IST-652 Final Project Report

WALS-Dataset

Randall Scott Taylor

Syracuse University

Abstract

Description of the data and its source(s); the purpose of this study is the presentation of a suitable dataset, that has been extracted, transformed, and loaded into python3 for further analysis. The subject matter of this study will be derived from the World Atlas of Language Structures (WALS) Online, which is a database of structural (phonological, grammatical, lexical) properties of languages gather from descriptive materials, from around the world. First release by Oxford University Press as a book with CD-ROM in 2005, it was released in a second addition on the internet in April 2008. The database is maintained by the Max Planck Institute for Evolutionary Anthropology. The editors are: Martin Haspelmath, Matthew S.Dryer, David Gil, and Bernard Comrie. The researcher finds it paramount to distinguish these contributors, and there works therein, before commenting further upon the study.

The purpose of this study of the above described data will be to provide insight and understanding to the dataset, the story the dataset contains and to answer further research question identified, within the document. The goal of this final project report will be to identify anything interesting within the distribution of the data, and the families of the various languages. Specifically, from the raw data can the genus family's groupings be shown within the data, and, are there any geospatial inferences that can be made, from visual display of those findings.

IST-652 Final Project Report

WALS-Dataset

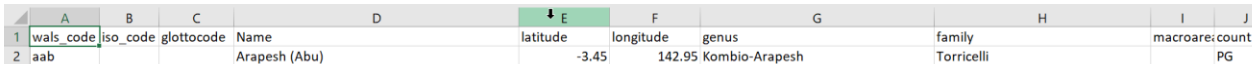
WALS-Dataset

The World Atlas of Language Structures dataset, here after referred to as WALS, can be accessed via the world wide web at the following address: https://wals.info/. The Dataset can be obtained from the website in the form of a comma separated value file. The researcher brought in the .csv file into the local environment of Jupyter notebooks for further data exploratory analysis utilizing the Python 3 programming language.

Methods of Analysis

Data Exploration.

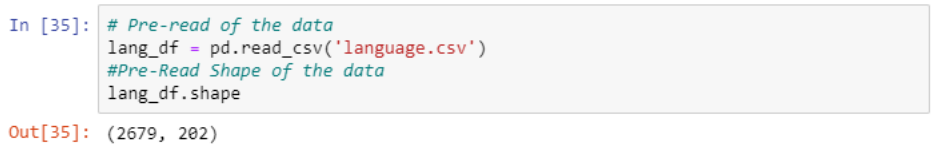
The WALS dataset, obtained to a local machine under the title of languages.csv, was downloaded from the aforesaid website, and was imported into Python 3 via the Jupyter notebook's IDE. The dataset contains the release of data showing the geographical distribution of structural linguistic features years of data (from 2005 to 2008). The database is updated yearly. Reviewing the Data, first step in the process is to understand the dataset that is to be cleaned, modeled, and visualized. The features (columns) of the dataset are presented for exploration in the wide format, a subject's repeated responses will be in a single row, and each response is in a separate column. To give the reader a sense of the data collected with these rows, see the following demonstration: a screenshot of the raw .csv data:



	A	B	C	D	E	F	G	H	I	J
1	wals_code	iso_code	glottocode	Name	latitude	longitude	genus	family	macroare	count
2	aab			Arapesh (Abu)	-3.45	142.95	Kombio-Arapesh	Torricelli		PG

Figure 1 (Raw Dataset)

The columns identify the languages ‘Name’, ‘latitude’, ’longitude’, ’genus’, ’family’, and finally the codes utilized as primary keys within the study; ‘wals_code.’ Utilizing Python to inspect the total rows and columns, the following code snippet demonstrates that import utilize pandas’ libraries:



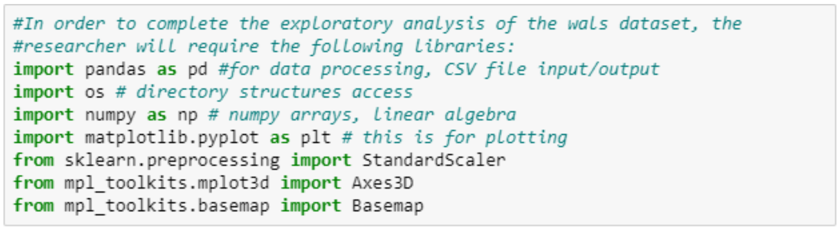
```
In [35]: # Pre-read of the data
         lang_df = pd.read_csv('language.csv')
         #Pre-Read Shape of the data
         lang_df.shape

Out[35]: (2679, 202)
```

Figure 2(Rows and Columns Totals)

The data exploration, thusly has identified this dataset to consist of the wide format, consisting of 2679 individual observations (rows) and 202 variables (columns). The nature of the data is that of categorical listings (languages, languages families) numeric listings of longitude and latitude, and ratings systems for various linguistics measures done within the study. At first inspection there are missing values within the .csv file.

Data Cleaning.

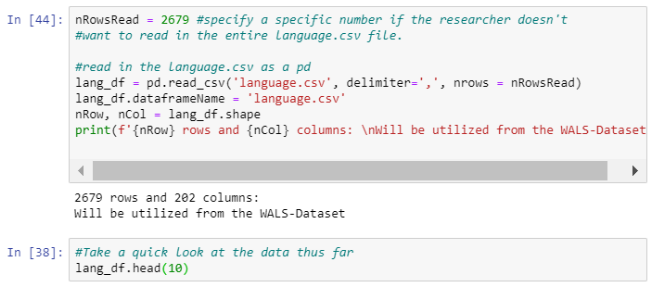
The data has been imported into Python with the following libraries through import statements:



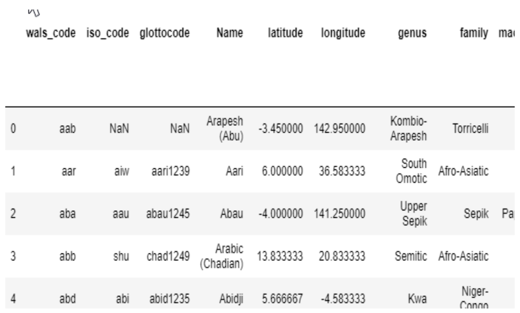
```
#In order to complete the exploratory analysis of the wals dataset, the
#researcher will require the following libraries:
import pandas as pd #for data processing, CSV file input/output
import os # directory structures access
import numpy as np # numpy arrays, linear algebra
import matplotlib.pyplot as plt # this is for plotting
from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.basemap import Basemap
```

Figure 3 (libraries, import)

As noted within the Data exploration section, the are quiet a number of missing values. Upon executing the following code within the IDE the researcher then inspected the missing data to establish the condition to which pandas had imported the missing data, as exampled by Figures 4-5:



```
In [44]: nRowsRead = 2679 #specify a specific number if the researcher doesn't
         #want to read in the entire language.csv file.

         #read in the language.csv as a pd
         lang_df = pd.read_csv('language.csv', delimiter=',', nrows = nRowsRead)
         lang_df.dataframeName = 'language.csv'
         nRow, nCol = lang_df.shape
         print(f'{nRow} rows and {nCol} columns: \nWill be utilized from the WALS-Dataset

         2679 rows and 202 columns:
         Will be utilized from the WALS-Dataset

In [38]: #Take a quick look at the data thus far
         lang_df.head(10)
```

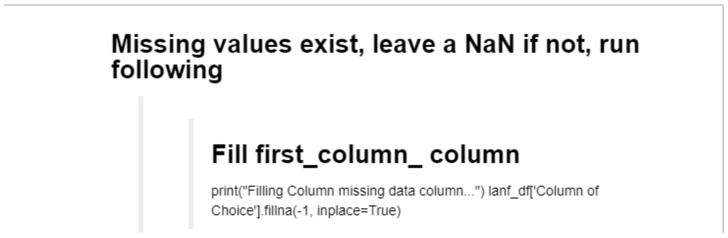


	wals_code	iso_code	glottocode	Name	latitude	longitude	genus	family	ma
0	aab	NaN	NaN	Arapesh (Abu)	-3.450000	142.950000	Kombio-Arapesh	Torricelli	
1	aar	aiw	aari1239	Aari	6.000000	36.583333	South Omotic	Afro-Asiatic	
2	aba	aau	abau1245	Abau	-4.000000	141.250000	Upper Sepik	Sepik	Pa
3	abb	shu	chad1249	Arabic (Chadian)	13.833333	20.833333	Semitic	Afro-Asiatic	
4	abd	abi	abid1235	Abidji	5.666667	-4.583333	Kwa	Niger-Congo	

Figure 4 (establishing pd data frame)

Figure 5 (NaN output)

Having considered the nature of the missing values, neither a mean of the data would suffice as a data fill in, the researcher has decided that the nature of the data requires that these values be left in a state of NaN until such times as NoneType errors start to avail themselves, or TypeErrors. Given that the nature of the data is linguistic in nature, the NaN cells reflect data not obtained during the study. If this becomes a problem the following code could be ran to address the missing values:



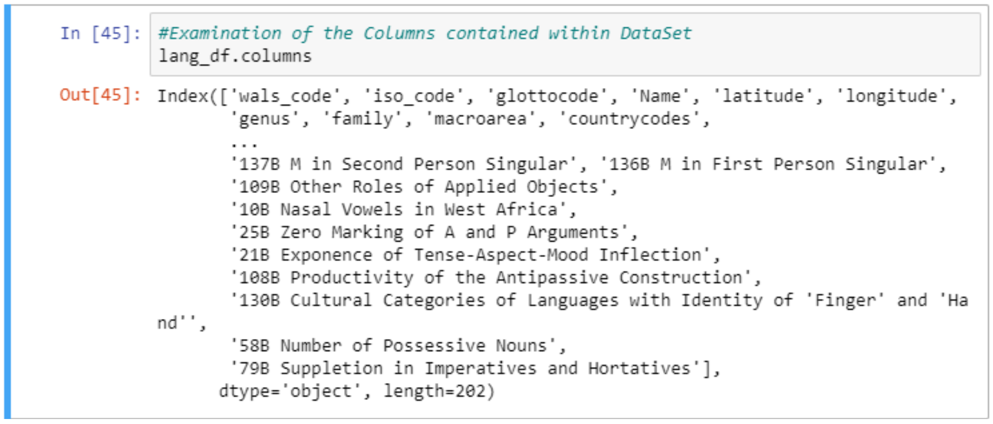
Missing values exist, leave a NaN if not, run following

Fill first_column_ column

print("Filling Column missing data column...") lanf_df['Column of Choice'].fillna(-1, inplace=True)

Figure 6 (Code for .fillna)

The WALS-Dataset per inspection doesn't require much cleaning, other than the determination of how many of the actual columns that want to be brought into the final data examination. During the cleaning process, the columns that will be utilized within the datasets study will needed to be identified, as demonstrated by Figure 7:

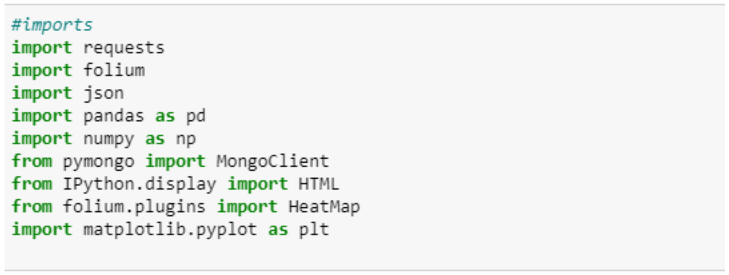


```
In [45]: #Examination of the Columns contained within DataSet
         lang_df.columns

Out[45]: Index(['wals_code', 'iso_code', 'glottocode', 'Name', 'latitude', 'longitude',
                'genus', 'family', 'macroarea', 'countrycodes',
                ...
                '137B M in Second Person Singular', '136B M in First Person Singular',
                '109B Other Roles of Applied Objects',
                '10B Nasal Vowels in West Africa',
                '25B Zero Marking of A and P Arguments',
                '21B Exponence of Tense-Aspect-Mood Inflection',
                '108B Productivity of the Antipassive Construction',
                '130B Cultural Categories of Languages with Identity of 'Finger' and 'Ha
         nd'',
                '58B Number of Possessive Nouns',
                '79B Suppletion in Imperatives and Hortatives'],
               dtype='object', length=202)
```

Methods of Analysis Part II; Semi-Structured Data Analysis

Data Exploration MongoDB

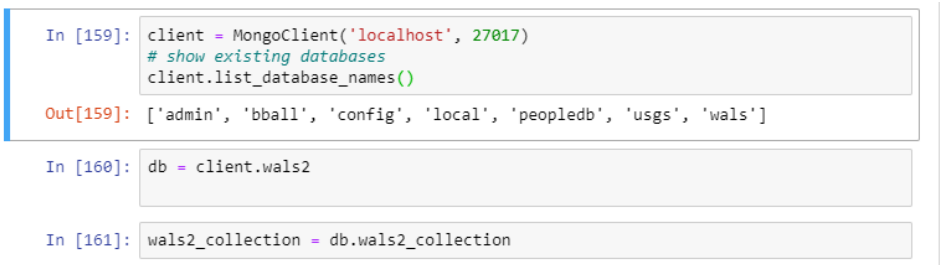
The WALS dataset, obtained to a local machine under the title of languages.csv, was downloaded from the aforesaid website, and was imported into Python 3 via the Jupyter notebook's IDE. To explore the data further the researcher has utilized the following libraries to web scrap and pull in the data from the main website, in order to further understand the vast data contained within the dataset.



