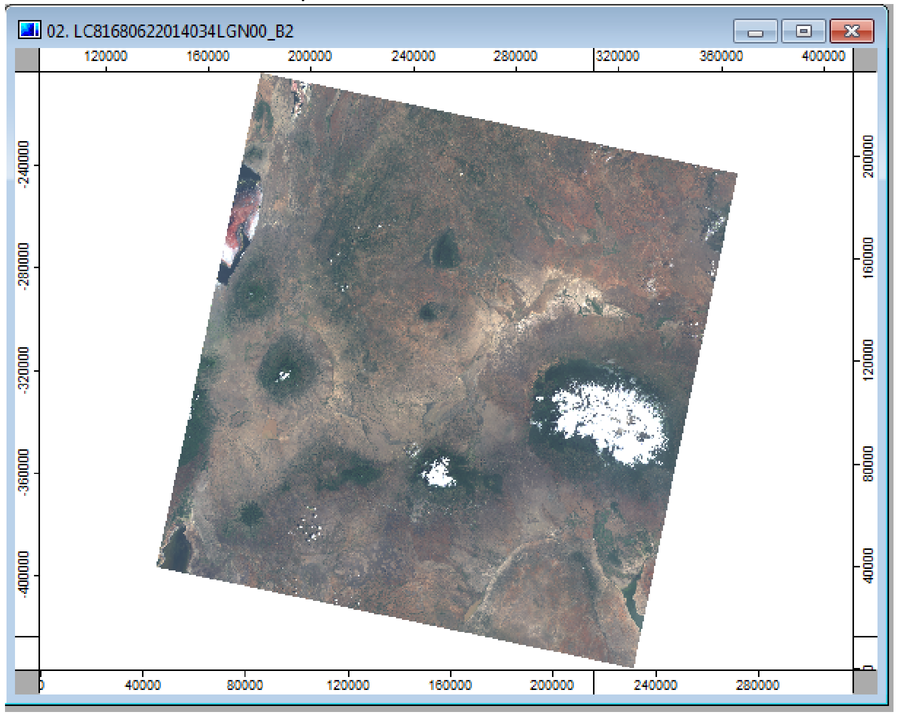


Make a natural-color RGB (red green blue) composite out of individual image bands

- Switch to the Modules tab, and find **Grid - Visualisation** -> **RGB Composite**
- Set the **Grid system** to your `30m` grids
- Set the **Red** to B4, **Green** to B3, and **Blue** to B2
- Set all the Value Preparation methods to Percentage of standard deviation
- Set all the Percentage of standard deviation values to `200`



- **Execute**, and the Composite image will be added to the Data tab. Double-click to add it to your map.
- The module results are only temporary-- right-click the composite layer and **save as** to save a permanent grid, `TrueColor.sgrd.`
- You may also change the **Name** of the grid in its settings to `TrueColor`

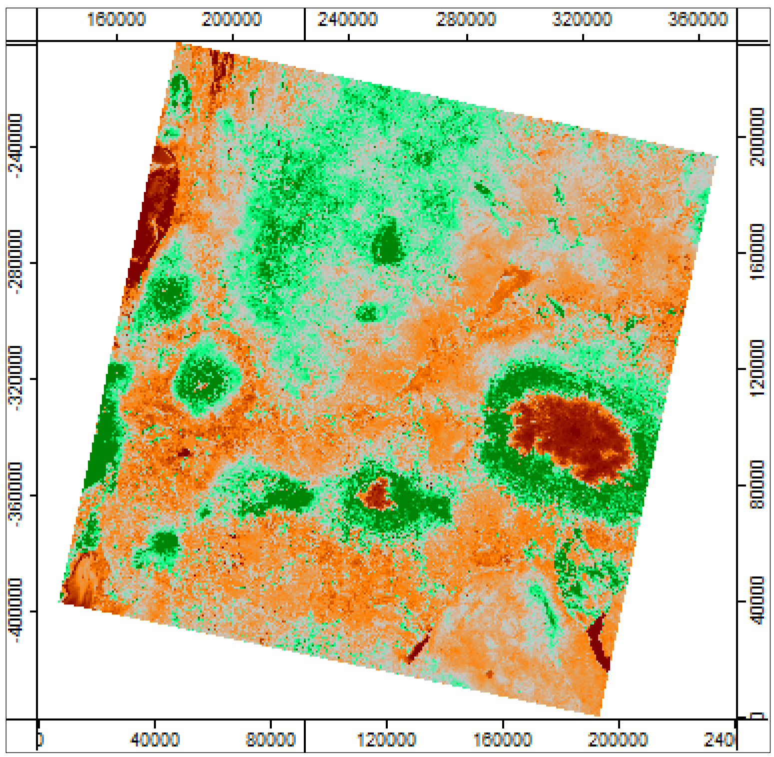
Notes on RGB Composites	• Your true color RGB composite should look like this:  • Percentage of standard deviation of 200 means that the composite image will show 2 standard deviations of data, and thereafter it will truncate higher values to white and lower values to black. If you want to increase contrast for the majority of data, then decrease the percentage. If you need to see more contrast in bright or dark values, then increase the percentage, or choose the *Rescale to 0 - 255* value preparation method. • There is a tempting short-cut to see your data in RGB without running this tool, but ***beware***: it does not work for Landsat 8, most likely because its radiometric resolution is higher than the 256 levels used to display the image. SAGA will crash if you try. The trick is to change the colors option of an image's object properties to RGB Overlay. Only try this with Landsat 7 or earlier.
Make your own false-color composites to visualize vegetation and urban areas.	• Use the `Landsat 7 Compositor and Landsat 8 Band Combinations` references for inspiration in creating new false-color composites. Remember that band numbers change between Landsat missions, so be sure to match the correct bands. • Make at least two false color composites, one for vegetation and one for urban areas. • When you re-run the RGB Composites module, avoid over-writing your existing composites by changing the **Composite** setting at the very bottom of the module to `[create]`. o In SAGA, module output parameters are named (e.g. '**Composite**') without clearly labeling them 'output composite'. You can either specify a file name, or use the default `[create]` to create a temporary output which you can save later.

`Arrange and synchronize multiple maps`

- Add your composites to new maps, and then tile them by going to: **Window** -> **Tile Horizontally**, and they'll be arranged automatically side-by-side.
- You can zoom and pan multiple maps simultaneously, e.g. to compare how an area looks in different bands or false color composites.
    - Go to **Map** -> **Synchronize Map Extents**

`Create a Normalized Difference image of vegetation (NDVI: Normalized Difference Vegetation Index)`

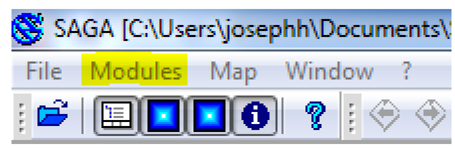
- In **Modules**, go to **Grid-Calculus -> Grid Calculator Tool**
- Notice that the formula, by default is `(g1 - g2) / (g1 + g2)`. This is the formula for normalized difference, ready to go!
- Set the Grid system to the `30m` resolution grid
- Add the near infrared band (to be `g1`), followed by the red band (to be `g2`).
- Set the **Name** to `NDVI`
- **Execute**
- If you change the gradient to **Red -> Grey -> Green**, your results should look like this:



- The image now shows thick vegetation in green, sparse vegetation in grey, dry areas or built up areas in red, and water and bare ground, open water and clouds in red.

