CREATING AN ETL WORKFLOW FOR PREDICTING AGRICULTURAL CROP YIELDS

by

Joshua Bonifield

A capstone submitted to Johns Hopkins University in conformity with the requirements for the master's degree

Baltimore, Maryland

December 2021

1. Introduction

Despite vast agricultural land expansion and technical advancements around the world, crop yields will need to increase exponentially to meet the needs of our growing global population (Jiang et al. 2004). Crop monitoring and yield estimation will be crucial to ensure food security, especially while climate change continues to intensify, and our natural resources are depleted. As I learned more about geographically enabled databases and the analysis that can be coupled with it, the more it became clear the opportunity to use this framework to provide insight on agriculture. Traditional crop yield prediction can be time consuming and complex yet creating a data pipeline that is GIS enabled can improve efficiency of the prediction process. Farmers, policy makers, aid organizations, and many more could benefit from the process described below.

2. Statement of Purpose:

The goal of this project was to create and automate an Extract, Transform, and Load (ETL) workflow that ultimately combines worldwide agricultural data into a format that can be used to predict crop yields using machine learning. The resulting, combined dataset is a geographically enabled dataset that can be updated with the latest data at any time. The workflow and subsequent tools are currently restricted to the years 1960 through 2019 and only the countries around the world that have data available from the Food and Agriculture Organization (FAO) and the World Bank. Figure # below shows the countries around the globe that have data available.

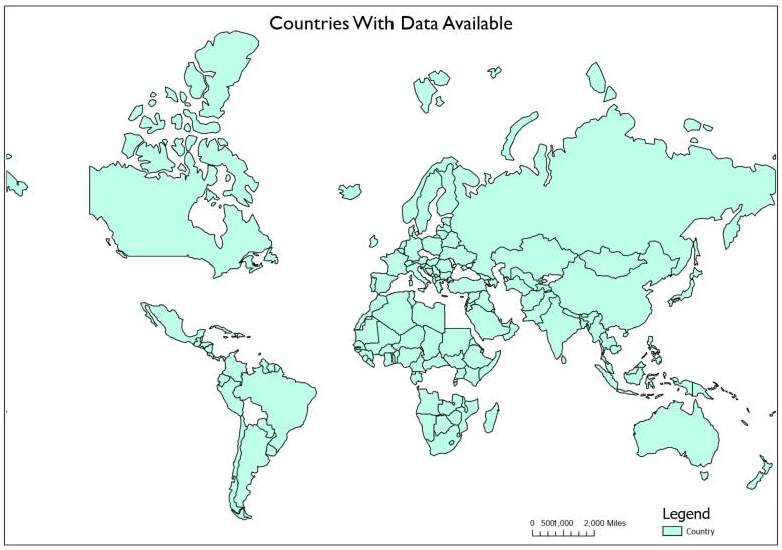

Figure #.

The ETL workflow was originally created using python in a jupyter notebook in order to contain detailed explanations and visualizations throughout. Following the creation and testing of the workflow, ArcGIS Pro tools in the form of python toolboxes were put together emulating the process in the notebook. The process and final combined data are displayed in a Flask web application built in python with HTML, CSS, and JavaScript for stakeholders and decision makers of the World Food Programme (WFP) to easily and quickly understand the data gathered. The python toolboxes were delivered to individuals at the WFP for use in machine learning models.

Crop data will be gathered from the FAO for some of the most important crops produced around the world. These include maize, cassava, rice, soybeans, wheat, potatoes, yams, and sorghum. Other data collected will include the country producing the crop (FAO), the GDP of the country (World Bank), crop yield (FAO), Normalized Difference Vegetation Index (NDVI) (WFP), rainfall (World Bank), temperature (WFP), pesticides (FAO), and population (World Bank). The data will be cleaned based on the principles and techniques described in “Tidy Data” by Hadley Wickham (2014). The data will also be engineered, changing all qualitative values to

quantitative values, based on the needs of the Random forest machine learning model which is described in “Random forests for Global and Regional Crop Yield Predictions” by Jig Han Jeong and Jonathan P. Resop (2016).