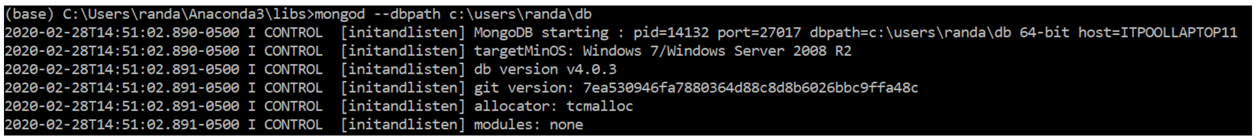
```
#imports
import requests
import folium
import json
import pandas as pd
import numpy as np
from pymongo import MongoClient
from IPython.display import HTML
from folium.plugins import HeatMap
import matplotlib.pyplot as plt
```

Having run the import statements for our libraries in order to bring in the data contained on the wals.info website, the researcher first has to establish a database in order to aggregate and store the data. The database chosen to accomplish this task is MongoDB. The reasoning for this choice, there are a couple; MongoDB is a general purpose, document-based, distributed database built for modern application developers and for the cloud era, as defined by its own website. The ability of this database, in its NOSQL language abilities, results in a versatile, fast place in which to take our unstructured data, and create structure without having to be bound by the strict rules as it relates to relational databases within the SQL language family.

The following examples shows how the setup for the database was accomplished via Jupyter Notebooks and python 3:



```
In [159]: client = MongoClient('localhost', 27017)
          # show existing databases
          client.list_database_names()

Out[159]: ['admin', 'bball', 'config', 'local', 'peopledb', 'usgs', 'wals']

In [160]: db = client.wals2

In [161]: wals2_collection = db.wals2_collection
```

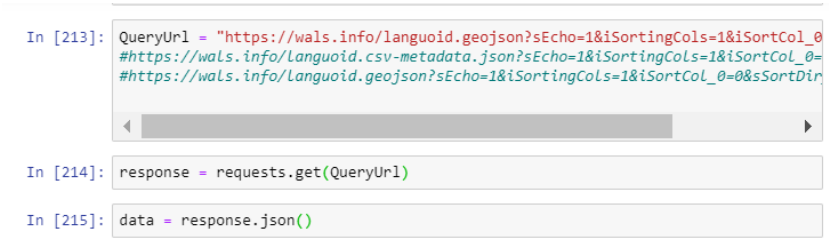


```
(base) C:\Users\randa\Anaconda3\libs>mongod --dbpath c:\users\randa\db
2020-02-28T14:51:02.890-0500 I CONTROL  [initandlisten] MongoDB starting : pid=14132 port=27017 dbpath=c:\users\randa\db 64-bit host=ITPOOLLAPTOP11
2020-02-28T14:51:02.890-0500 I CONTROL  [initandlisten] targetMinOS: Windows 7/Windows Server 2008 R2
2020-02-28T14:51:02.891-0500 I CONTROL  [initandlisten] db version v4.0.3
2020-02-28T14:51:02.891-0500 I CONTROL  [initandlisten] git version: 7ea530946fa7880364d88c8d8b6026bbc9ffa48c
2020-02-28T14:51:02.891-0500 I CONTROL  [initandlisten] allocator: tcmalloc
2020-02-28T14:51:02.891-0500 I CONTROL  [initandlisten] modules: none
```

Successfully establishing the database connection, and building the specific database as it relates to this study, ‘wals’ database, the researcher is now ready and able to start upon the next steps of giving structure to unstructured data, by importing the needed json files in order to build the MongoDB needed, to conduct further exploratory data analysis.

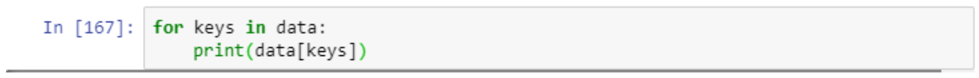
Reviewing the Data, first step in the process is to understand the dataset that is to be cleaned, modeled, and visualized.

In this instance, given that the data that we are pulling in is in geojson form and unstructured data, the following lines of code demonstrate how we create the QueryUrl variable by way of request.get() function as demonstrated:



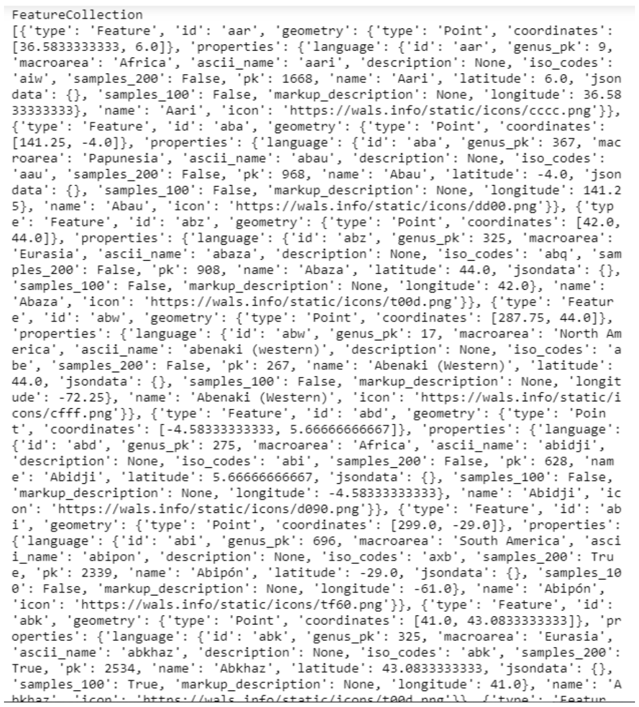
```
In [213]: QueryUrl = "https://wals.info/languoid.geojson?sEcho=1&iSortingCols=1&iSortCol_0
          #https://wals.info/languoid.csv-metadata.json?sEcho=1&iSortingCols=1&iSortCol_0=
          #https://wals.info/languoid.geojson?sEcho=1&iSortingCols=1&iSortCol_0=0&sSortDir

In [214]: response = requests.get(QueryUrl)

In [215]: data = response.json()
```

After bringing in the geojson data into the response variable, a loop was written to explore what type data is contained within the geojson file, via a for loop, as demonstrated:



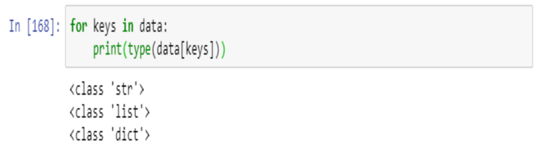
```
In [167]: for keys in data:
              print(data[keys])
```

When ran against the 'data variable' the following visualization demonstrates the keys found within the data:



```
FeatureCollection
[{'type': 'Feature', 'id': 'aar', 'geometry': {'type': 'Point', 'coordinates':
[36.5833333333, 6.0]}, 'properties': {'language': {'id': 'aar', 'genus_pk': 9,
'macroarea': 'Africa', 'ascii_name': 'aari', 'description': None, 'iso_codes':
'aiw', 'samples_200': False, 'pk': 1668, 'name': 'Aari', 'latitude': 6.0, 'json
data': {}, 'samples_100': False, 'markup_description': None, 'longitude': 36.58
33333333}, 'name': 'Aari', 'icon': 'https://wals.info/static/icons/cccc.png'}},
{'type': 'Feature', 'id': 'aba', 'geometry': {'type': 'Point', 'coordinates':
[141.25, -4.0]}, 'properties': {'language': {'id': 'aba', 'genus_pk': 367, 'mac
roarea': 'Papunesia', 'ascii_name': 'abau', 'description': None, 'iso_codes':
'aau', 'samples_200': False, 'pk': 968, 'name': 'Abau', 'latitude': -4.0, 'json
data': {}, 'samples_100': False, 'markup_description': None, 'longitude': 141.2
5}, 'name': 'Abau', 'icon': 'https://wals.info/static/icons/dd00.png'}}, {'typ
e': 'Feature', 'id': 'abz', 'geometry': {'type': 'Point', 'coordinates': [42.0,
44.0]}, 'properties': {'language': {'id': 'abz', 'genus_pk': 325, 'macroarea':
'Eurasia', 'ascii_name': 'abaza', 'description': None, 'iso_codes': 'abq', 'sam
ples_200': False, 'pk': 908, 'name': 'Abaza', 'latitude': 44.0, 'jsondata': {},
'samples_100': False, 'markup_description': None, 'longitude': 42.0}, 'name':
'Abaza', 'icon': 'https://wals.info/static/icons/t00d.png'}}, {'type': 'Featur
e', 'id': 'abw', 'geometry': {'type': 'Point', 'coordinates': [287.75, 44.0]},
'properties': {'language': {'id': 'abw', 'genus_pk': 17, 'macroarea': 'North Am
erica', 'ascii_name': 'abenaki (western)', 'description': None, 'iso_codes': 'a
be', 'samples_200': False, 'pk': 267, 'name': 'Abenaki (Western)', 'latitude':
44.0, 'jsondata': {}, 'samples_100': False, 'markup_description': None, 'longit
ude': -72.25}, 'name': 'Abenaki (Western)', 'icon': 'https://wals.info/static/i
cons/cfff.png'}}, {'type': 'Feature', 'id': 'abd', 'geometry': {'type': 'Poin
t', 'coordinates': [-4.58333333333, 5.66666666667]}, 'properties': {'language':
{'id': 'abd', 'genus_pk': 275, 'macroarea': 'Africa', 'ascii_name': 'abidji',
'description': None, 'iso_codes': 'abi', 'samples_200': False, 'pk': 628, 'nam
e': 'Abidji', 'latitude': 5.66666666667, 'jsondata': {}, 'samples_100': False,
'markup_description': None, 'longitude': -4.58333333333}, 'name': 'Abidji', 'ic
on': 'https://wals.info/static/icons/d090.png'}}, {'type': 'Feature', 'id': 'ab
i', 'geometry': {'type': 'Point', 'coordinates': [299.0, -29.0]}, 'properties':
{'language': {'id': 'abi', 'genus_pk': 696, 'macroarea': 'South America', 'asci
i_name': 'abipon', 'description': None, 'iso_codes': 'axb', 'samples_200': Tru
e, 'pk': 2339, 'name': 'Abipón', 'latitude': -29.0, 'jsondata': {}, 'samples_10
0': False, 'markup_description': None, 'longitude': -61.0}, 'name': 'Abipón',
'icon': 'https://wals.info/static/icons/tf60.png'}}, {'type': 'Feature', 'id':
'abk', 'geometry': {'type': 'Point', 'coordinates': [41.0, 43.0833333333]}, 'pr
operties': {'language': {'id': 'abk', 'genus_pk': 325, 'macroarea': 'Eurasia',
'ascii_name': 'abkhaz', 'description': None, 'iso_codes': 'abk', 'samples_200':
True, 'pk': 2534, 'name': 'Abkhaz', 'latitude': 43.0833333333, 'jsondata': {},
'samples_100': True, 'markup_description': None, 'longitude': 41.0}, 'name': 'A
bkhaz', 'icon': 'https://wals.info/static/icons/t00d.png'}}, {'type': 'Featur
```

Further examination of the keys for loop shows that the type of data keys are as follows:

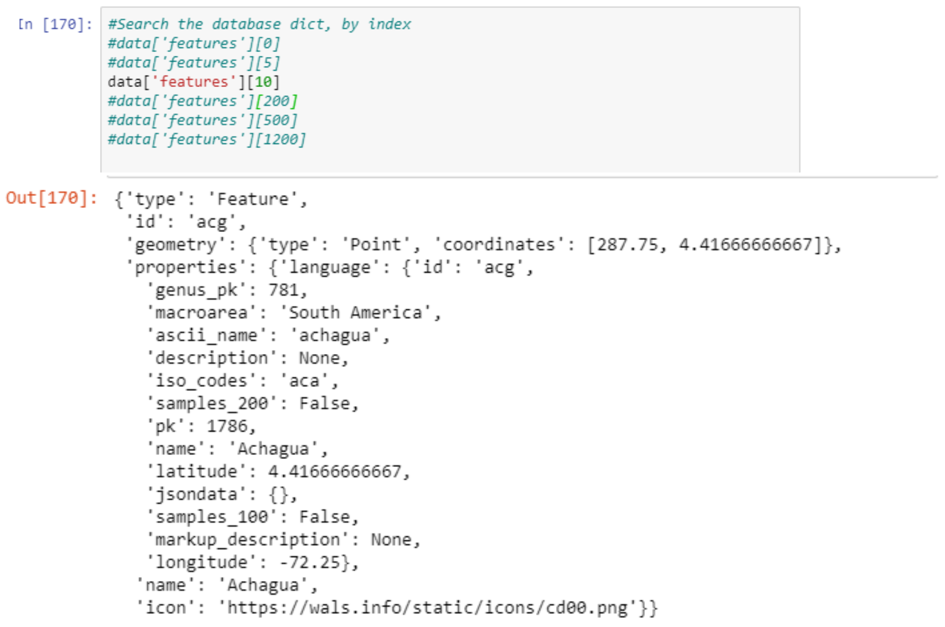


```
In [168]: for keys in data:
              print(type(data[keys]))

<class 'str'>
<class 'list'>
<class 'dict'>
```

Further exploration of the 'data' variable confirms its addition to the database as a class 'dict'.