Classify land cover categories with an unsupervised classifier

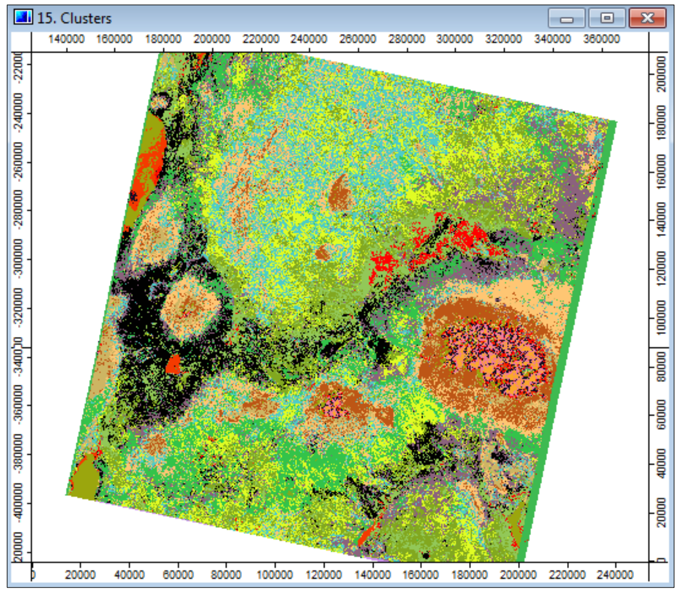
- In Modules, go to **Imagery-Classification** -> **Cluster Analysis for Grids**
- Add grids for all bands except the aerosol band and panchromatic band (i.e. bands 2, 3, 4, 5, 6, 7, 9, 10, and 11)
  - Check that you set the *NoData* parameter for all of the input grids.
- For **method**, I suggest using the *combined minimum distance / hill-climbing* algorithm.
- Set the number of **clusters** to `18`.
  - Always choose more clusters than you ultimately want. You can always aggregate two clusters into a single land cover class together afterward, but if clusters haven't distinguished between two land cover classes that you need, you'll have to run cluster analysis again with more clusters.
- Check the option to **normalise**, as this will significantly improve computational time.
- **Execute**.
- This algorithm takes a *long* time to run its course to an optimal set of clusters. It can take over 24 hours… The status bar at the bottom of the SAGA window will update as the algorithm runs, starting at:
  - Pass: 1 >> change 3.879993



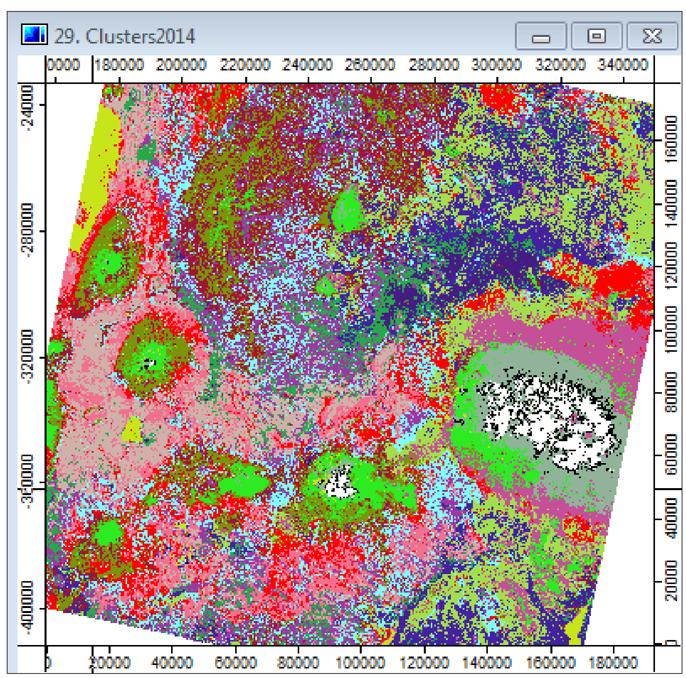
- Let the module run for about 20 passes, until the change between passes is very minimal.
- Once you are satisfied with the precision of change, click the **Modules** menu at the top of the SAGA window once (be patient for the menu to open), and deselect the **Cluster Analysis for Grids** module.
- A dialogue will ask, *Shall execution be stopped?*
  - Yes, stay the execution! The results up to this point will be saved in the temporary `clusters` grid.
- Save the temporary `clusters` grid as `clusters2014.sgrd`.

Cluster Analysis results

- The results of a cluster analysis will display with random discrete colors, like the examples below.
- Notice that the first example has a green stripe down the right and left sides of the image. This is caused by one or more of the input raster grids having slightly more or less data on the edges of the Landsat scene. This wastes some of the clusters, leaving fewer clusters to differentiate land cover classes. To avoid this, double-check that *all* of the input raster grids have *NoData* properly set before you run the **cluster analysis for grids** module.



- This example had *nodata* properly set before cluster analysis, yielding cleaner results:



Interpret Classification Results and Edit the Lookup Table	• As you move the mouse around an image, the z-value in the status bar at the bottom will display the class number. • In the **Data** tab of the **Workspace** window, select `Clusters2014`. • In the **Settings** tab of the **Object Properties** window, set the **Colors Type** to *Lookup Table*. o The lookup table is essentially a legend, and can be used to reclassify. • Open the ellipsis button (…) to the right of the Table (columns: 5, rows: 18) • Once the table is open, you can change the **color** and **name** of each class. • I suggest changing a cluster to a distinct color to find it. Once you know what it is, change its name to an appropriate land cover label and assign it an appropriate color. More than one cluster may be assigned the same color and class. • Use the papers, maps, and data contained in `gg328/lab07/kilimanjaro` and `gg328/satellite` for reference in your interpretation of the landscape. • To see this area in Google Earth, open `gg328/lab07/Kilimanjaro scene_area.kml`

Check your results with the professor	• Show the following to the professor:
	  o True color RGB composite
	  o Two false color composites
	  o NDVI
	  o Classified clusters

Saving Work and Closing SAGA	• When you save a project, SAGA will prompt you to save any temporary grids, shapes, or tables. If it does not prompt you to save something, then a permanent copy has already been saved.
	• SAGA automatically re-opens the previous project when you restart. This may include a lengthy process of loading all the project grids into working memory. If you want a fresh start when you re-open SAGA, then start a new project before you close the program (**File** -> **Project** -> **New Project**)
	• Be kind to our server: delete the original `.tar` and `.gz` files if you have saved them in your own drive.
	• If you delete temporary data layers from SAGA or close SAGA without saving the temporary layers, they will not be saved anywhere.
	• If you remove data layers from SAGA that *have* been saved permanently, then the original data files will still exist on disk.
	• Similar to other GIS software packages, SAGA stores *data* and data or map *properties* (including *nodata* settings, coordinate system, classification, symbology, and map layout) separately.
	  o If you save a SAGA project, then data and map properties are saved as part of the project.
	  o If you want to save data properties for a single layer, select the layer and use the **SAVE** button on the **Object Properties Window** to save a *Saga Parameter File* (`.sprm`). You can then LOAD these properties to apply them any data layer later.

PreLab 8: Filling Gaps

Purpose:

The Landsat program suffered big setbacks when Landsat 6 failed to achieve orbit and Landsat 7's side-line corrector (SLC) failed. Landsat 5 kept working well beyond its intended mission, but eventually shut down as well. From 2003 to 2013, most of the Landsat data suffers from the SLC-off gaps. This lab demonstrates three methods for filling SLC gaps in Landsat data: 1) patching, 2) simple gap filling, and 3) filling gaps with spline interpolation.

Data:

- Red bands and gap masks from Landsat 7 SLC-off images acquired in 2012 (February 6 and March 9)

References:

- Landsat file naming conventions: https://landsat.usgs.gov/naming_conventions_scene_identifiers.php
- USGS: filling the gaps to use in scientific analysis: https://landsat.usgs.gov/sci_an.php
- SAGA User Manuals, Volume 1 and Volume 2 at: http://www.saga-gis.org/en/about/references.html

Deliverables:

- For lab on Monday, be ready to show the professor two images of Kilimanjaro in March of 2012: `B3patch`, for which gaps are patched with data from February and `B3spline`, for which gaps have been filled with the spline interpolation method.