3. Data and Study Area:

There are 9 primary datasets that were utilized for this project: world country boundaries, population, rainfall, agricultural land, crop yields, pesticide and fertilizer use, temperature change, and drought, flood, and extreme temperatures. All the datasets were either from the World Bank, Food and Agriculture Organization, or the GeoPandas python module. All the data was manipulated using the Pandas python module.

3.1. World Countries

A boundary dataset was obtained from the GeoPandas python module, which is an open-source project formed with the intention of making working with geospatial data in python easier. The Shapefile is obtained by using the following snippet of GeoPandas method: world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres')). This essentially puts the shapefile into the variable named world. This contains all the administrative boundaries for each country in the world. This is the only spatial dataset that is being used and it is projected using the World Geodetic System 1984, which houses a Geodetic latitude and Geodetic longitude. Projection information was obtained using the GeoPandas crs method which can be seen in Figure # below. For a successful merging of the other datasets with this GeoPandas Shapefile, both need a common variable. The variable or column that appears in every dataset was the country name. To make sure all the countries matched, a list was created from each dataset made up of the country names and then compared. The pandas replace function can be used to change the names

of certain fields to match those in this dataset. For example, in this dataset South Sudan is labeled as S.Sudan and therefore, it is changed in all other datasets to match the shapefile.

```
#find projection
gdf.crs
```

```
<Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

Figure #.

```
#grab a shp from gpd
world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
world.plot(figsize=(10,9), legend = True)
```

```
<AxesSubplot:>
```

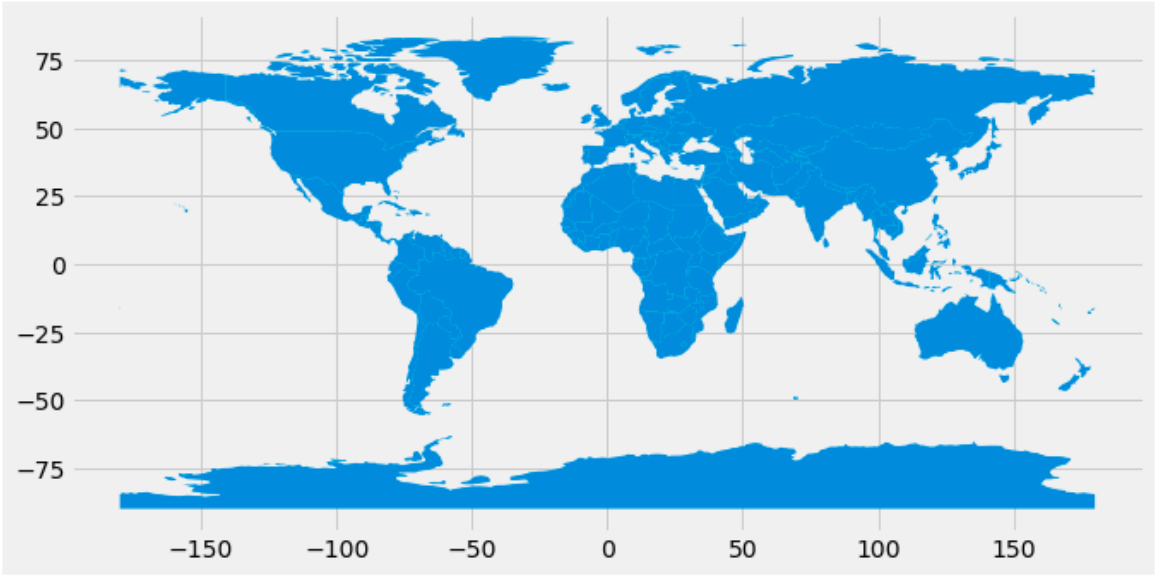

Figure #.

3.2. Population

There are many sources where population could be collected but it turned out to be convenient to obtain the population for each country from 1960 to 2020 from the World Bank API. There is a python module, world_bank_data, that makes it easier to explore the World Bank Indicators published by the World Bank. In order to obtain this dataset, the world_bank_data python module needs to be installed. This was done with the following code: import world_bank_data as wb. Once installed, there is a function, wb.get_indicators(), that shows all the datasets that are available. Once the population dataset has been located, which is called "SP.POP.TOTL", there is another function, wb.get_series, which puts the dataset into a Pandas python series. A series is a one-dimensional array that holds data. To continue to process the data and eventually join this dataset to others, this series was turned into a Pandas data frame, using the to_frame function. The last step for this data is to change the name of the column from "SP.POP.TOTL" to "population" to represent all 16,226 rows of data. Population is considered the count of all residents within a country whether they are a citizen or not, and all the values are mid-year estimates. All other series were added to this data frame.