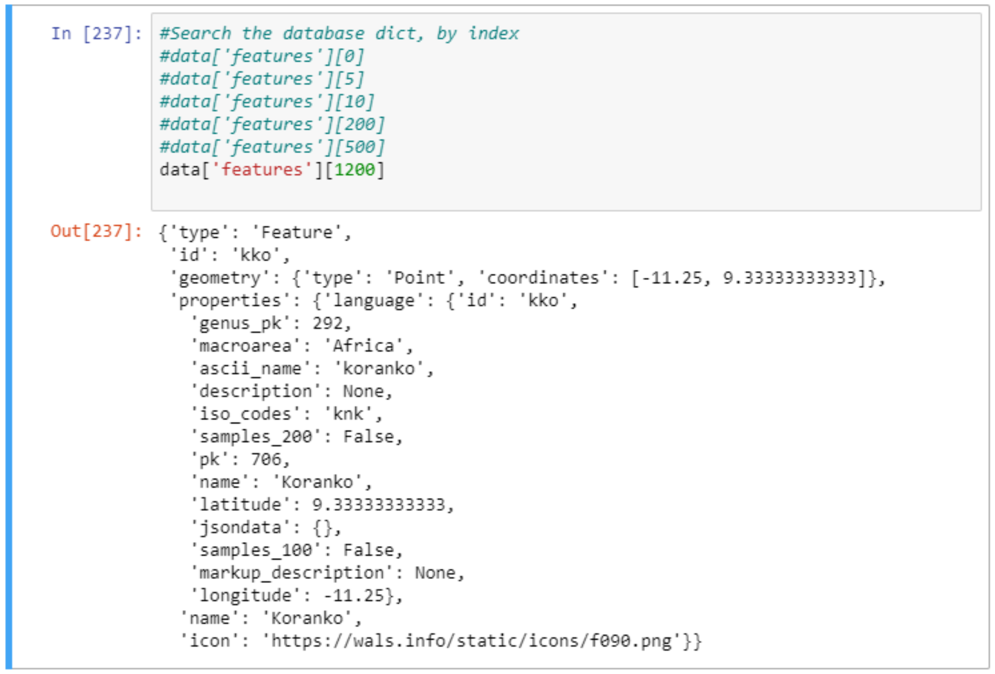
These having been established, we can now test the functionality of the 'data' dictionary, by running against it, so database extracts based up in 'index' of the 'features' key, as the following code extractions demonstrate:

1). *Search the database dict, by index (9)*



```
In [170]: #Search the database dict, by index
          #data['features'][0]
          #data['features'][5]
          data['features'][10]
          #data['features'][200]
          #data['features'][500]
          #data['features'][1200]

Out[170]: {'type': 'Feature',
           'id': 'acg',
           'geometry': {'type': 'Point', 'coordinates': [287.75, 4.41666666667]},
           'properties': {'language': {'id': 'acg',
             'genus_pk': 781,
             'macroarea': 'South America',
             'ascii_name': 'achagua',
             'description': None,
             'iso_codes': 'aca',
             'samples_200': False,
             'pk': 1786,
             'name': 'Achagua',
             'latitude': 4.41666666667,
             'jsondata': {},
             'samples_100': False,
             'markup_description': None,
             'longitude': -72.25},
            'name': 'Achagua',
            'icon': 'https://wals.info/static/icons/cd00.png'}}
```

2). *Search the database dict, by index (1199)*

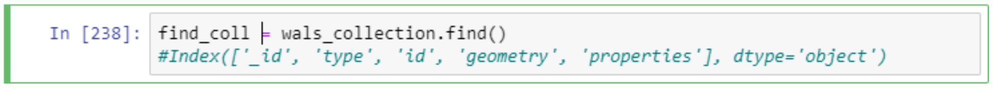


```
In [237]: #Search the database dict, by index
          #data['features'][0]
          #data['features'][5]
          #data['features'][10]
          #data['features'][200]
          #data['features'][500]
          data['features'][1200]

Out[237]: {'type': 'Feature',
           'id': 'kko',
           'geometry': {'type': 'Point', 'coordinates': [-11.25, 9.33333333333]},
           'properties': {'language': {'id': 'kko',
             'genus_pk': 292,
             'macroarea': 'Africa',
             'ascii_name': 'koranko',
             'description': None,
             'iso_codes': 'knk',
             'samples_200': False,
             'pk': 706,
             'name': 'Koranko',
             'latitude': 9.33333333333,
             'jsondata': {},
             'samples_100': False,
             'markup_description': None,
             'longitude': -11.25},
            'name': 'Koranko',
            'icon': 'https://wals.info/static/icons/f090.png'}}
```

This demonstrates that, the geojson file has been successfully imported into python, via pymongo from the MongoDB NoSQL suite, and is now indexable as a data dictionary within python. Each document demonstrates the unique values per entry. As the above examples demonstrate these documents contain all the data points regarding each language entry, from the entire wals online website. The data will need to be further cleaned and examined. It is a vast amount of data.

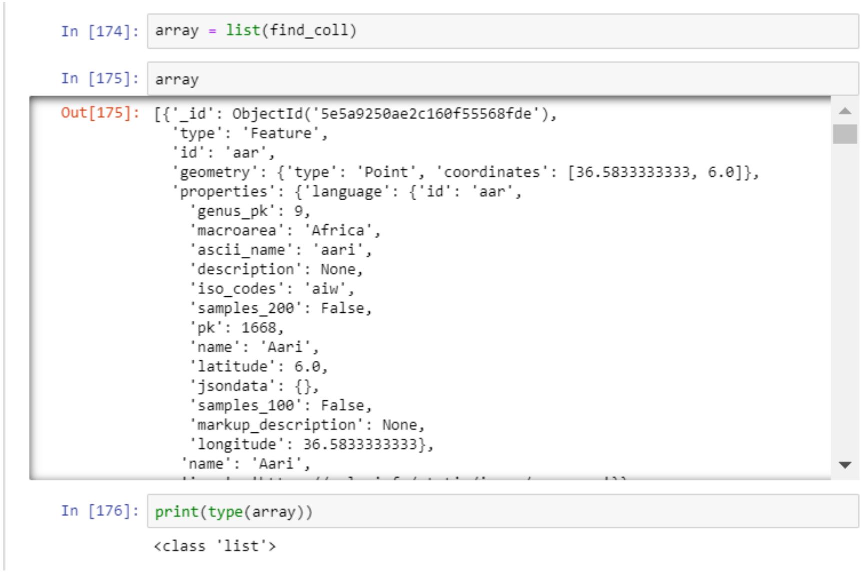
Data Cleaning of Semi-Structured geojson

As noted within the Data exploration section of the previous study of the csv version of the dataset, there are quite several missing values. This is no exception regarding the unstructured data pulled via the geojson. In order to attempt to start to make sense of the data, the researcher first established a document collection,



```
In [238]: find_coll = wals_collection.find()
          #Index(['_id', 'type', 'id', 'geometry', 'properties'], dtype='object')
```

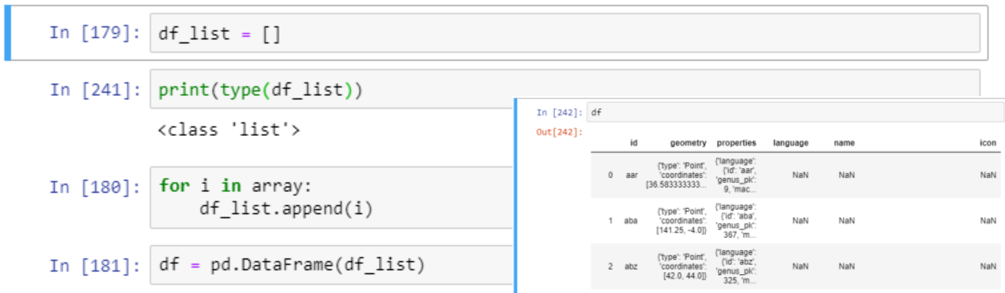
And then, from this collection the researcher created a ‘list’ named array as demonstrated by the following:



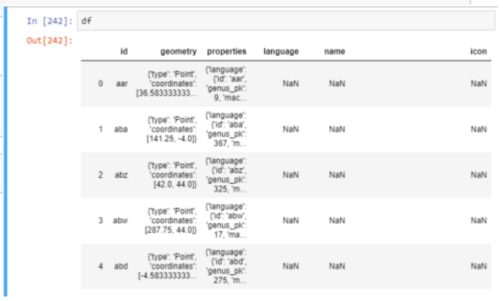
```
In [174]: array = list(find_coll)

In [175]: array

Out[175]: [{'_id': ObjectId('5e5a9250ae2c160f55568fde'),
            'type': 'Feature',
            'id': 'aar',
            'geometry': {'type': 'Point', 'coordinates': [36.5833333333, 6.0]},
            'properties': {'language': {'id': 'aar',
              'genus_pk': 9,
              'macroarea': 'Africa',
              'ascii_name': 'aari',
              'description': None,
              'iso_codes': 'aiw',
              'samples_200': False,
              'pk': 1668,
              'name': 'Aari',
              'latitude': 6.0,
              'jsondata': {},
              'samples_100': False,
              'markup_description': None,
              'longitude': 36.5833333333},
             'name': 'Aari',

In [176]: print(type(array))
          <class 'list'>
```

Python data lists now having been created, and, further structure given to the data, it is prudent to inspect the data, to understand it; its size, its shape, such that we can better understand how to give it further structure, for aggregation. The researcher took the following actions to further clean the data, and add structure:

To not corrupt the array data list, the research created a separate empty list 'df_list[]' and with a simple for loop, appends the array list and then commits the newly created list to a pandas data frame, as demonstrated:



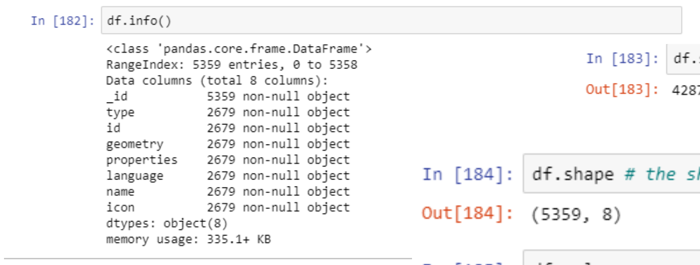
```
In [179]: df_list = []

In [241]: print(type(df_list))
          <class 'list'>

In [180]: for i in array:
              df_list.append(i)

In [181]: df = pd.DataFrame(df_list)
```



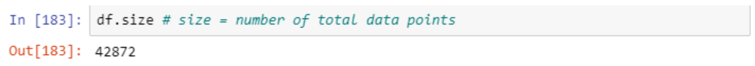
```
In [242]: df
Out[242]:
```

	id	geometry	properties	language	name	icon
0	aar	{'type': 'Point', 'coordinates': [36.583333333...	{'language': {'id': 'aar', 'genus_pk': 9, 'mac...	NaN	NaN	NaN
1	aba	{'type': 'Point', 'coordinates': [141.25, -4.0]}	{'language': {'id': 'aba', 'genus_pk': 367, 'm...	NaN	NaN	NaN
2	abz	{'type': 'Point', 'coordinates': [42.0, 44.0]}	{'language': {'id': 'abz', 'genus_pk': 325, 'm...	NaN	NaN	NaN
3	abw	{'type': 'Point', 'coordinates': [287.75, 44.0]}	{'language': {'id': 'abw', 'genus_pk': 17, 'ma...	NaN	NaN	NaN
4	abd	{'type': 'Point', 'coordinates': [-4.583333333...	{'language': {'id': 'abd', 'genus_pk': 275, 'm...	NaN	NaN	NaN

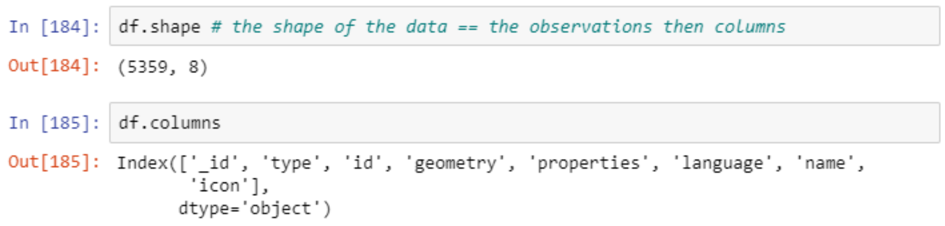
Having created the pandas 'df' data frame, we can now inspect the data to further understand its information, its size, its shape, and its columns, as demonstrated by the following screenshots of those commands ran against the data.



```
In [182]: df.info()
          <class 'pandas.core.frame.DataFrame'>
          RangeIndex: 5359 entries, 0 to 5358
          Data columns (total 8 columns):
          _id           5359 non-null object
          type          2679 non-null object
          id            2679 non-null object
          geometry      2679 non-null object
          properties    2679 non-null object
          language      2679 non-null object
          name          2679 non-null object
          icon          2679 non-null object
          dtypes: object(8)
          memory usage: 335.1+ KB
```