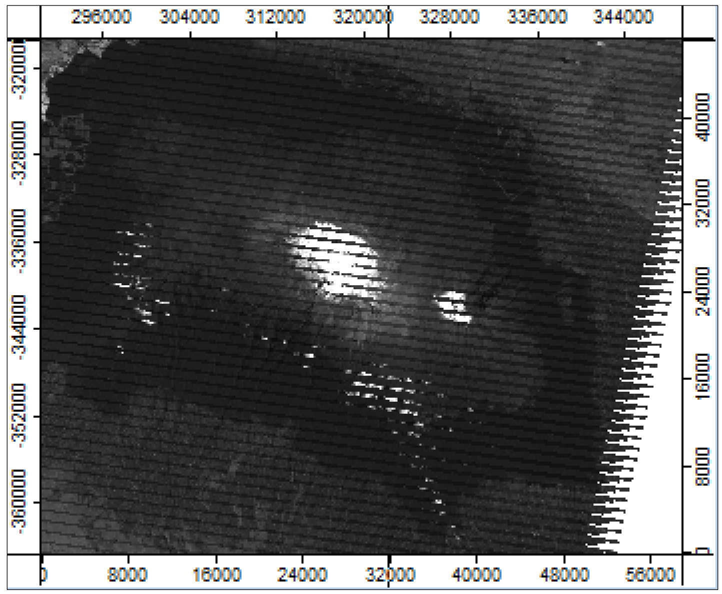
Procedure:

`Set up a project to fill gaps in Landsat 7 SLC off images`

- For the purposes of this pre-lab, I will instruct you how to fill the gap in a single band of the Landsat image. In reality, the gap-fill procedure would be necessary for each of the bands you intend to use.
- Open SAGA, and load the B3 (red) band for February 6 and March 9 2012.
- Load the `_GM_B3` grid for March into SAGA – this is a boolean *gap mask* grid showing where the L7 sensor collected valid data.
- Set the *nodata* value range for all three grids to include 0 as *nodata*.
- Save the project and both grids.
- Tip: if you're having trouble recognizing which image is which, the file name says all: LE7 168 062 2012 037 ASN 00 B3
    - LE7 = Landsat 7
    - 168 = Path 168
    - 062 = Row 062
    - 2012 = Year 2012
    - 037 = 37th day of 2012 (31 days in Jan + 6 days in Feb = 37)
    - ASN = Ground Station Code, 00 = Version number
    - B3 = Band 3

Add both images to the same map and inspect the results

- Add the February image to a map, and then the March image on top.
- Also set the stretching of each grid to match: **right-click** the grid layer, and select **classification** -> **set range to minimum/maximum**
    - Both grid color settings should now be using Graduated Color with a Value Range from 1 to 255.
- With the March image stacked on top of the February image, the Kilimanjaro area looks like this:



- These two images have luckily proven to be very complementary—the gaps did not overlap except at the very edge of the image.
- Some funny results come of this where clouds are involved and where the brightness of red in the landscape has changed between February and March—you can see on the southern and western slopes, cloudy areas from Feb are showing through the gaps in the Mar image. At the summit, bare rock from the Feb image appears in contrast to fresh snow in March.

Patch the March image's gaps with data from the February image

- Even though the results of patching are clearly not ideal, this is the primary method for fixing SLC-off data errors. Much of Google Earth imagery showed these distinct bands until Landsat 8 data has gradually replaced the SLC-off data. For example, the image at right is a screen capture of Lake Manyara from Google Earth, taken on April 2, 2015.
- Use the **Grid – Tools** -> **Patching** module to resample the patch grid and patch the gaps all in one step.
  - o Set the **Grid System** and **Grid** to the March grid.
  - o Set the **Patch Grid** to the February grid.
  - o **Interpolation method** should not matter because these grids are already aligned, *but*, since imagery is continuous data and we think the interpolation should be simple, choose *Bilinear Interpolation*.

- Set the *nodata* values for the results and **Save** as `B3patch.sgrd`

Prepare a mask in order to improve the efficiency of closing gaps

- Modules for closing gaps will, by default, attempt to fill the values of *nodata* throughout the entire grid. With Landsat data, however, there are large sections of the grid outside of the image footprint that should remain *nodata*. Therefore, we will mask the process by creating a 20-cell buffer around the valid Landsat data.
- Open the **Grid – Tools** -> **Grid Shrink/Expand** module.
  - o Set the input **Grid** to the March `GM_B3` grid
  - o Set the **Operation** to *Expand*
  - o Set the **Search Mode** to *Square*
  - o Set the **Radius** to 8 (the gaps are up to 400m, and the cell size is 30m, so expanding from each size by 240m will close all the gaps with some room to spare)
  - o Set the **Method** to *max* (all the good data areas = 1, and elsewhere = 0, so taking the max will expand the 'good areas')
  - o Note: this operation took about 3 minutes to run on my laptop.
  - o Set the *nodata* range to include the background and save the results as `maskExpanded.sgrd`

Interpolate the March image's gaps with Close Gaps

- If good images are not available for patching, you may need to *interpolate* the missing data. Spatial interpolation is a process of estimating unknown values based on nearby known values.
- The simplest module, **Grid – Tools** -> **Close Gaps** seems to use a focal statistic to assign *nodata* cells with the average value of their surrounding cells.
  - Set the input **Grid** to the March B3 grid.
  - Set the **Mask** to `maskExpanded`
  - Set the **Changed Grid** output to `[create]`
  - *Note*: this operation took 5 minutes to run on my laptop.
  - Set the *nodata* value for the results and save as `B3closeGaps.sgrd`

Interpolate the March image's gaps with spline interpolation

- Open the **Grid – Tools** -> **Close Gaps with Spline** module
  - Set the input **Grid** to the March B3 grid
  - Set the **Mask** to the `maskExpanded` grid
  - Set the **Closed Gaps Grid** to `[create]`
  - Set the **Maximum Points** to `20` (so it will not use more than 20 cells to interpolate a cell)
  - Set the **Number of Points for Local Interpolation** to 2.
  - Set the **Neighborhood** to *Moore* (a square-shaped neighborhood)
  - Set the **Radius** to `20` (allowing the neighborhood size to include up to 20 cells in any direction to find data for interpolation)
  - *Note*: this operation took 5 minutes to run on my laptop.

- The results of Spline *look* terrible because they've included a very large data range. Reclassify all the extreme data values to *nodata*: open the **Grid – Tools** -> **Reclassify Grid Values** module
  - Set the input **Grid** to the Close Gaps as Spline results: `LE7…_B3 [no gaps]`
  - Set the **Method** to *simple table*
  - Set the **Operator** to *min <= value <= max* (so that the ranges are inclusive)
  - Open the Lookup table, and reclassify extreme negative values to 0 and extreme positive values to 255, placing the interpolated extremes back within the range of a Landsat image. Make your Min and Max values small and large enough to capture both extremes in your data.