```
#Population
population = wb.get_series('SP.POP.TOTL', id_or_value='id', simplify_index=True)
```

This is a series, not a dataframe. The next steps takes the population dataset and creates a dataframe

```
#turn population series into dataframe
df_wb = population.to_frame()
# change the population column name
df_wb = df_wb.rename(columns={"SP.POP.TOTL":"population"})
```

Figure #.

3.3 Rainfall

Rainfall was also collected from the World Bank API using the same technique as above to return the data as a Pandas series. There was a total of 16,226 rows in the AG.LND.PRCP.MM dataset representing the total millimeters of rain per year from 1960 to 2020 for every country. This data is measured as the long-term average depth over space and time of annual precipitation in the country. This data was collected from the API as a series; it was then directly placed into the previously created data frame above, df_wb, with a new column titled “rainfall(mm/year)”. This is shown below in Figure #.

```
#Rainfall
perc = wb.get_series('AG.LND.PRCP.MM', id_or_value='id', simplify_index=True)

#add rainfall data in with column names defined
df_wb["rainfall(mm/year)"] = perc
```

Figure #.

3.4 Agricultural Land

Agricultural land, called 'AG.LND.AGRI.K2' was also collected from the World Bank API using the same technique as above to return the data as a pandas series and then add the data to the df_wb data frame. Agricultural land is measured in sq. km and there were 16,226 rows of data from 1960 to 2020 for every country that the World Bank keeps data about. Agricultural land is considered land that is either arable, under permanent crops, or under permanent pastures. Arable land is any land that is under temporary crops, temporary meadows for mowing or pastures, land under market or kitchen gardens, and land temporarily fallow. Land under permanent crops is any land that is cultivated with crops that occupy the land for long periods of time and does not need to be replanted after harvest. Lastly, permanent pasture is land used for five or more years for forage, including natural and cultivated crops.

3.5 Value Added to Agriculture

Value added to agriculture, called 'NV.AGR.TOTL.CD' was also collected from the World Bank API using the same technique as above to return the data as a Pandas series and then add the data to the df_wb data frame. Value added to agriculture is the net output of the agriculture and fishing sector after summing outputs and subtracting the inputs. All records were measured in US dollars and calculated without adjusting for depreciation of assets or depletion of natural resources. This dataset was originally supposed to be included in the final machine learning model but after obtaining and cleaning the data, it was clear that this data would not increase the effectiveness of the model. There were 6,282 records with null values out of the 16,226 total records. The random forest machine learning model also provided insight on the importance of this variable. Figure # shows the top ten most important variables loaded into the model and value added to agriculture was very far down the list, therefore it was decided to be kept out of the model.