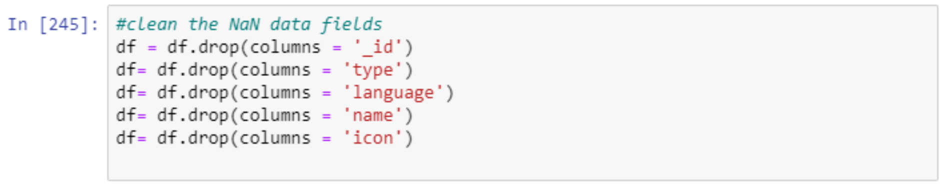
```
In [183]: df.size # size = number of total data points
Out[183]: 42872
```



```
In [184]: df.shape # the shape of the data == the observations then columns
Out[184]: (5359, 8)

In [185]: df.columns
Out[185]: Index(['_id', 'type', 'id', 'geometry', 'properties', 'language', 'name',
                 'icon'],
                dtype='object')
```

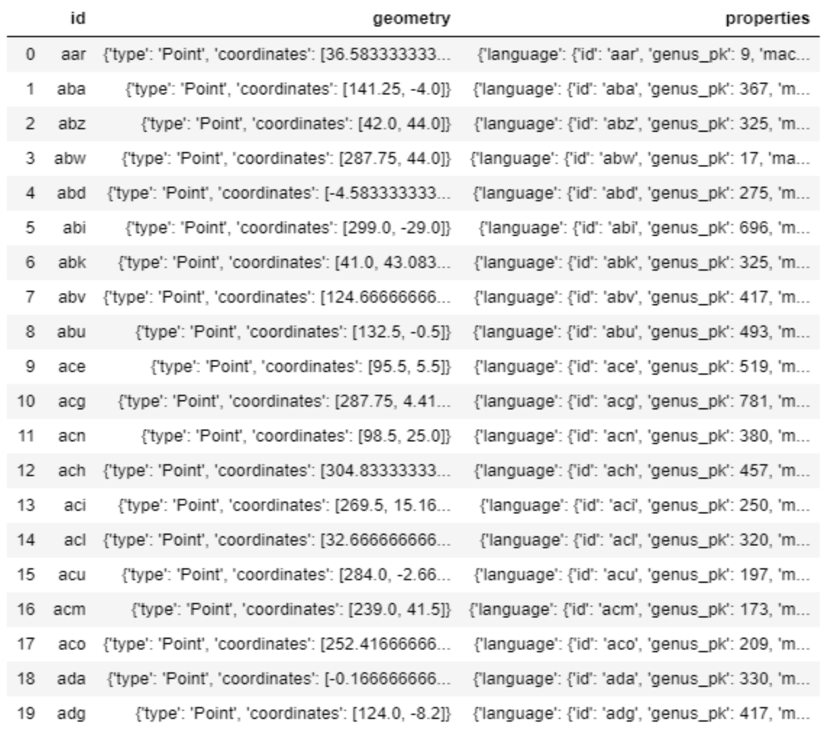
The NaN are numerous so the following cleaning measure we taken against the pandas data frame

Clean the columns →



```
In [245]: #clean the NaN data fields
          df = df.drop(columns = '_id')
          df= df.drop(columns = 'type')
          df= df.drop(columns = 'language')
          df= df.drop(columns = 'name')
          df= df.drop(columns = 'icon')
```

The resulting pandas dataframe:



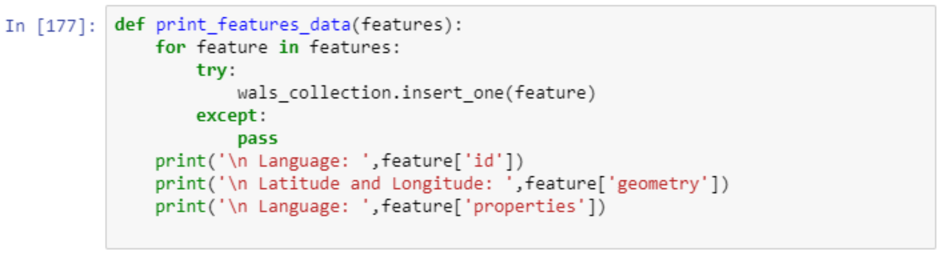
	id	geometry	properties
0	aar	{'type': 'Point', 'coordinates': [36.583333333...	{'language': {'id': 'aar', 'genus_pk': 9, 'mac...
1	aba	{'type': 'Point', 'coordinates': [141.25, -4.0]}	{'language': {'id': 'aba', 'genus_pk': 367, 'm...
2	abz	{'type': 'Point', 'coordinates': [42.0, 44.0]}	{'language': {'id': 'abz', 'genus_pk': 325, 'm...
3	abw	{'type': 'Point', 'coordinates': [287.75, 44.0]}	{'language': {'id': 'abw', 'genus_pk': 17, 'ma...
4	abd	{'type': 'Point', 'coordinates': [-4.583333333...	{'language': {'id': 'abd', 'genus_pk': 275, 'm...
5	abi	{'type': 'Point', 'coordinates': [299.0, -29.0]}	{'language': {'id': 'abi', 'genus_pk': 696, 'm...
6	abk	{'type': 'Point', 'coordinates': [41.0, 43.083...	{'language': {'id': 'abk', 'genus_pk': 325, 'm...
7	abv	{'type': 'Point', 'coordinates': [124.66666666...	{'language': {'id': 'abv', 'genus_pk': 417, 'm...
8	abu	{'type': 'Point', 'coordinates': [132.5, -0.5]}	{'language': {'id': 'abu', 'genus_pk': 493, 'm...
9	ace	{'type': 'Point', 'coordinates': [95.5, 5.5]}	{'language': {'id': 'ace', 'genus_pk': 519, 'm...
10	acg	{'type': 'Point', 'coordinates': [287.75, 4.41...	{'language': {'id': 'acg', 'genus_pk': 781, 'm...
11	acn	{'type': 'Point', 'coordinates': [98.5, 25.0]}	{'language': {'id': 'acn', 'genus_pk': 380, 'm...
12	ach	{'type': 'Point', 'coordinates': [304.83333333...	{'language': {'id': 'ach', 'genus_pk': 457, 'm...
13	aci	{'type': 'Point', 'coordinates': [269.5, 15.16...	{'language': {'id': 'aci', 'genus_pk': 250, 'm...
14	acl	{'type': 'Point', 'coordinates': [32.666666666...	{'language': {'id': 'acl', 'genus_pk': 320, 'm...
15	acu	{'type': 'Point', 'coordinates': [284.0, -2.66...	{'language': {'id': 'acu', 'genus_pk': 197, 'm...
16	acm	{'type': 'Point', 'coordinates': [239.0, 41.5]}	{'language': {'id': 'acm', 'genus_pk': 173, 'm...
17	aco	{'type': 'Point', 'coordinates': [252.41666666...	{'language': {'id': 'aco', 'genus_pk': 209, 'm...
18	ada	{'type': 'Point', 'coordinates': [-0.166666666...	{'language': {'id': 'ada', 'genus_pk': 330, 'm...
19	adg	{'type': 'Point', 'coordinates': [124.0, -8.2]}	{'language': {'id': 'adg', 'genus_pk': 417, 'm...

Research Questions 'comparison questions' semi-structured data

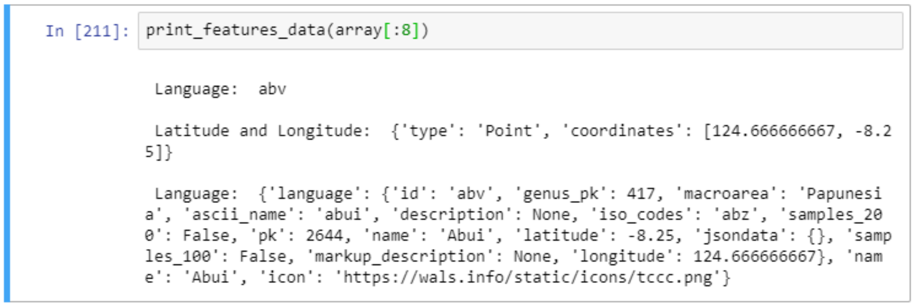
To better understand the data; the researcher chose to process one collection of data and summarize information from several fields.

QUESTION I

In order to better demonstrate the data that is contained within each document, a function was written to essentially grab the feature 'id', 'geometry', and 'properties' and then show that information in a user-friendly manner, as exampled below:

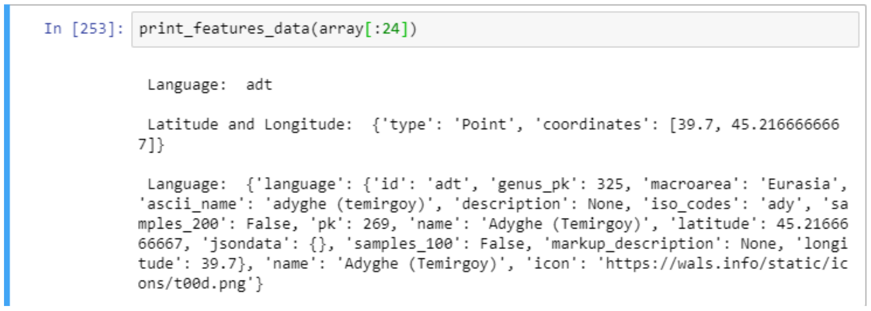


```
In [177]: def print_features_data(features):
              for feature in features:
                  try:
                      wals_collection.insert_one(feature)
                  except:
                      pass
              print('\n Language: ',feature['id'])
              print('\n Latitude and Longitude: ',feature['geometry'])
              print('\n Language: ',feature['properties'])
```



```
In [211]: print_features_data(array[:8])

           Language:  abv

           Latitude and Longitude:  {'type': 'Point', 'coordinates': [124.666666667, -8.2
          5]}

           Language:  {'language': {'id': 'abv', 'genus_pk': 417, 'macroarea': 'Papunesi
          a', 'ascii_name': 'abui', 'description': None, 'iso_codes': 'abz', 'samples_20
          0': False, 'pk': 2644, 'name': 'Abui', 'latitude': -8.25, 'jsondata': {}, 'samp
          les_100': False, 'markup_description': None, 'longitude': 124.666666667}, 'nam
          e': 'Abui', 'icon': 'https://wals.info/static/icons/tccc.png'}
```

index 7 Abui language

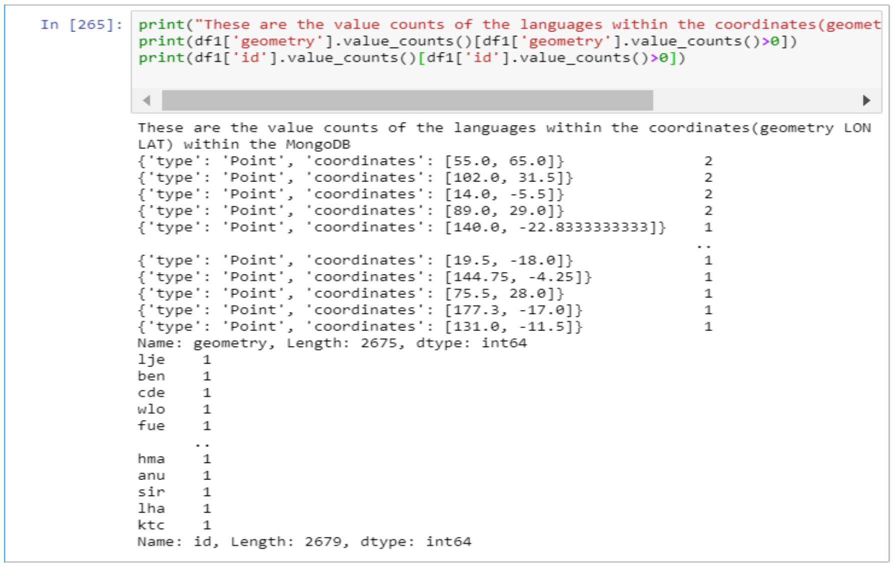


```
In [253]: print_features_data(array[:24])

           Language:  adt

           Latitude and Longitude:  {'type': 'Point', 'coordinates': [39.7, 45.216666666
          7]}

           Language:  {'language': {'id': 'adt', 'genus_pk': 325, 'macroarea': 'Eurasia',
          'ascii_name': 'adyghe (temirgoy)', 'description': None, 'iso_codes': 'ady', 'sa
          mples_200': False, 'pk': 269, 'name': 'Adyghe (Temirgoy)', 'latitude': 45.21666
          66667, 'jsondata': {}, 'samples_100': False, 'markup_description': None, 'longi
          tude': 39.7}, 'name': 'Adyghe (Temirgoy)', 'icon': 'https://wals.info/static/ic
          ons/t00d.png'}
```