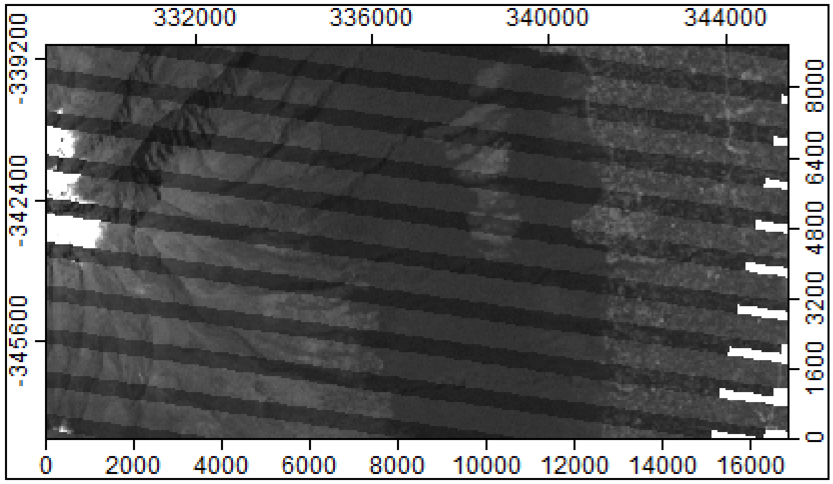
Minimum	Maximum	New
-9999999999999999999999	0	0
255	9999999999999999999999	255

  - **Execute**
  - Set the *nodata* values of the results and save as `B3spline.sgrd`

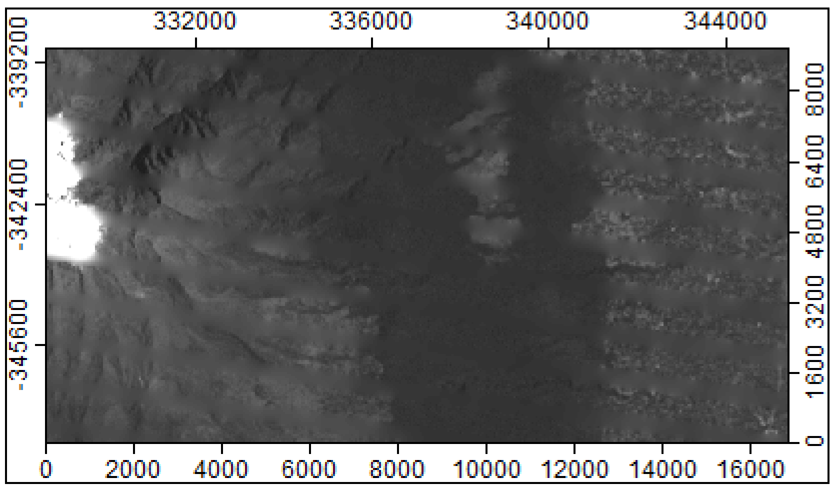
Consider the results of gap filling

- I have set the colors to *greyscale* and the value range to `0; 155` for each of the gap-filled images. The figures below show the same area east of Mt Kilimanjaro for each gap-filling method.

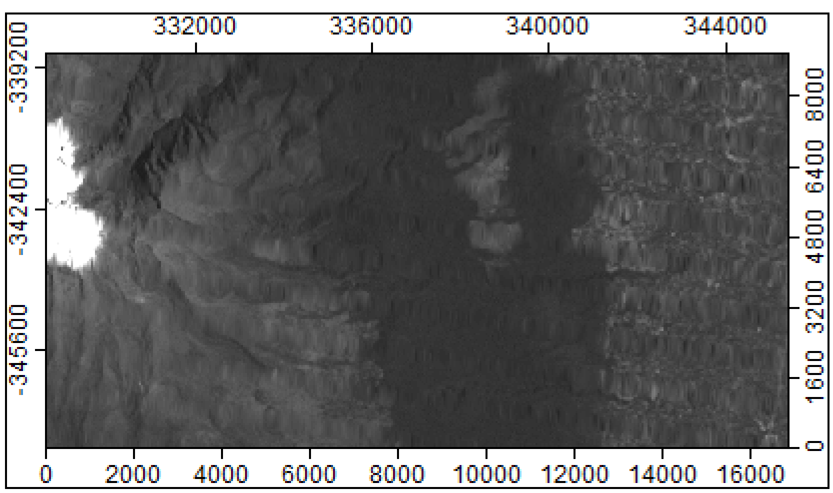
Method 1: Patching:



Method 2: Close Gaps



Method 3: Close Gaps with Spline



Considerations for filling Landsat 7 gaps:	• A more sophisticated gap filling procedure would combine interpolation with patching, applying a function to a patch image to match the brightness of the target image. Such a custom procedure is not available in SAGA. • Patching will work best if the patch images are very close in time, or perhaps at the same time in consecutive years with similar weather. For your own project, you'll need to fill the gaps of *every band* to be used in classsification. • Thankfully, Landsat 8 is now fully operational and features push-broom data collection that does not use a side-line corrector!

Lab 8: Supervised Classification

Purpose:

This week's challenge is to create *supervised classifications* of land cover categories in Landsat images. To prepare for supervised classification, we'll first make a *mask* showing only the areas of valid data for satellite images at $Time_1$ and $Time_2$. This will require *smoothing* the results of unsupervised classification with a *majority filter*, reclassifying to a Boolean mask, and combining the two masks. *Supervised classification* will require creating polygon *training areas* for each land cover category. A *maximum likelihood* classification algorithm will use the training areas to build a spectral signature and calculate probabilities for each land cover category. It then assigns each location to the category with the highest probability.

Data:

- Landsat 7 SLC-on image acquired February 21, 2000
- Landsat 8 image acquired February 5, 2014 (from Lab 7)

References:

- SAGA User Guide Volume II pages 247 to 263. The topics are Image Classification, Unsupervised Classification, Supervised Classification, and Preparing the Input Polygon Shapes Data Layer. Find the manual at: http://www.saga-gis.org/en/about/references.html
- The USGS Anderson Land Cover Classification system technical paper. This classification system will help you think the classes you'll use for your own projects: http://landcover.usgs.gov/pdf/anderson.pdf
- Examples of Land Cover Data: http://landcover.usgs.gov/landcoverdata.php
- Tanzania Land Cover from Africover (an ArcGIS layer with data): http://www.fao.org/geonetwork/srv/en/metadata.show?id=38238&currTab=simple

Deliverables:

- Please show the professor the following:
  - a vector version of a combined mask
  - a map of land cover created with supervised classification
  - a table of land cover classes resulting from supervised classification

Procedure:

SAGA makes extensive use of Lookup Tables for classifying and symbolizing colors for raster grids. You can save them and use them for other layers' symbology or for reclassification.

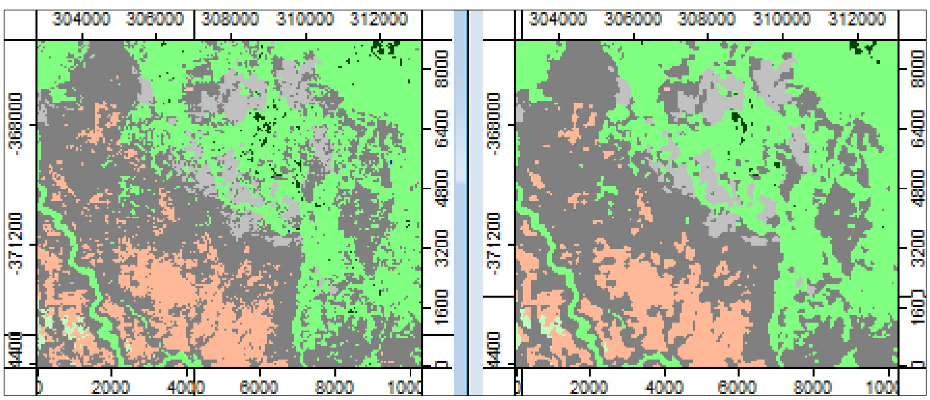
- In Lab 7, you used the **cluster analysis for grids** module to create a categorical raster (a raster of discrete categories) a Landsat image. This was *unsupervised classification*.
- Re-open your Lab 7 project and the lookup table for the clusters (you may have renamed these `Clusters2014`).
- The lookup table has a **Save** button, which gives options to save the table as plain text, comma-delimited text, or dbase. (`.csv`). Please save your lookup table as a `clusters2014classes.csv`
- You can also save a parameter file for any data layer or module—this file essentially saves all of the layer settings sot that you can load them in future projects or apply them to other layers. Save and load these with the **Save** and **Load** buttons in the **Object Properties** window.