```
plt.style.use('fivethirtyeight')
ax = feat_imp.plot(kind='bar', title='Feature Importance')
```

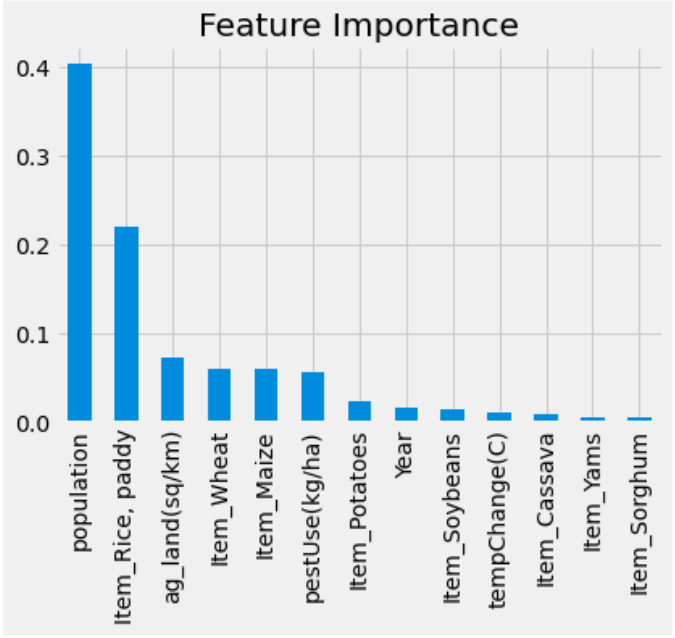

Figure #.

3.6 Crop Yields

Crop yield data was obtained from the FAO. The FAO does not have an active API, where the data can be easily accessed using a python module like the World Bank. This data is gathered by using a URL that points to the webpage containing the data. The URL points to a json object containing a breakdown of all the information. This data is accessible in python using the Requests module, which sends an HTTP request and returns a Response object with all the data. In this case it returns a json object. Once the json object was obtained, all the datasets can be explored. For each dataset there is a file location, a URL that points to a zip file containing all the data. The Requests module is used yet again to send out a Get request for this URL file path. The content is then unzipped, iterated through, and written to a file in a local folder. The ZipFile python module was then used to unzip the file and finally, the local unzipped file path was used

in a Pandas function that creates a data frame. The process is shown in Figure #. This dataset returns a list of crops and the specific yields. However, only rice, potatoes, yams, soybeans, wheat, maize, sorghum, and cassava were used for this analysis. This list was selected as they are some of the most highly produced crops in the analysis.

url where data will be extracted

```
fao_json = 'http://fenixservices.fao.org/faostat/static/bulkdownloads/datasets_E.json'
```

send a request to start searching json

```
response = requests.get(fao_json)
data = response.json()
```

```
#yield file location
yiel = data['Datasets']['Dataset'][47]['FileLocation']
yiel
```

```
#request the zip file
yiel_r = requests.get(yiel, stream=True)

#this is the name of the zip file will be downloaded in local jupyter notebook folder
local_file_yiel = 'yiel_zip.zip'

with open(local_file_yiel, 'wb') as fd:
    for chunk in yiel_r.iter_content(chunk_size=128):
        fd.write(chunk)
```

```
#create a zipfile object from created zip above and extract the contents to local directory
with ZipFile(local_file_yiel, 'r') as zipObj:
    zipObj.extractall()
```

Crop Yield Dataset

```
df_yiel = pd.read_csv('Production_Crops_Livestock_E_All_Data_(Normalized).csv')
df_yiel.head()
```

Figure #.

3.7 Pesticide and Fertilizer Use

The pesticide and fertilizer use dataset was obtained in the same way as the crop yield data from the FAO. Pesticide and fertilizer use is the 7th dataset in the json object and is measured in kilograms per hectare (kg/ha). This dataset contains around 30,740 records, and for

each country and year it contains the kg/ha of three different fertilizers or pesticides including nitrogen (N), phosphorus (P2O5), and potassium (K2O) based. This is a time-series dataset from 1961 to present day. For ease of merging and manipulating the future dataset, an aggregation took place of all the pesticides and fertilizers based on their year and country. This resulted in a dataset that only had one total number of pesticides and fertilizers used for every year for each country resembling the other datasets. Figure #. shows how this aggregation happened via the groupby and agg functions within Pandas.