index 23 Adyghe language

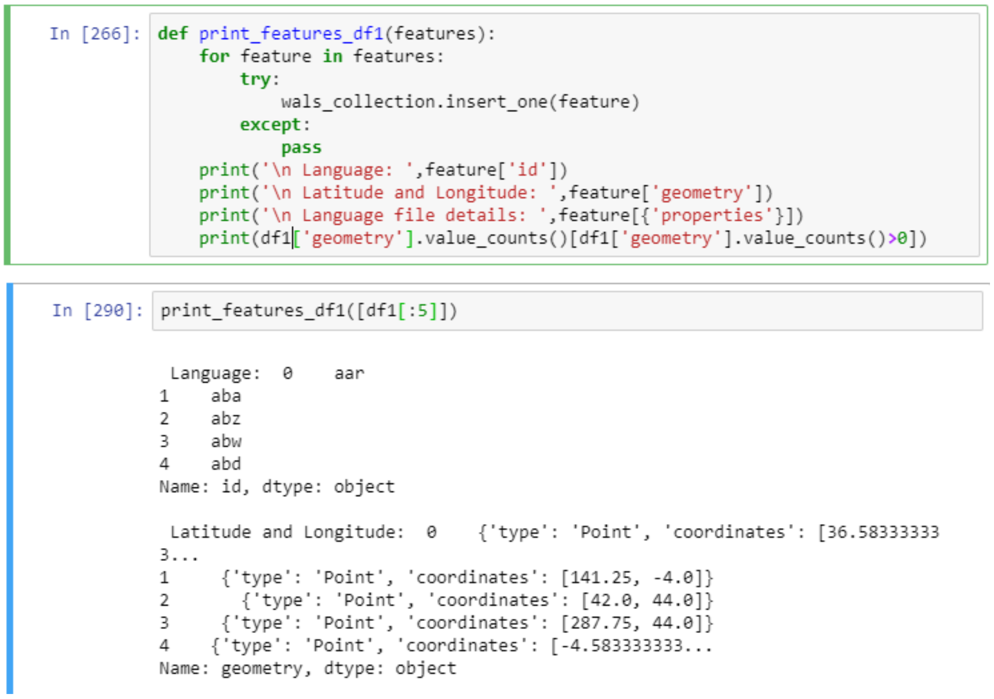
QUESTION II

Another question about the data that could be identified, after offer it structure within our pymongo database, utilizing pandas data frames, we are able to utilize a function to identify whether or not a language group has multiple occurrences at a given longitude and latitude, as demonstrated:



```
In [265]: print("These are the value counts of the languages within the coordinates(geomet
          print(df1['geometry'].value_counts()[df1['geometry'].value_counts()>0])
          print(df1['id'].value_counts()[df1['id'].value_counts()>0])

          These are the value counts of the languages within the coordinates(geometry LON
          LAT) within the MongoDB
          {'type': 'Point', 'coordinates': [55.0, 65.0]}                2
          {'type': 'Point', 'coordinates': [102.0, 31.5]}               2
          {'type': 'Point', 'coordinates': [14.0, -5.5]}                2
          {'type': 'Point', 'coordinates': [89.0, 29.0]}                2
          {'type': 'Point', 'coordinates': [140.0, -22.8333333333]}     1
                                                                       ..
          {'type': 'Point', 'coordinates': [19.5, -18.0]}               1
          {'type': 'Point', 'coordinates': [144.75, -4.25]}             1
          {'type': 'Point', 'coordinates': [75.5, 28.0]}                1
          {'type': 'Point', 'coordinates': [177.3, -17.0]}              1
          {'type': 'Point', 'coordinates': [131.0, -11.5]}              1
          Name: geometry, Length: 2675, dtype: int64
          lje    1
          ben    1
          cde    1
          wlo    1
          fue    1
                ..
          hma    1
          anu    1
          sir    1
          lha    1
          ktc    1
          Name: id, Length: 2679, dtype: int64
```

This is a great representation of the frequency, but it is rather busy, so finally the researcher created this function to identify the needed data, but at a rate of one index within the document at a time.



```
In [266]: def print_features_df1(features):
              for feature in features:
                  try:
                      wals_collection.insert_one(feature)
                  except:
                      pass
              print('\n Language: ',feature['id'])
              print('\n Latitude and Longitude: ',feature['geometry'])
              print('\n Language file details: ',feature[{'properties'}])
              print(df1['geometry'].value_counts()[df1['geometry'].value_counts()>0])
```

```
In [290]: print_features_df1([df1[:5]])
```

```
 Language:  0    aar
1    aba
2    abz
3    abw
4    abd
Name: id, dtype: object

 Latitude and Longitude:  0    {'type': 'Point', 'coordinates': [36.58333333
3...
1     {'type': 'Point', 'coordinates': [141.25, -4.0]}
2       {'type': 'Point', 'coordinates': [42.0, 44.0]}
3     {'type': 'Point', 'coordinates': [287.75, 44.0]}
4    {'type': 'Point', 'coordinates': [-4.583333333...
Name: geometry, dtype: object
```

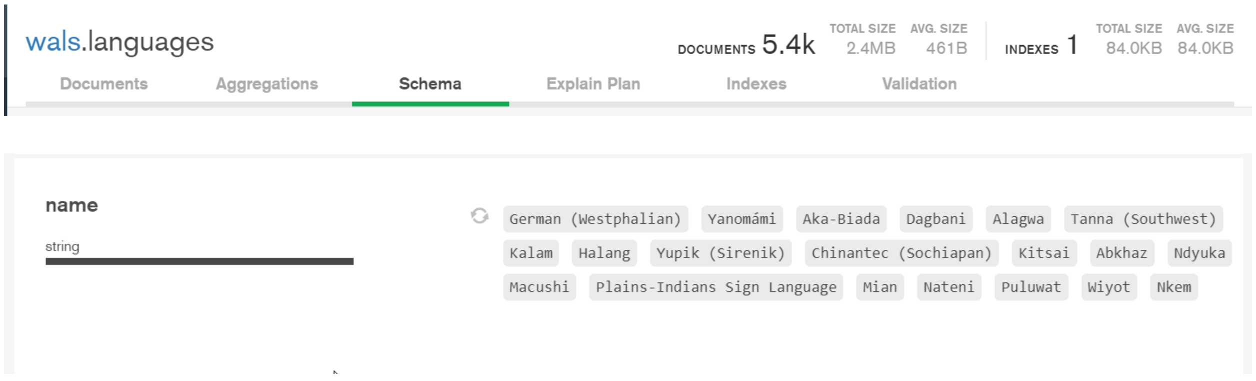
MongoDB

…my kingdom for a little UI …

To recap, we have: collected unstructured data from the following website: https://wals.info/ . After utilizing the pymongo and request libraries, the researcher was able to bring in the geojson file into python 3 Jupyter's notebooks and start to clean and form the unstructured data into a structured form. Then, utilizing python, the researcher was able to create pandas dataframes in order to give the data further structure.

The final section of this research report is the implementation of the unstructured database into a structure MongoDB documents database, utilizing all the work that was just completed, and rendering the results via the MongoDB Compass Program.

The geojson file having been converted into the MongoDB Wals entry, the researcher is able to explore in depth more aspects of the data, as demonstrated in the following screenshots:



wals.languages

DOCUMENTS 5.4k TOTAL SIZE 2.4MB AVG. SIZE 461B

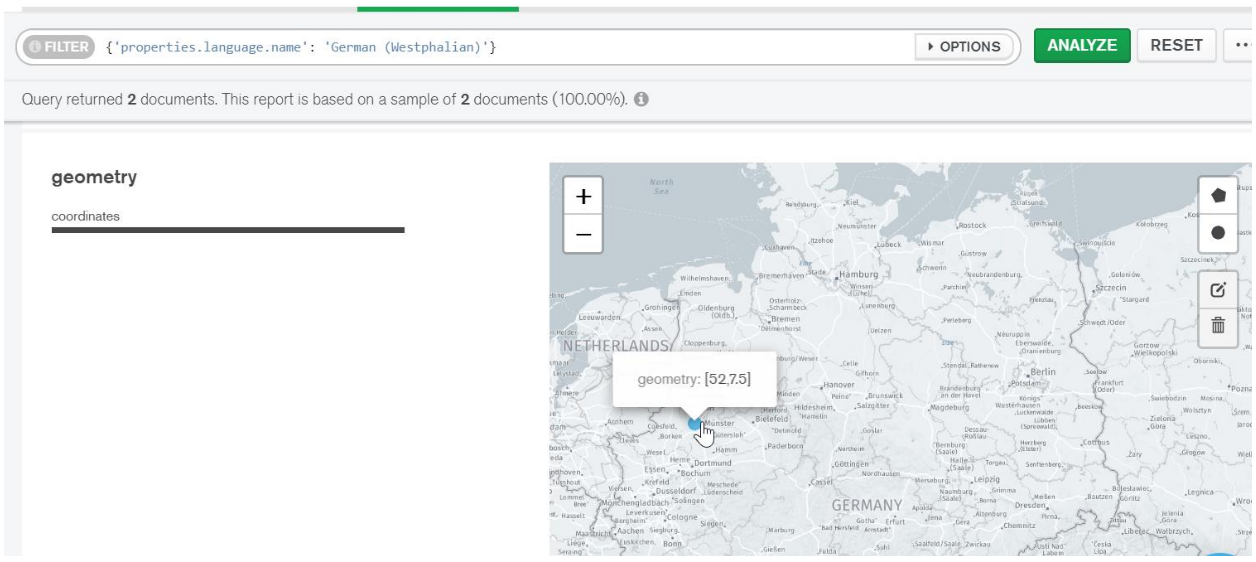
INDEXES 1 TOTAL SIZE 84.0KB AVG. SIZE 84.0KB

Documents | Aggregations | Schema | Explain Plan | Indexes | Validation

name

string

German (Westphalian) Yanomámi Aka-Biada Dagbani Alagwa Tanna (Southwest) Kalam Halang Yupik (Sirenik) Chinantec (Sochiapan) Kitsai Abkhaz Ndyuka Macushi Plains-Indians Sign Language Mian Nateni Puluwat Wiyot Nkem



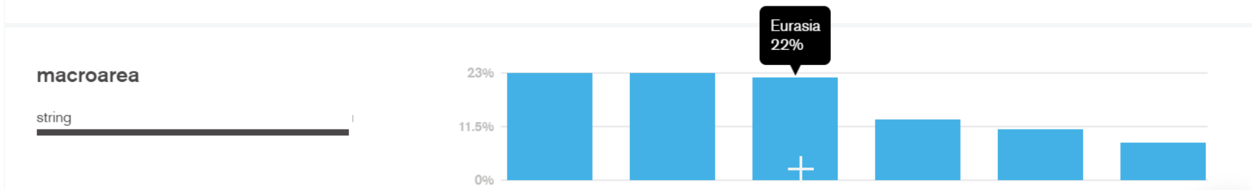
FILTER {'properties.language.name': 'German (Westphalian)'}

Query returned 2 documents. This report is based on a sample of 2 documents (100.00%).

geometry

coordinates

geometry: [52,7.5]



macroarea

string

PART II

RESEARCH QUESTIONS ANALYSIS

Geospatial Analysis:

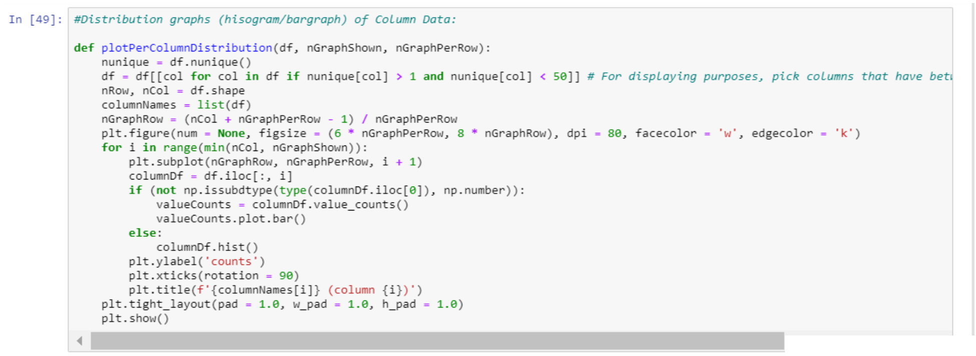
Research Questions

In order to begin to answer research questions, comparison questions about the data, the data transformation and feature extraction process will needs be completed, in order to enhance the data in such as fashion as to increase its likelihood for classification algorithms. This will help to establish any meaningful prediction that the data may provide. The dataset as presented here is rather sorted for the specifics of the beginning of this research project.

Exploratory research questions presented against the data:

What is the frequency distribution of the language's, determinate upon their macroarea?