Clean the classification with a majority filter

- Classification results are often very noisy, with a few cells of one class isolated amongst many cells of another class. This can be generalized / smoothed with a focal operation: majority filter. Majority filter will reassign the values of isolated cells with the majority of their surrounding neighborhood.
- Go to the **Grid –Filter** -> **Majority Filter** module
  - Set the input **Grid** to `Clusters2014`
  - **Search mode**: `Circle`
  - **Radius**: `2`
  - **Threshold**: `30`
- The threshold will keep any cell equal to 30% of the neighborhood the same. It will reassign any cell dissimilar to 70% of the neighborhood to the *majority* class of the neighborhood.

Apply your lookup table to the results of majority filter.

- Re-apply your previous look up table:
  - Set the **colors type** to `Lookup Table`
  - Open the Lookup Table, and **LOAD**
  - Load the table you just saved: `clusters2014classes.csv`
  - Apply the changes, and you should see your colors and classes as usual.
- Save the Majority Filter output as `Clusters2014Majority`
- Compare the results. The image on the left is the original, and on the right is the smoothed version.



Create a Mask of all valid areas of the satellite data, excluding clouds, nodata, and any other problematic areas. First, make a copy of the clusters lookup table.

- To make a mask, use your unsupervised classification (*clusters*) to reclassify all good data and classes to `1`, and all bad classes to `0`. To do this, you'll make a copy of the Lookup table for clusters and use it for reclassification.
- Load `clusters2014classes.csv` to the project from the **Data Source** window. You may have to *refresh* the window (**right click** -> **refresh**).
  - The default *nodata* values for the table conflict with your first class. Change the *nodata* range from `0; 0` to `-1; -1`
  - Rename your `clusters2014classes` table to `reclassMask` using the object properties window.
  - Right-click the `reclassMask` table in the Workspace window and **save as...** to save the table as `reclassMask.csv`

`Edit the lookup table for reclassifying to a Boolean mask`

- Open the `reclassMask` table.
- Go to **Table** -> **Add Field** and add a field after Maximum
  - o Set the **Name** to `NEW`
  - o Set the **Field Type** to *1-byte integer*
  - o Insert the field **After** the `Maximum` field
- Now edit the New values for reclassification:
  - o Enter `0` for the `New` value for any clouds, smoke, or other bad data. This will become `nodata` later on.
  - o Enter `1` for the `New` value of any legitimate class.
- The **reclassify grid values** module requires exactly three columns: `minimum`, `maximum`, and `new.` Therefore, the other columns must be deleted.
  - o Go to **Table** -> **Delete Fields**. Delete the `Color`, `Name`, and `Description` fields.
- Close the table, right-click the table in the **Workspace** window, and **Save** to make the changes permanent.

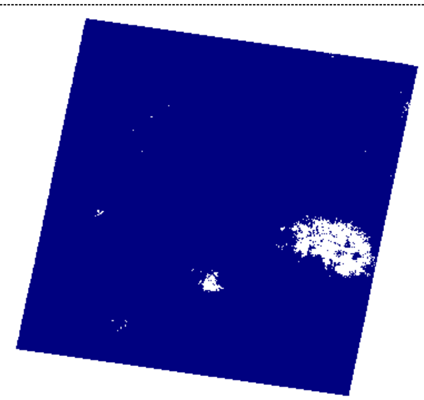
`Reclassify the clusters to a Boolean mask`

- Open the **Grid – Tools** -> **Reclassify Grid Values** module.
  - o Set the input **Grid** to `Clusters2014`
  - o Set the **Reclassified Grid** output to `[create]`
  - o Set the **Method** to *simple table*
  - o Open the **Lookup Table** and **load** `reclassMask.csv`
  - o Change the **operator** to *min <= value <= max*
- Set the *No Data* range of the mask to include `-1` and `0`, change the name to `mask2014`, and save the grid as `mask2014.sgrd`.

`Now create a mask for the Landsat 7 2000 image`

- Make a mask for the year Landsat 7 image from 2000.
  - o Load each band, set the *nodata* range for each
  - o run **cluster analysis for grids**
  - o **Reclassify** any clouds or bad data to *nodata*
  - o Save the results as `mask2000`.
- **Save the Project**, including the new mask images.

`Integrate the two masks with Grid Masking`



- Make sure you have set the *No Data* range of `mask2014` and `mask2000` to
- Go to **Grid Tools** -> **Grid Masking**
  - o **Grid:** `mask2000`
  - o **Masked Grid:** `[create]`
  - o **Mask:** `mask2014`
- Rename the output, `masked grid`, to `maskcombined`
- The results should look similar to the mask at the right.
- There are so many grids loaded in the project by now, you may want to save and start a new SAGA project.

Convert the mask grid to vector polygons

- Before you start digitizing training polygons in Google Earth, you need to know where the study area is and what areas to avoid due to cloud cover.
- First, convert the mask raster grid to a vector polyon shapefile.
- Go to the **Shapes – Grid** -> **Vectorising Grid Classes** module:
  - Set the input **grid** to `maskcombined`
  - Set the **Class Selection** to *one single class specified by class identifier*
  - Set the **Class Identifier** to `1`
  - Set the **Vectorised class as…** to *each island as a separated polygon* (because Google Earth is better with simpler polygons, and choosing the single part option will help with that)
  - **Execute.** This tool can take some time to run depending on the number of classes and complexity of the polygons…
  - Save the `Polygons` Shapes layer as `mask.shp`

Use QGIS to convert the mask from a shapefile to a KML file for Google Earth

- Open **QGIS** and add `mask.shp` to the map
- Save `mask.shp` as a **KML** file, `Clusters2014.kml`.
- While saving, change the **CRS** to `WGS 84` (`epsg: 4326`). This is the CRS used by Google Earth
- *Note*: If you need a polygon of your study area with clean boundaries, use QGIS's **Vector** -> **Geoprocessing Tools** -> **Convex Hull(s)** tool on your mask.

Digitize training areas in Google Earth

- Open `mask.kml` – it should automatically launch in **Google Earth**.
- In the **Places** panel, Right-click **My Places**, and **Add** -> **Folder**
  - Name the folder `Training`
- With the Training folder highlighted, use the **Add Polygon** tool 
  - Draw a polygon around an area of land cover that *has not changed* from 2000 to 2014.
  - Check for change using the time-slider  in Google Earth and by comparing the RGB composites and NDVI images in SAGA.
  - Set the **Name** of the area to a land cover category name (e.g. water), numeric code (e.g. 9), or letter code (e.g. WAT) of your choosing. Be sure to spell and type category labels consistently.
  - Add a few polygon areas for each land type within the extent of the mask and make sure they are all saved under the `Training` folder.
  - A potential list of categories includes:

```
1.  Agroforestry
2.  Closed-canopy Forest (> 70% tree cover)
3.  Wooded savannah / pasture (30% to 70% tree cover)
4.  Open savannah / pasture (< 30% tree cover)
5.  Rain-fed agriculture
6.  Irrigated agriculture
7.  Bare ground (rock and soil)
8.  Urban / Built Environment
9.  Open Water
10. Shadows (add shadows later, in SAGA)
```

- When finished, right-click the `Training` folder and **Save Place As** to save as `training.kml` (*not KMZ*. QGIS cannot open kmz files)

Use QGIS to convert training areas from a KML file to a shapefile

- Open **QGIS** and add `training.kml` to the map
- Save `training.kml` as an ESRI shapefile, `training.shp`
- While saving, change the **CRS** to *WGS 84 / UTM Zone 37N* (`epsg: 32637`)

Load training areas into SAGA. Edit existing areas and add new areas

- Open `training.shp` in **SAGA**
- Open an RGB composite from `2014` in one map, and an RGB composite from `2000` in a second map.
  - If a composite loads with poor symbology, set the **Colors** type to *RGB*.
- Synchronize the map extents.
- Add the `training` shapes to the `2014` map.
- In the **Data** panel of the **Workspace** window, highlight the training shapes layer.
- You can edit the vertices or attributes of any shape:
  - Use the selection tool  to select any of your training polygons.
  - Switch the **Object Properties** window to the **Attributes** tab.
  - Here, you may edit the **name** of any training polygon.
  - Right-click the map window and select **Edit Selected Shape**
  - Click and drag any of the **vertices**.
  - Press **enter** on the keyboard to finalize the edits. (or right-click and deselect **Edit Selected Shape**)
- You can also add new training areas (e.g. add areas for `Shadow` now):
  - Right-click on the map
  - Select **add shape**
  - Click once for each vertex of the new shape
  - Press **enter** on the keyboard to finalize the shape
  - Edit the **name** of the shape in the **Attributes** panel of the **Object Properties** window.