The following frequency histogram display and code answer this first question:

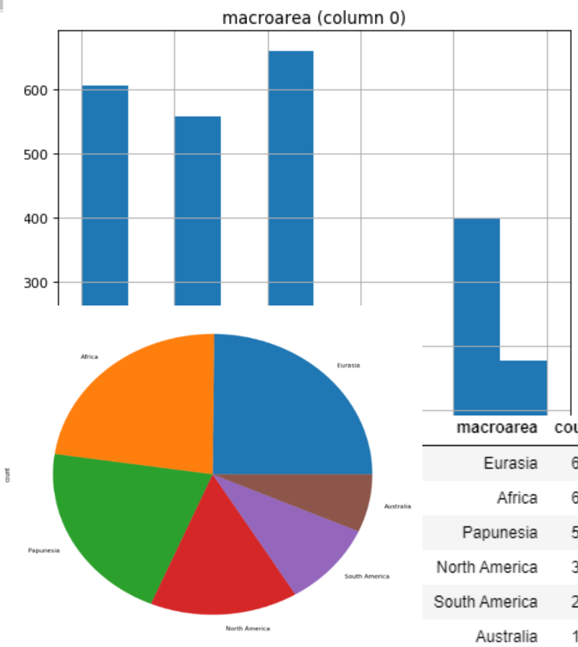


```
In [49]: #Distribution graphs (hisogram/bargraph) of Column Data:

         def plotPerColumnDistribution(df, nGraphShown, nGraphPerRow):
             nunique = df.nunique()
             df = df[[col for col in df if nunique[col] > 1 and nunique[col] < 50]] # For displaying purposes, pick columns that have bet
             nRow, nCol = df.shape
             columnNames = list(df)
             nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow
             plt.figure(num = None, figsize = (6 * nGraphPerRow, 8 * nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')
             for i in range(min(nCol, nGraphShown)):
                 plt.subplot(nGraphRow, nGraphPerRow, i + 1)
                 columnDf = df.iloc[:, i]
                 if (not np.issubdtype(type(columnDf.iloc[0]), np.number)):
                     valueCounts = columnDf.value_counts()
                     valueCounts.plot.bar()
                 else:
                     columnDf.hist()
                 plt.ylabel('counts')
                 plt.xticks(rotation = 90)
                 plt.title(f'{columnNames[i]} (column {i})')
             plt.tight_layout(pad = 1.0, w_pad = 1.0, h_pad = 1.0)
             plt.show()
```

Figure 8 (code)

```
plotPerColumnDistribution(lang_df, 10, 5)
```

Figure 9 (Frequency of Languages per Macro Area)



macroarea	count
Eurasia	660
Africa	607
Papunesia	558
North America	398
South America	257
Australia	177

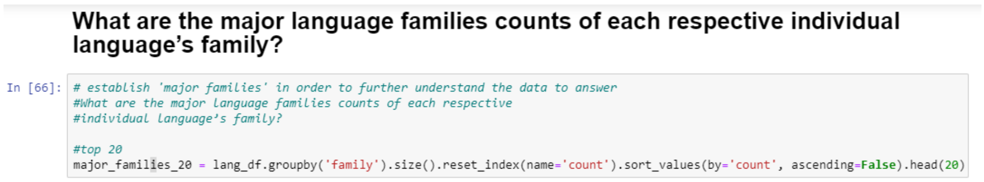
As demonstrated in Figure 9, the greatest frequency of Languages spoken via their respective geographic dispersion is found in the Eurasian land mass. In fact, the entire findings of Figure 9 are intuitive, as per history, it has been demonstrated that the migration of human populations throughout history has been found in the African, Eurasian, and Papunesia landmasses, with the later arrivals to South and North America. Australia doesn’t fit this dispersion however, and in fact should be viewed within the Papunesia frequency count, as history has identified the aboriginals of Australia as having left/been occupants of Australia for the past 50,000 years (PNAS 1). Given that the data has now been sorted into the respective genus family’s macroarea, this will allow for the analysis of this data, from a geospatial perspective.

In order to do this, the language families within the dataset need to be grouped in such a fashion that will assist in further exploration to identify what longitudes and latitudes contain the pertinent dispersion of the Indo-European language family. The determination of this question will assist in further establishment of any potential geocomputational representation, or and clustering that may potentially be identified throughout further analysis.

What are the major language families counts of each respective individual languages family?

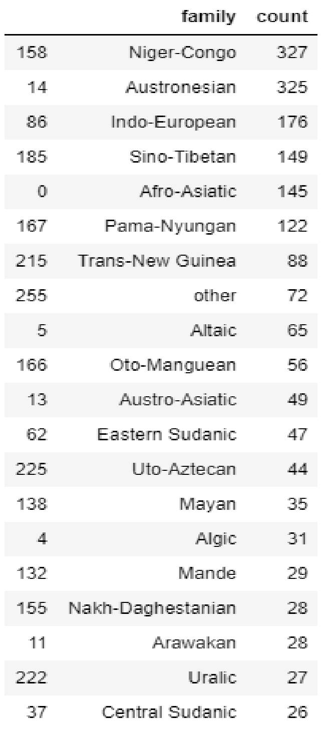
The following Figures, provide the top twenty distributions of the Major Family Groupings, grouped by ‘family’ utilizing a groupby function, based upon the index of name= ‘count’, as demonstrated below:

Figure 10 (code)



What are the major language families counts of each respective individual language’s family?

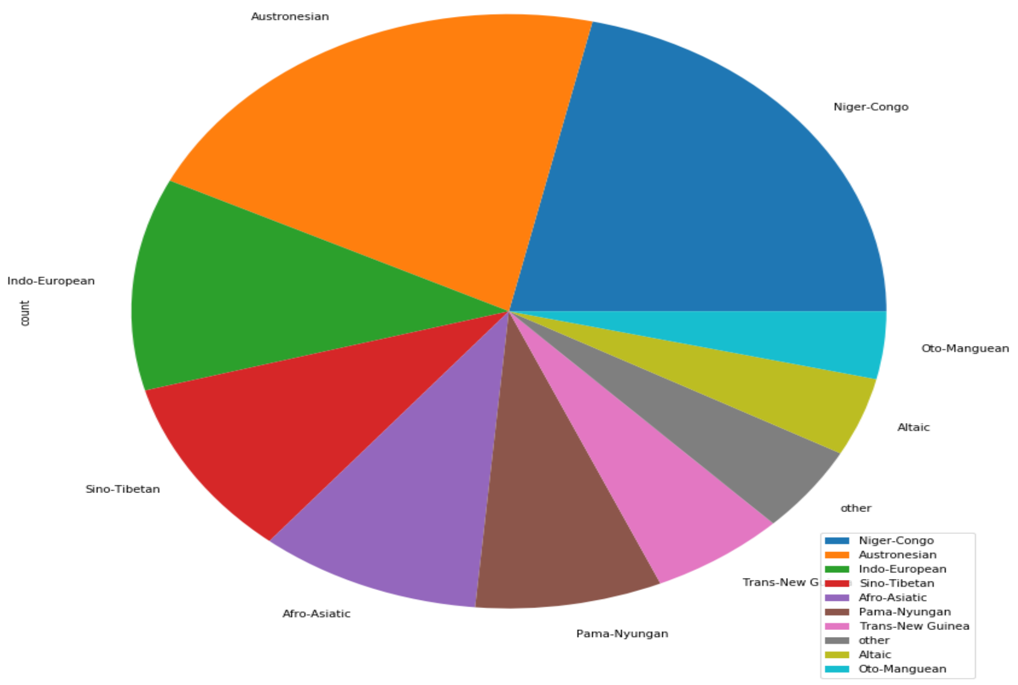
```
In [66]: # establish 'major families' in order to further understand the data to answer
         #What are the major language families counts of each respective
         #individual language’s family?

         #top 20
         major_families_20 = lang_df.groupby('family').size().reset_index(name='count').sort_values(by='count', ascending=False).head(20)
```



	family	count
158	Niger-Congo	327
14	Austronesian	325
86	Indo-European	176
185	Sino-Tibetan	149
0	Afro-Asiatic	145
167	Pama-Nyungan	122
215	Trans-New Guinea	88
255	other	72
5	Altaic	65
166	Oto-Manguean	56
13	Austro-Asiatic	49
62	Eastern Sudanic	47
225	Uto-Aztecan	44
138	Mayan	35
4	Algic	31
132	Mande	29
155	Nakh-Daghestanian	28
11	Arawakan	28
222	Uralic	27
37	Central Sudanic	26

Figure 11 (Major Families, top 20)

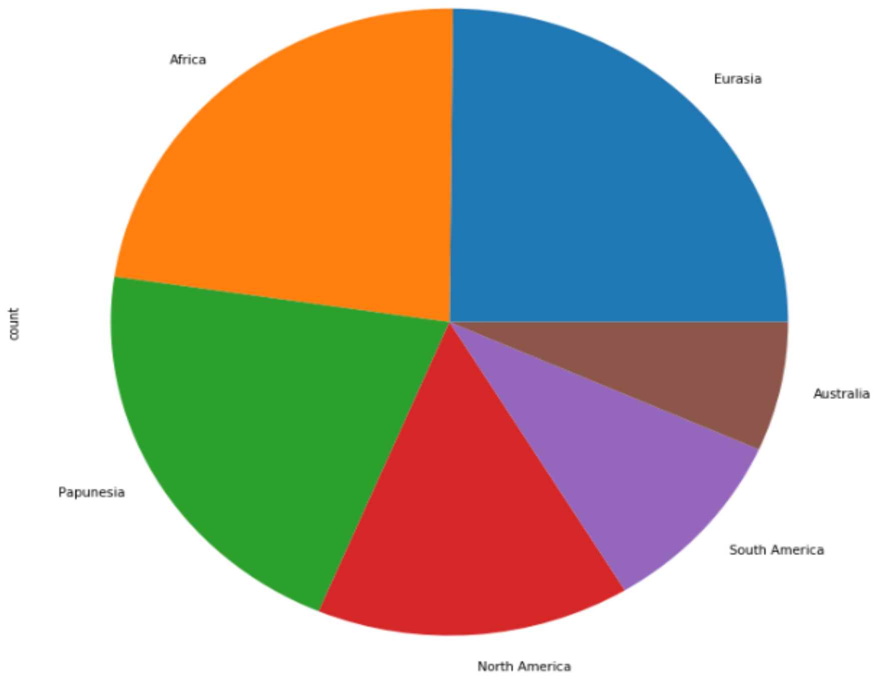
As demonstrated by Figure 10, the Major Family of Languages dispersion has now been established within the data frame, and this particular grouping of the data will assist in further analysis of the dataset in future, such as the geographic dispersion of languages, globally, as demonstrated in Figure 11:

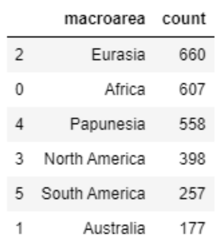


The answers found through above analysis will assist toward the overall goal of the researcher's final project main question:

Can a Geographic spatial analysis of the top three Language Families be identified from the data? Having analyzed the data and establishing what macroarea(s) contained what language family's frequencies, the next aspect of the research project is a geographic spatial analysis of the language families.

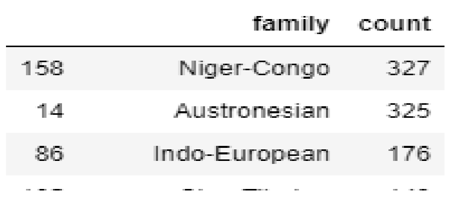
In order to approach this, code was written to establish where the counts of languages were, per 'macroarea,' which has been filter and displayed via their respective continent, as demonstrated:





	macroarea	count
2	Eurasia	660
0	Africa	607
4	Papunesia	558
3	North America	398
5	South America	257
1	Australia	177

Combination of the macroarea count findings, and the top-ranking language families, will allow for the geospatial analysis of the distribution of the top three language family groups, as demonstrated by the frequency count analysis:

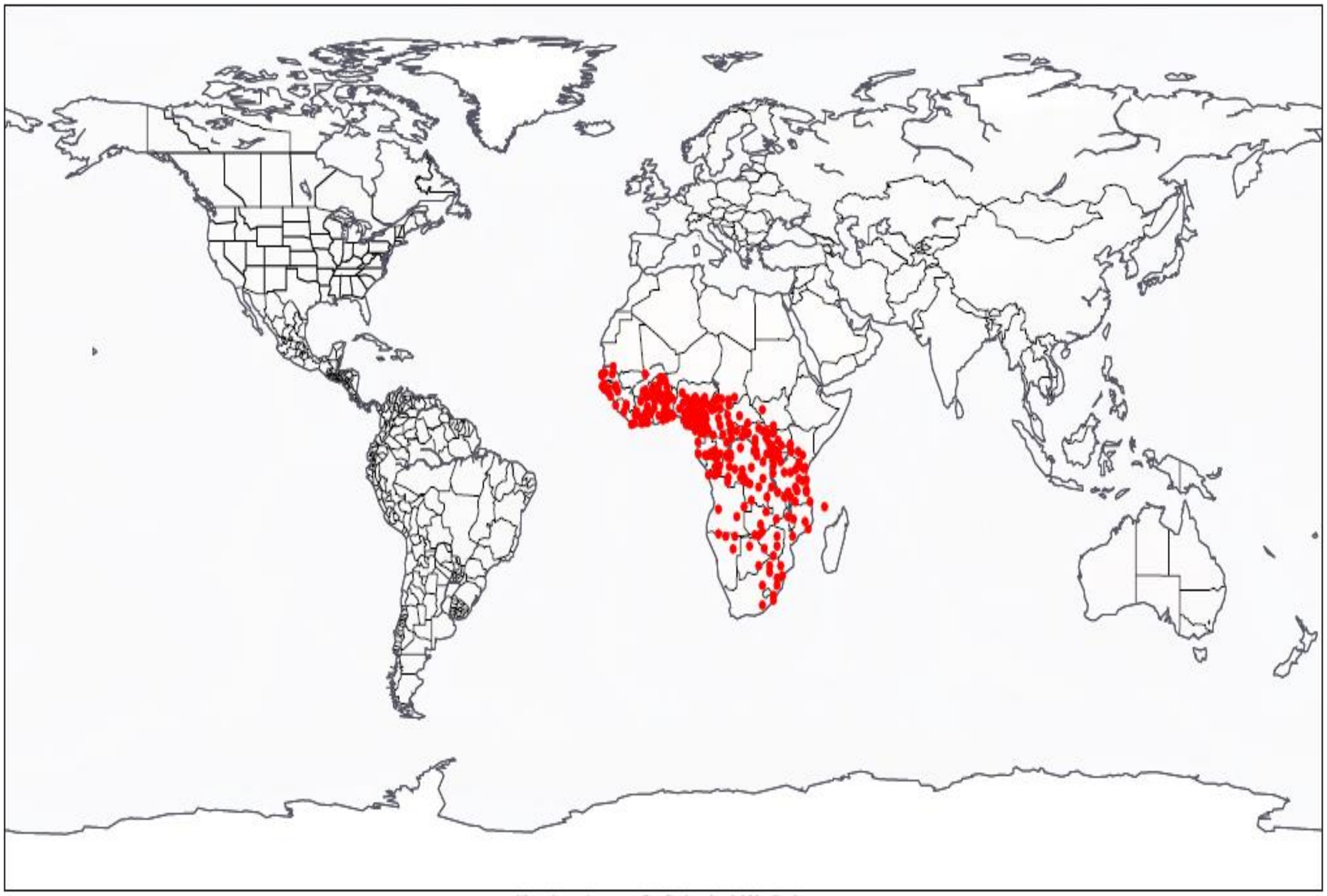


	family	count
158	Niger-Congo	327
14	Austronesian	325
86	Indo-European	176
[illegible]	[illegible]	[illegible]

No.1 Niger-Congo Language Family:

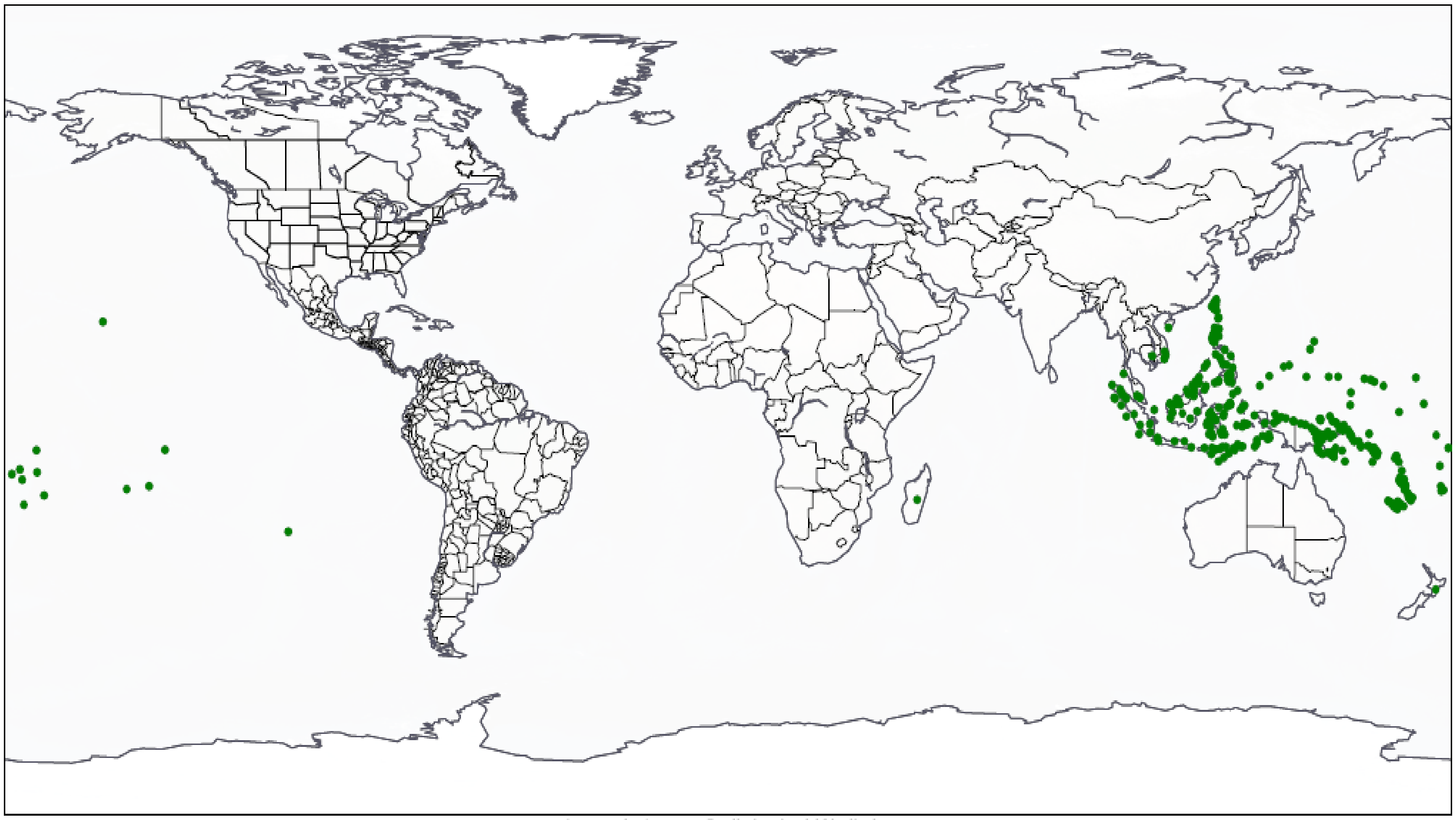
Geographic distribution is found throughout Africa; it is the worlds third largest spoken language family, however, per the dataset, is first in the counts of individual language members to the respective language family. The Niger-Congo Language Family is first, in total individual members to their language family, which is intuitive, humans evolved within the Africa continent.

As demonstrated by the following: we can conduct a Geographic spatial analysis of Niger-Congo Language Family from the data:


Niger-Congo Language Family Geo-Spatial Distribution

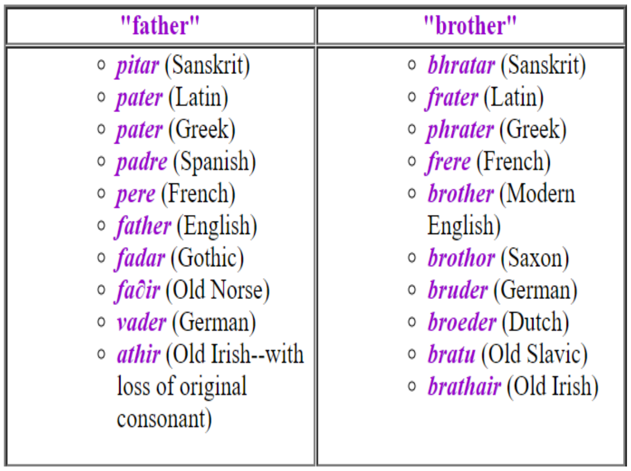
No.2 Austronesian Language Family

Geographic distribution is found throughout: Taiwan, the Malay Peninsula, Maritime Southeast Asia, Madagascar, and the islands of the Pacific Ocean. It is the fifth-largest largest spoken language family, however, per the dataset, it is second in terms of the counts of individual language members to the respective language family. The numbers of individual languages to this language family follow close in-line with the Niger-Congo language family, which can also be seen as intuitive and interesting for the following reasons: the geographic area that this language family is distributed through is massive, thus, allowing for distance and time to create sub-dialects of the proto-language family inheritance, AND, as the first waves of human migration (DNA haplogroups F – Forward) initiated along the very migration pattern demonstrated by the languages geospatial analysis, as seen below:

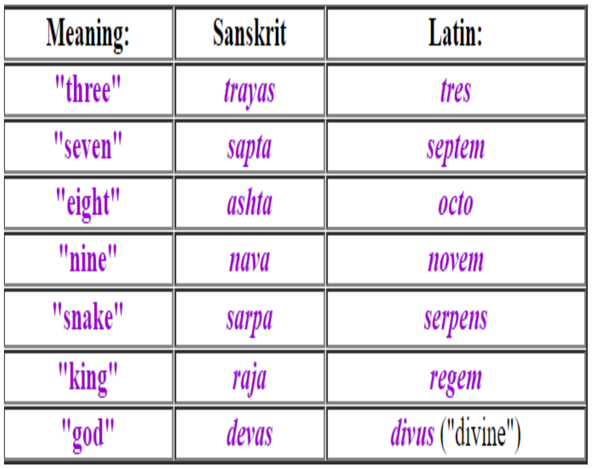

Austronesian Language Family Geo-Spatial Distribution

No.3 Indo-European Language Family

Geographic distribution is found throughout: western Eurasia compromising most of the languages of Europe together with those of the Indian Subcontinent (mostly in the northern portions of the subcontinent) and the Iranian Plateau. It is the second-largest spoken language family, however, per the dataset, it is third in terms of the counts of individual language members to the respective language family. The Language families 'linguistic homeland,' is in dispute, but given the nature of similarities found throughout Latin and Sanskrit and their respective language subdivisions, anthropologist, linguist, and, archeologist are beginning to determine that the 'linguistic homeland,' of the ancient proto-Indo-European language family to be found in the Caucus regions of southern Russia, therein radiating out westerly, and southernly through the passes of the Hindu Kush, into Northern India. Linguistic analysis of the various sub-divisions within the language demonstrates this, as demonstrated by the following:

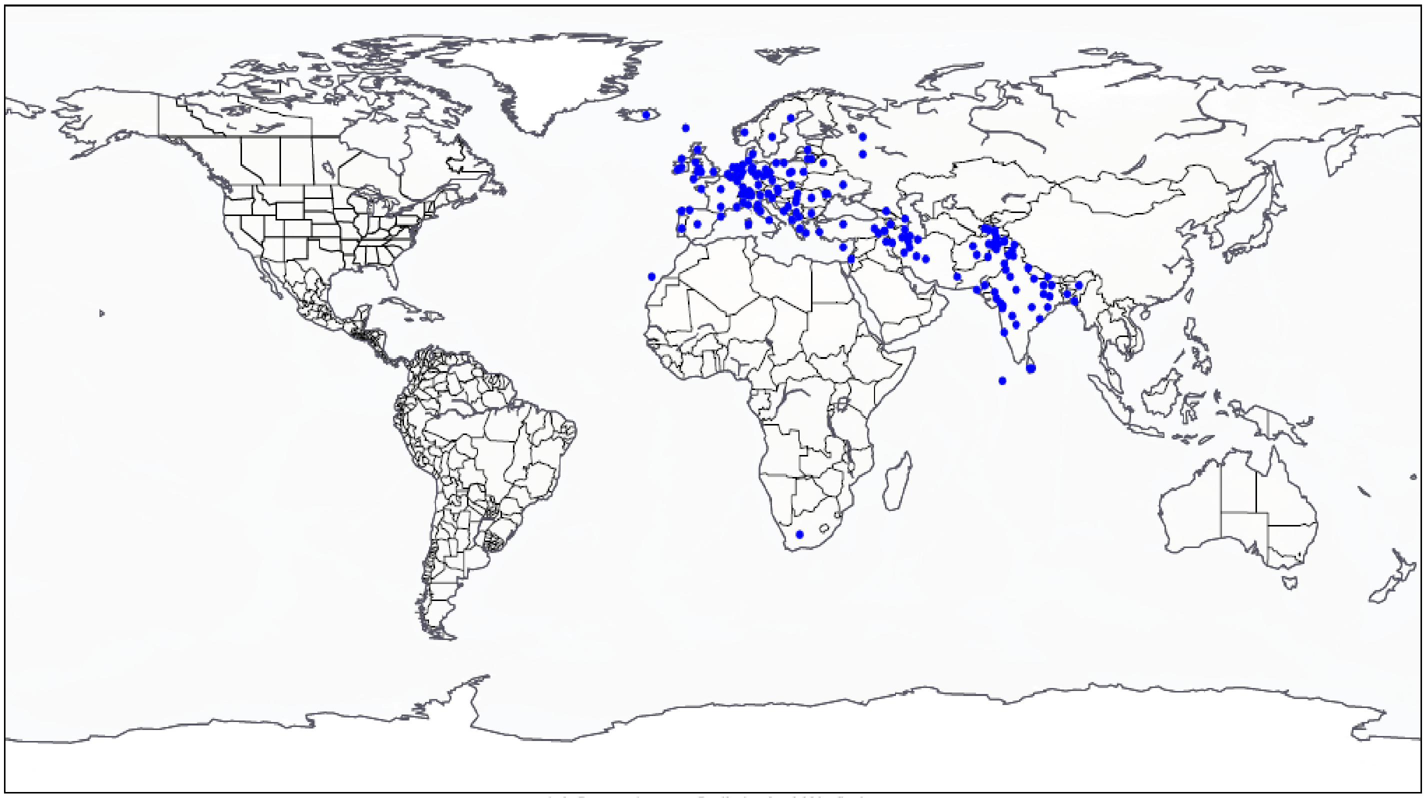


"father"	"brother"
pitar (Sanskrit)	*bhratar* (Sanskrit)
pater (Latin)	*frater* (Latin)
pater (Greek)	*phrater* (Greek)
padre (Spanish)	*frere* (French)
pere (French)	*brother* (Modern English)
father (English)	*brothor* (Saxon)
fadar (Gothic)	*bruder* (German)
faĉir (Old Norse)	*broeder* (Dutch)
vader (German)	*bratu* (Old Slavic)
athir (Old Irish--with loss of original consonant)	*brathair* (Old Irish)