Classify the 2014 Landsat 8 image using supervised classification

- Load the Landsat 8, 2014 image into SAGA. Use bands `2`, `3`, `4`, `5`, `6`, `7`, `10` and `11`
- Go to the **Imagery – Classification** -> **Supervised Classification** module and apply the following settings
  - **Grids**: load all 7 bands
  - **Classification**: create
  - **Quality**: create
  - **Training Areas**: training (your shapefile)
  - **Class Identifier**: name
  - **Class Information**: create
  - **Method**: maximum likelihood
  - **Normalize**: check
  - **Probability Threshold**: 5
  - **Probability Reference**: Absolute
- Rename the outputs to `Classify2014`, `Quality2014`, and `ClassInfo2014`, using the settings panel *name* field. **Save**…

`Interpret the supervised classification results`	• The `Class Information` table shows the spectral characteristics of each class, according to your training areas o Tot_N = number of classified cells o Roi_N = number of training cells o 01_Roi_M = band 1 mean (average) o 01_Roi_S = band 1 standard deviation (variability) o 01_Roi_Min = band 1 min o 01_Roi_Max = band 1 max • The `Classification` image shows the results o The grid should already have a lookup table with labels already. Edit the colors and labels to make a good legend. o Save the table so that you can apply it to other grids. • The `Quality` image shows the maximum likelihood (probability of the most likely class) • If large areas of the image have remained unclassified, it means that the spectral signatures of those areas are not similar enough to any of your classes. Consider adding training areas to existing classes and/or adding new classes. • Often times, two or more areas that should belong to the same class *look* very different in the satellite images. For example, *clear water*, *turbid water*, and *water with algae blooms* all have different spectral signatures, but may belong to the same class of water. In this situation, you should make three separate classes for each type of water, classify the image, and then use the reclassify tool to merge the three water classes into one. • It may help to add NDVI to the stack of grids used for classification. o It is also possible to use data derived from digital elevation models, other normalized difference band ratios, texture analysis, or principal component analysis. • Land cover classification often requires an iterative process of classification, revision, and re-classification. However, use this data for practice learning, and spend time revising with your own project.
`Reclassify and Mask the results`	• Use the **Reclassify Grid Values** module to aggregate Shadow and Forest together, as well as irrigated agriculture and rainfed agriculture. Follow a similar workflow as that of creating a mask from unsupervised classification (clusters and a lookup table). • Apply the `maskCombined` mask image to both classification results, using the **Grid Masking** module.
`Repeat for 2000...`	• Before Lab 9, repeat the classification process for the Landsat 7 image from 2000. For Landsat 7 images, load all bands except the panchromatic band (i.e. bands `1`, `2`, `3`, `4`, `5`, `6` ,and `7`). • **Save**… • Run Supervised Classification again, this time with the 2000 images. • **Save…**

Lab 9: Ground Truth and Change Detection

Purpose:

This week's challenges are to assess the accuracy of land cover classification, detect change, and summarize change per enumeration area.

Accuracy will be assessed by creating random ground truth points stratified by land cover category, cross-tabulating classified locations with ground truth locations, and calculating producer's and consumer's accuracies and a Kappa statistic. In order to create random points, you'll generalize the classification grid, vectorize the grid into polygons, and use the Random Points research tool in QGIS. You'll assess accuracy only for the most recent classification, because you do not have sufficient data to 'ground truth' historical data.

Change will be detected by cross-tabulating classified land cover for $Time_1$ with $Time_2$. Before doing this, you will likely want to generalize the land cover categories into a just a few categories of interest to you.

Finally, you will summarize change per enumeration area by cross-tabulating administrative areas with change.

Data:

- Your supervised classification grids from Lab 8
- Your Tanzania districts from Lab 5.

References:

- Confusion Matrices for Classification Error Analysis:
  - http://spatial-analyst.net/ILWIS/htm/ilwismen/confusion_matrix.htm
  - http://www.biology.ualberta.ca/gis/uploads/instructions/AVErrorMatrix.pdf

Deliverables:

- Create a map of districts with rates of change for any land cover category calculated.
- Create a confusion matrix table from your supervised classification and ground truth points, and calculate accuracy, reliability, and overall accuracy.

Procedure:

Prepare classified grids for ground truthing	• Classification algorithms normally make very noisy results, which produce overly complex grids and polygons. We eventually want to create random points stratified by land cover categories, and that function in QGIS will crash if polygons are too complex. • First, if you have multiple classes referring to the same land cover type, aggregate them together by reclassifying. For example, if you have *clear water* and *turbid water*, reclassify both to *water*. • Second, make sure the background/nodata is *included* as one of the classes for the purposes of generalization. o Set *NoData* to very large negative numbers, so that all data is included.

`Generalize the classification with the Resampling module.`	• *Resampling* to a more generalized grid size will simplify the data enough for QGIS to generate a stratified random sample.
	• Go to the **Grid – Tools** -> **Resampling** module
	  o **Grid**: select your classified grid, `Classify2014`
	  o **Target Grid**: user defined
	  o When defining the user-defined grid, only change the **cellsize** to 300 (it's best to choose a multiple of the present cell size for the most straightforward resampling)
	  o **Interpolation Method:** Majority

`Remove noise from the classification with the Majority Filter module.`	• Now that you've resampled, remove noise from the 300-meter resolution grid with the **Grid – Filter** -> **Majority Filter**
	  o Try a square search mode, radius of 3, and Threshold of 30 percent. This means that within a 7*7 moving window, the center of the window will be converted to the majority class in that window.
	  o If you wanted a less generalized output, you could use a smaller radius, and/or a higher threshold percentage.
	• Apply a lookup table to the majority filter results, using the same classes and labels as you did for the classified image. You should now exclude *nodata* again.
	• *<u>Optional</u>*: for your analysis, you may also want to use a Majority Filter on the 30-meter resolution image, for less noisy results.

`Convert grid classes to vector polygons`	• Go to the **Shapes – Grid -> Vectorising Grid Classes** module
	  o **Grid:** Select the results of your resampling to 300m cellsize
	  o **Polygons:** [create]
	  o **Class Selection:** all classes
	  o **Class Identifier:** ignore this
	  o **Vectorised class as:** one single (multi-)polygon object
	• Right-click the shapes layer and **save as…** a *shapefile*, `classes2014.shp`

`Create a stratified random sample`	• Load `classes2014.shp` into QGIS
	  o *SAGA has not saved projection information for the shapes!* The Supervised Classification tool seems to have a bug, whereby it does not record the coordinate reference system for the output grid.
	  o Set the coordinate reference system to UTM Zone 37 North (WGS 84 datum), with EPSG: 32637.
	• Go to Vector -> Research Tools -> Random Points
	  o **input boundary layer:** to your vectorised classes
	  o Under **stratified sample design,** set the **number of points**: to 10
	  o **output shapefile**: `random_pts.shp`
	  o *this operation can take a several minutes and 'freeze' on one percentage for a while…*
	  o Point locations will not be exactly on each class due to our generalization, but they at least will cluster around each class.
	  o Save the output as `random.kml` , changing the CRS to WGS 84.