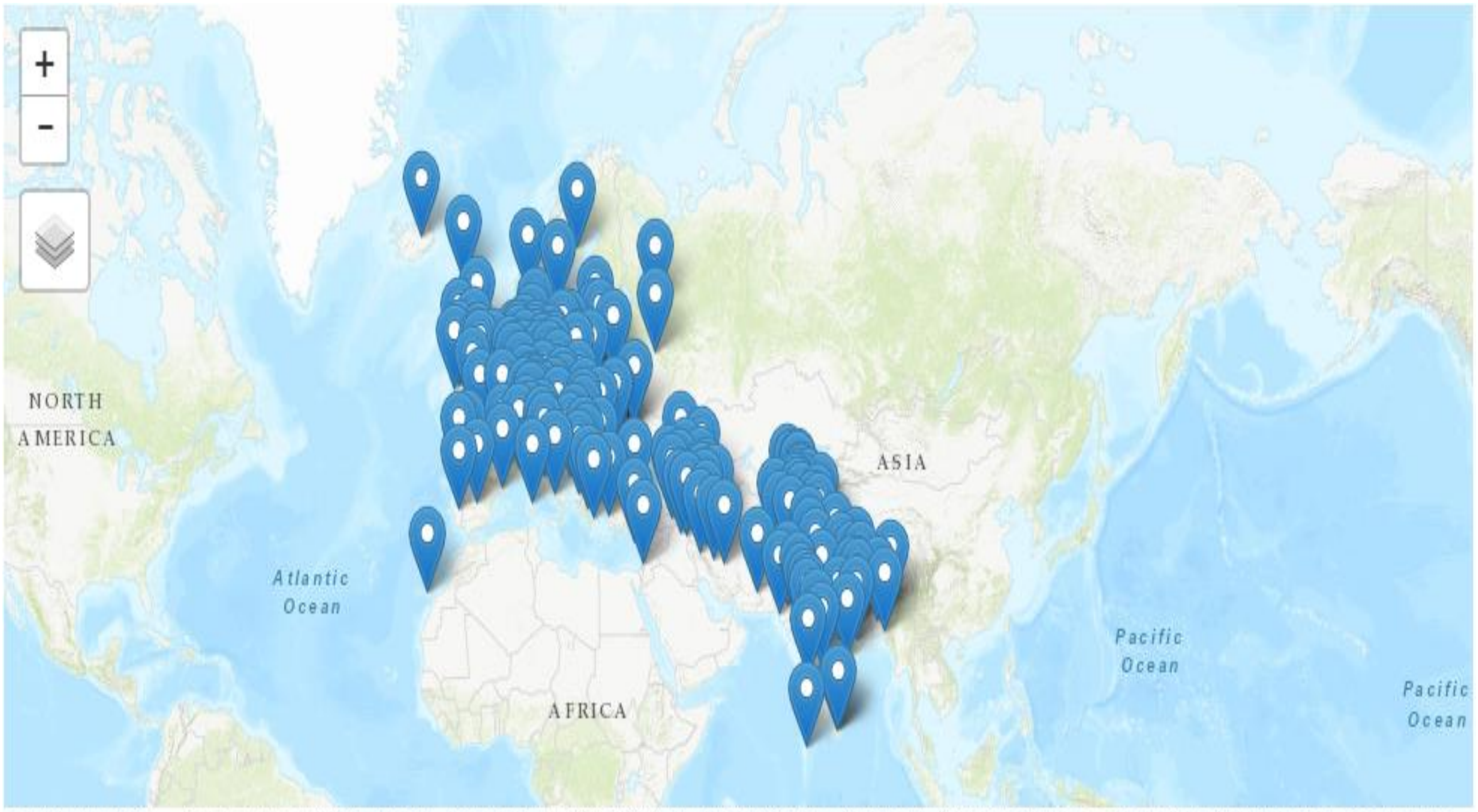


Meaning:	Sanskrit	Latin:
"three"	*trayas*	*tres*
"seven"	*sapta*	*septem*
"eight"	*ashta*	*octo*
"nine"	*nava*	*novem*
"snake"	*sarpa*	*serpens*
"king"	*raja*	*regem*
"god"	*devas*	*divus* ("divine")

Linguistic analysis considered, geospatial analysis of the Indo-European language family shows the nature of its common distribution from its proto homeland, as seen below:



Indo-European Language Family Geo-Spatial Distribution



Leaflet | Tiles © Esri — Esri, DeLorme, NAVTEQ, TomTom, Intermap, iPC, USGS, FAO, NPS, NRCAN, GeoBase, Kadaster NL, Ordnance Survey, Esri Japan, METI, Esri China (Hong Kong), and the GIS User Community

ANGOLA
ZAMBIA
Lusaka
Zambezi
Harare
MOZAMBIQUE
ZIMBABWE
Antananarivo
MADAGASCAR
NAMIBIA
BOTSWANA
Pretoria
Johannesburg
Maputo
Orange
SOUTH AFRICA
Cape Town
Leaflet | Tiles © Esri — Esri, DeLorme, NAVTEQ, TomTom, Intermap, iPC, USGS, FAO, NPS, NRCAN, GeoBase, Kadaster NL, Ordnance Survey, Esri Japan, METI, Esri China (Hong Kong), and the GIS User Community

Interesting observation; the vestige of one of the last Indo-European invasions

WALS DATASET STUDY CONCLUSIONS:

At the onset of the research project, the following questions were presented to be answered:

What is the frequency distribution of the language's, determinate upon their macroarea?

What are the major language families counts of each respective individual language's family?

Can a Geographic spatial analysis of the top three Language Families be identified from the data?

From the dataset obtained, via semi-structured data and structured data, the research project was able to identify the frequency distribution of each language, determinate upon their macroarea, and the researcher was able to demonstrate the shape and size of that data. The research project was also able to identify the major language families counts of each respective individual language's family subdivisions. Finally, the dataset provided for the ability to render a geospatial analysis of the data contained within the dataset, it allowed for the researcher to plot the language families on a geographic representation of the earth, whereby the distribution of the family of languages, per their language family could be demonstrated by static and interactive mappings. In so doing, the research projects conclusion are as follows: the top three language family's totals are intuitive, given the historical, anthropological, and archeological studies that have been conducted, throughout the 19th, 20th, and 21st century. The language families, along with their specific subdivision can be visually demonstrated via geographic spatial analysis. The research projects goals of showing that distribution via the data, and expounding upon the researcher's native spoken language family, the Indo-European language family specific geo spatial analysis was accomplished.

RESEARCH PROJECT DETAILS

GROUP OR INDIVIDUAL PROJECT:

This was an individual project, based upon the work of Applied Data Science master's student Randall Scott Taylor, utilizing the brilliant work done by those at The World Atlas of Language Structures Online.

The project began in late January 2020. The conclusion of this research project: March 11, 2020.

Python Program

The python program written to complete the analysis of this dataset was compiled within the Anaconda suites, Jupyter Notebooks. The following are the libraries utilized within the program:

#In order to complete the exploratory analysis of the wals dataset, the

#researcher will require the following libraries:

import pandas as pd #for data processing, CSV file input/output

import os # directory structures access

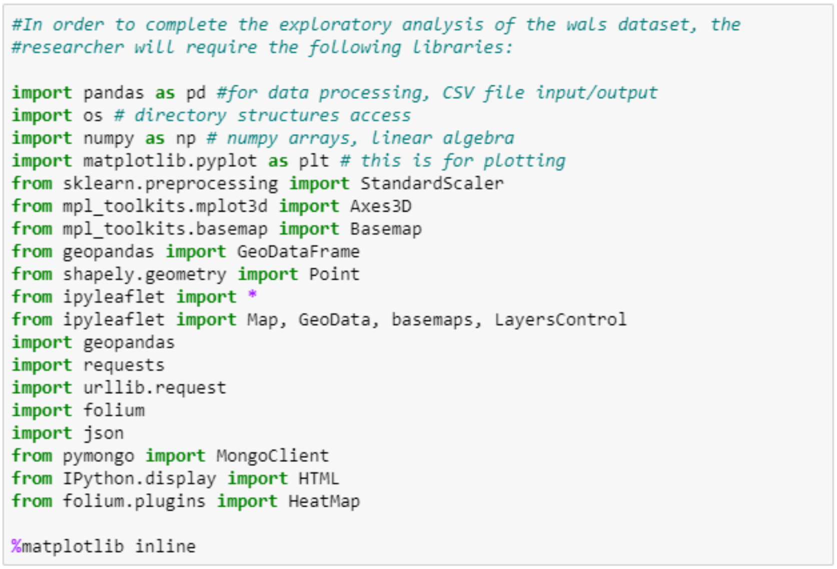
import numpy as np # numpy arrays, linear algebra

import matplotlib.pyplot as plt # this is for plotting

from sklearn.preprocessing import StandardScaler

from mpl_toolkits.mplot3d import Axes3D

from mpl_toolkits.basemap import Basemap



```
#In order to complete the exploratory analysis of the wals dataset, the
#researcher will require the following libraries:

import pandas as pd #for data processing, CSV file input/output
import os # directory structures access
import numpy as np # numpy arrays, linear algebra
import matplotlib.pyplot as plt # this is for plotting
from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.basemap import Basemap
from geopandas import GeoDataFrame
from shapely.geometry import Point
from ipyleaflet import *
from ipyleaflet import Map, GeoData, basemaps, LayersControl
import geopandas
import requests
import urllib.request
import folium
import json
from pymongo import MongoClient
from IPython.display import HTML
from folium.plugins import HeatMap

%matplotlib inline
```

#In order to complete the exploratory analysis of the wals dataset, the

#researcher will require the following libraries:

#imports

import requests

import urllib.request

import folium

import json

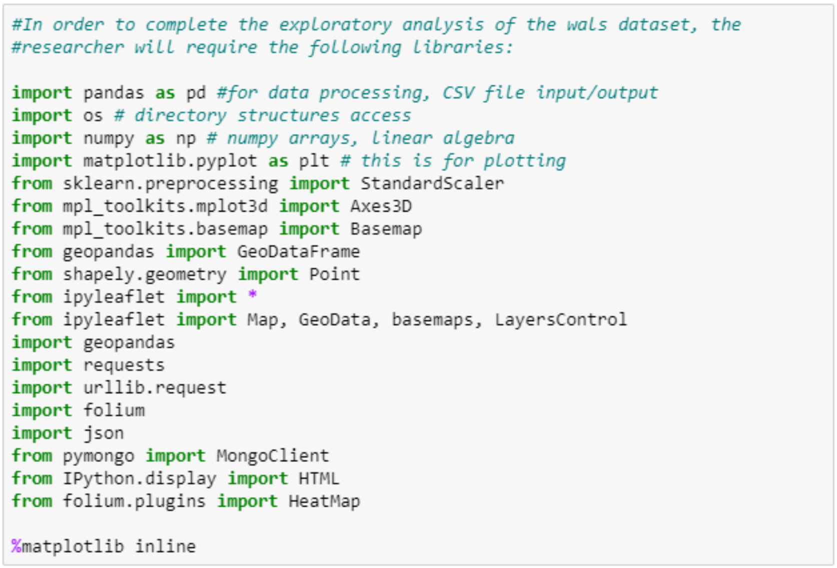
import pandas as pd

import numpy as np

from pymongo import MongoClient

from IPython.display import HTML

from folium.plugins import HeatMap



```
#In order to complete the exploratory analysis of the wals dataset, the
#researcher will require the following libraries:

import pandas as pd #for data processing, CSV file input/output
import os # directory structures access
import numpy as np # numpy arrays, linear algebra
import matplotlib.pyplot as plt # this is for plotting
from sklearn.preprocessing import StandardScaler
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.basemap import Basemap
from geopandas import GeoDataFrame
from shapely.geometry import Point
from ipyleaflet import *
from ipyleaflet import Map, GeoData, basemaps, LayersControl
import geopandas
import requests
import urllib.request
import folium
import json
from pymongo import MongoClient
from IPython.display import HTML
from folium.plugins import HeatMap

%matplotlib inline
```

```
import matplotlib.pyplot as plt
```

The notebook code is organized to follow and flow with the answers rendered within its contents, to answer the questions presented in this study, and the subsequent study to follow.

References

Dryer, Matthew S. & Haspelmath, Martin (eds.) 2013. The World Atlas of Language Structures Online. Leipzig: Max Planck Institute for Evolutionary Anthropology. (Available online at http://wals.info, Accessed on 2015-07-30.)

https://www.pnas.org/content/115/34/8482