Ground Truth random points in Google Earth	• Open `random.kml` in Google Earth
	• Double-click the first random pin point to zoom to it
	  o Right-click the pin and go to **properties**
	  o Edit the name of the properties to an integer number corresponding to the appropriate category in your classified grid. To avoid confusion later, *use the same numbers that SAGA assigned to your classes* in the Supervised Classification module (the min/max values). Find these in the lookup table.
	  o Repeat for the remaining pins
	• When you are finished with all the pins, right-click the random folder and **save place as**
	  o Save as a KML file, `groundTruth.kml`

Import ground truth data into SAGA	• Use QGIS to convert `groundTruth.kml` to a shapefile, `groundTruth.shp`.
	  o When you save the shapefile, change the coordinate reference system to UTM Zone 37N (WGS 1984), EPSG: `32637`
	  o Use the field calculator to create a new field of the *integer* data type, named `gtruth`, and equate it to `Name` (converting from text to integer)
	• Open the ground truth shapefile in SAGA and convert it to a grid.
	• Go to the **Grid – Gridding** -> **Shapes to Grid** module
	  o **Shapes**: `groundTruth.shp`
	  o **Attribute**: the field you created to convert name to integers
	  o **Preferred target grid type**: 1 byte integer
	  o **Target grid**: *grid*
	  o **Execute**, setting the following:
	  o **Grid system**: 30m resolution grid used by your classified grid,
	  o output **Grid**: `[create]` (don't overwrite another grid!)
	• In my test, SAGA applied the value 129 to all the cells with *no data*. Check this, and set the *No Data* range appropriately.
	• Change the name to `GroundTruth2014` and **Save**

Cross-tabulate ground truth data classified data	• Go to the **Grid – Analysis** -> **Cross-Classification and Tabulation** module
	  o **Input Grid 1**: `GroundTruth2014` (*grid1* becomes *rows* in the table)
	  o **Input Grid 2:** `Classify2014` (*grid2* becomes *columns* in the table)
	  o **Cross-Classification Grid**: `[create]`
	  o **Cross-Classification Table**: `[create]`
	  o **Maximum Number of Classes**: enter any number greater than the number of classes you have used
	• Right-click on the cross-tabulation table output and save as a `.csv` file
	• Open the `.csv` in Excel
	• Label the columns and rows with their class names or abbreviations.
	• Calculate the *accuracy* (producer's), *reliability* (consumer's or user's), and *overall accuracy*. Refer to the references for this lab for assistance.

`Aggregate classes using Reclassify Grid Values`	• Your research question may only need a few classes, or may need several classes to be aggregated in order to simplify results. Accordingly, you may want to aggregate many classes into fewer classes or into *nodata* before you analyze change.

`Analyze Change`	• Grids must be in the same grid system in order to analyze change.
	  o Go to **Grid – Tools** -> **Resampling**
	  o **Resample** the `classified 2014` grid into the `2000` grid system using the *nearest neighbor* interpolation method
	  o Both grids should now appear together in the same grid system
	• Grids should have lookup tables with labels for best results
	  o Apply or create a lookup table for both images
	• Go to **Imagery – Classification** -> **Change Detection.** Leave the defaults except for the following:
	  o **Initial State**: `classified 2000 image`
	  o **Final State**: `classified 2014 image`
	  o **Changes**: `[create]`
	  o **Tables –Changes**: `[create]`
	• The `Changes table` is a crosstabulation displaying how much land was observed to change between each category from the first image to the second.
	• The `Changes grid` shows every possible combination of categories.
	• Save the `table` and `grid`.

`Generalize change with the reclassify module`	• There many categories of change and this is the last chance to generalize them with the Reclassify Grid Values module. Unnecessary categories may be changed to *nodata*.
`Summarize Change by Enumeration Areas`	• If there was only one variable to summarize, Zonal Statistics would work well for summarizing change by enumeration area. However, you probably have several categories of change to summarize. To summarize them all at once, we can again use the **Cross-Classification and Tabulation** module. To prepare for this, first create a grid version of the enumeration areas.

`Convert polygons to grids`	• From QGIS, save a version of Tanzania's districts as a shapefile. While saving, change the coordinate reference system to that used by your SAGA grids: UTM Zone 37 North (WGS 84 datum), with EPSG: 32637.
	• Load the Districts shapes into SAGA and convert them to a grid:
	• **Grid – Gridding** -> **Shapes to Grid**
	  o **Shapes:** districts
	  o **Attribute:** cat (or any other integer primary key)
	  o **Method for multiple values:** first
	  o **Preferred target grid type:** integer (1 byte)
	  o **Target grid:** grid
	  o **Execute.** Set the **Grid system** to match the system used by your change image, and set the **Grid to** `[create]`

`Summarize change by enumeration area`	• If there is any *nodata* area on the grid of enumeration areas, identify its value and include it in the *nodata* range of the grid before summarizing change.
	• Go to the **Grid – Analysis** -> **Cross-Classification and Tabulation** module
	  o **Input Grid 1**: `TZdistricts` (*grid1* becomes *rows* in the table)
	  o **Input Grid 2:** your `Changes` grid (*grid2* becomes *columns* in the table)
	  o **Cross-Classification Grid**: `[create]`
	  o **Cross-Classification Table**: `[create]`
	  o **Maximum Number of Classes**: enter a number equal to the maximum value of both inputs. E.g. if the largest District ID is 83 and the largest change class is 35, enter 83.

`Pivot the data to summarize amount of each type of change by enumeration area`	• Save the `cross-tabulation` table as a `.csv` file • Open the `csv` file in Excel • The table should be arranged with change categories as columns, and the cell counts as the values. • Row labels, corresponding to district IDs, are missing. Because there is a header row, the row #'s are all one greater than the district ID. You should therefore add a column to the beginning of the data and enter in the appropriate district ID for the districts in your study area. • You may delete extraneous rows that do not correlate to a district in the study area. • Re-name the column headers from integers to appropriate field names for a database. These should correlate to your change classes. • Save the `.csv` file.

`Join the change data to enumeration areas and calculate change!`	• Open QGIS and add the .csv file to your map. Then import it into your SQLite database with Tanzanian districts.
	• Join the table to the districts
	• Knowing that each cell is 900 square meters and Area_Azim is an accurate area calculation in square kilometers, it is now possible to calculate and map rates of change for a land cover category of interest, e.g. *forest*.
	  o percentage of each district covered by forest at time1 and at time1
	  o percentage area of each district with forest regrowth
	  o percentage area of each district with forest loss

Term Project

Executive Summary

Abstract

The overall goal of the term project is for you to develop a research plan for a question involving land cover and population change in a developing country. You'll implement your plan using open source and/or free GIS data and software, specifically QGIS for vector analysis of population and SAGA for image processing and raster analysis of land cover. Your use and interpretation of GIS data and methods should be critically aware of limitations in the data and methods, and potential impact of the research. Finally, you will communicate your research question, data and methods, results, and conclusions clearly enough for GIS experts to evaluate and replicate your work, and for non-experts to understand your question and key findings.

Honor Code

I expect that you will run into technical challenges at various stages in this independent project. It is also expected that you will want to talk through problems and seek advice from your peers and others, and you are encouraged to do so. All of your work, however, should be your own. Think of it like asking a friend to proofread a paper for you: your friend may suggest revisions, but you must make the revisions yourself. In exchange for this open and collaborative policy, I'd also like everyone to contribute to a more public body of knowledge on using open source GIS in the form of an open Question and Answer forum at https://github.com/GIS4DEV/Q-and-A/.

Part 1: Corrected Population

Goals:

Find population/census data for two different years for any developing country, except Tanzania. Countries on the United Nations least developed country (LDC) list[1] are acceptable; for others please ask approval first.

Find spatial data for administrative areas corresponding to the population data. The population data and spatial data will ideally be at the 2nd or 3rd administrative level or better[2].

Project spatial data into an appropriate projected coordinate system and correct any cardinality errors resulting from single-part/multi-part features or differences between attribute tables of population data and geometries of administrative areas. Document the initial state of the data, the procedures you have taken to fix any errors, and the final state of the data.

Integrate all the spatial and attribute data in a new SpatiaLite database.

Thoroughly describe metadata for the two sets of population data and for the spatial data (one or two sets).

Describe how well you can represent the population data for both years, including use of primary keys, foreign keys, and/or data aggregation for joins. Be aware that administrative areas may change over time, so if administrative areas are only available for $Time_1$, you will need to assess how well $Time_2$ will join to them.

Assess the accuracy of attribute data, e.g. by comparing descriptive statistics with other reputable sources and by checking its internal consistency. Identify and describe any errors. Fix attribute errors wherever possible, and document all changes to data (e.g. SQL code for statements that change data, and comment the purpose of that code).

If there is any mismatch between your tabular data and geographic data caused by boundary changes, document the mismatch in your data tables and represent mismatched polygons as missing data on the map. You are not expected to edit the geographic features at this stage.

Deliverables:

- Metadata for all data sources describing the state of the data as you found them, including appropriate summary statistics and code and description for any custom projection.
- Description of attribute data correction methods.
- Revised Metadata for all data sources describing the corrected state of the data, including primary keys and foreign keys and explanation of any problems.
- Two choropleth maps of population density, at $Time_1$ and $Time_2$.

---

[1] http://unctad.org/en/pages/aldc/Least%20Developed%20Countries/UN-list-of-Least-Developed-Countries.aspx
[2] See http://en.wikipedia.org/wiki/Table_of_administrative_divisions_by_country

Part 2: Population Change

Goals:

Assess the spatial accuracy of your data and correct any spatial data errors, i.e. geometry errors and topology errors. Document the initial state of the data (geometry errors, topological errors), the procedures you have taken to fix any errors, and the final state of the data.

Make sure that the administrative areas for both population times match exactly, so as to not misrepresent change in population over time. Document any modifications you make to the data in order to match administrative areas from $Time^1$ to $Time^2$.

Find the total population change and percent population change for your country. Present results in the form of a map of percent population change and a table of total change and percent change per enumeration area.

Deliverables:

- Documentation of analysis, including:
    - Description of the relationship between enumeration areas at $Time_1$ and $Time_2$. In other words, how have enumeration areas changed?
    - Explain/justify your choice of enumeration areas for $Time_1$ and $Time_2$, and your choice of areas for the final results.
    - Clearly document the status of geometry and topology errors in your data, your methods to correct these errors, and the final status of geometry and topology errors.
    - Clearly document any additional edits or changes to the data made after Part 1.
    - Explain any remaining uncertainties or errors in your results.
- Thematic map of the percent population change in your country.
- If necessary, revised/updated maps and tables of the population densities at $Time_1$ and $Time_2$

Part 3: Research Proposal

Goals:

Choose a specific study area within the extent of one Landsat scene. Research a conservation and/or human development issue that can be addressed with the use of geospatial technology, incorporating both population and land cover information.

While you are researching an issue, identify and record important landscape and land cover categories in the area. Take note of different categories in use by stakeholders with different points of view, e.g. development agencies, state government agencies, local authorities, and different ethnic or indigenous groups. Make digital copies of any maps you find of landscapes and land cover for your own reference.

Deliverables:

The deliverable is a research grant proposal (one pdf file), including:

- Title
- Executive Summary (100 to 200 words)
- Research Description (900 to 1100 words) including:
  - Purpose, with a specific research question to be answered
  - Very brief review of any previous work done in the area, including 2 peer review papers
  - Your methodology, including work and data you already have
  - Description of the land cover categories present in your study area and their significance for your research question.
- Expected Results, Benefits & Broader Impacts (100 to 200 words)
- References (use the APA style)
- Map of the Area of Interest (make this yourself, showing, at a minimum, your country and the study area or satellite image footprint).
- Budget (List and cost) of specific high-resolution image(s) or other data archived by commercial data provider(s)

Evaluation:

I will evaluate the proposals like a grant program officer would. These are the criteria, for example, for an applied GIS award:

- How clearly does the student explain the purpose of his or her research?
- How relevant is the research to Open GIS, population, and land cover in developing countries? (text of this point is modified to fit this course)
- How creative is the proposed research?
- Are the methods technically capable of achieving the purpose of the research?
- Will the money be wisely used?
- How does the proposed research positively impact such categories as society, the environment, education, public policy, science in general, etc. ?

Part 4: Land Cover Change

Goals:

Classify land cover at two different times, in the extent of the same Landsat path and row.

Assess the accuracy of your most recent image classification, with at least 10 points sampled in each land use category.

Detect change from time 1 to time 2, and represent this change in map and tabular format.

Finally, summarize important aspects of land cover change for your research question according to the aerial units you defined in part 2 of this project.

I expect the Remote Sensing and GIS analysis and methods for the projects to be complete at this phase of the project. In part 6, you will write up your interpretation of the results.

Deliverables:

The Deliverable is a single pdf document, including:

- Land cover map at time 1,
- Land cover map at time 2,
- Legend for land cover maps
- Table of class info for time 1,
- Table of class info for time 2,
- Table of accuracy assessment for time 2,
- Land cover change map
- Legend for land cover change map
- Tabular summary of land cover change in square kilometers
- Tabular summary of your selected land cover change variables per enumeration areas. Enumeration areas may cover only part of the Landsat scene, and the land cover change variables may be reclassified generalizations of change, keeping only the types of change that you are interested in.
- Map of land cover change per enumeration areas, expressed as a density, rate, or percentage.
- Summary of the methodology used, with enough detail that any GIS analyst could replicate the results, and enough explanation that any end user of your map could understand the purpose of the methods.

Part 5: Final Report

Goals:

Interpret and draw conclusions from your background research and your analysis of population change and land cover change.

Deliverables:

Format all of your work to-date into a single report containing the following sections:

- Introduction
  - From research proposal
- Background
  - From research proposal – include *brief* review of other studies overlapping your study area or question topic.
- Methods
  - Integrate methods from all parts into one organized description of how the work was accomplished. A competent GIS analyst should be able to reproduce your methods from the beginning: downloading population data, enumeration areas, and Landsat images.
- Results
  - Present the results – maps, tables & statistics, but do not offer your own interpretations or analysis of the results yet. The tables and figures in a results section should be designed with simplicity in mind: give only enough information to tell the story answering your question. The final figure(s) should be a map and/or table to summarize the relationship between land cover change and population change.
- Discussion
  - In this section, discuss the meaning of the results. Use the results to tell a story. Be sure to also discuss limitations of the data and methods, drawing from the many critiques we have read and discussed in this course.
- Conclusions
  - First, answer your research question based on your results and discussion. Second, discuss the implications of your research: does the research contribute to scientific theory? Does the research have implications for policy and practices? What additional work is needed, or what special considerations should be taken to maximize positive impacts or avoid negative impacts of this research?
- References
  - Include citations to all papers, books, data sources, etc. Use the APA style.
- Appendices
  - Here, include data tables that did not fit well in the results section. This includes metadata, large tables of descriptive statistics, data on population change and density for the whole country, class information for your classified land cover, and the full change detection image and table.