```
#Total amount of pesticides used each year
df_pest = df_pest.groupby(['Area', 'Year']).agg({'Value': 'sum'})
df_pest.reset_index(inplace=True)
df_pest.rename(columns = {'Value':'pestUse(kg/ha)'}, inplace = True)
df_pest
```

	Area	Year	pestUse(kg/ha)
0	Afghanistan	1961	0.14
1	Afghanistan	1962	0.14
2	Afghanistan	1963	0.14
3	Afghanistan	1964	0.14
4	Afghanistan	1965	0.14
...	...	...	...
10489	Zimbabwe	2015	19.17
10490	Zimbabwe	2016	31.78
10491	Zimbabwe	2017	35.71
10492	Zimbabwe	2018	32.39
10493	Zimbabwe	2019	32.39

Figure #.

3.8 Temperature Change

Temperature change was also gathered in the same way as the pesticide and fertilizer use and crop yield data from the FAO. Temperature change is the 15th dataset in the json object and

is measured as the mean surface temperature change by country for each month from 1961 to present year. This data is collected by the FAO from the National Aeronautics and Space Administration Goddard Institute for Space Studies. The returned data included around 537,370 records and included a record for each month, which was changed to have only one record for every year for each country. This was completed by using the same group by and agg Pandas functions used for the pesticide and fertilizer use dataset. Once calculated, it is then divided by 12 to get the average temperature change per year. Figure #. shows this process.

```
#sum the temp change for each year then average that number
df_temp = df_temp.groupby(['Area', 'Year']).agg({'Value': 'sum'})
df_temp.reset_index(inplace=True)
df_temp['Value'] = df_temp['Value']/12
df_temp.rename(columns = {'Value':'tempChange(C)'}, inplace = True)
df_temp
```

	Area	Year	tempChange(C)
0	Afghanistan	1961	1.659750
1	Afghanistan	1962	1.332000
2	Afghanistan	1963	2.886083
3	Afghanistan	1964	0.326083
4	Afghanistan	1965	1.539583
...	...	...	...
15800	Zimbabwe	2016	2.904667
15801	Zimbabwe	2017	1.539667
15802	Zimbabwe	2018	2.082167
15803	Zimbabwe	2019	2.956333
15804	Zimbabwe	2020	1.916833

Figure #.

3.9 Droughts, floods, and extreme temperatures

As our climate continues to increase in intensity and frequency, the larger the potential impact it will have on agriculture. The World Bank API keeps a droughts, floods, and extreme temperatures index, which represents the annual average percentage of the population that is

affected by those events listed. The World Bank considers a drought to be an extended period of below average precipitation that results in a decrease in water supply. Droughts are an important factor because they can lead to a decrease in yields, increase the likelihood of famine due to limited supply of drinking water, and cause a lack of natural and artificial irrigation. A flood is defined by the World Bank as "a significant rise of the water level in a stream, lake, reservoir, or coastal region" (2019). Lastly, extreme temperatures are considered either long bouts of above or below average temperatures. Extreme temperatures can cause damage to agriculture, infrastructure, and property. This dataset is measured as the average percentage of population impacted, which is the "the number of people injured, left homeless or requiring immediate assistance during a period of emergency resulting from a natural disaster; it can also include displaced or evacuated people." This dataset was also not used because there were 16,058 null records out of the 16,226. In exchange, the temperature change and rainfall data were added to the model.

GIS Anticipated Results:

A set of python tools that can be used to extract agricultural data from the sources listed above, clean that data, and it be available and ready to predict crop yields for countries around the world. This will be available for download on GitHub and Johns Hopkins University's ArcGIS Online page. A GIS hosted feature layer created from the workflow above will be available on the Johns Hopkins University's ArcGIS Online page and held internally at the WFP and can be updated periodically from the using a tool. A web application will show the final dataset as well as the option to download the cleaned data to run the Random forest machine learning model. The main purpose for this project is so that researchers working on food and agriculture related topics can have readily available data to download and study. This will save

time, money and result in more informed and timely decisions regarding aid to these countries involved.

4. GIS Technology and Methods:

4.1. Tools

All the data was processed using python 3.7, utilizing many tools from two open-source software libraries, Pandas and sklearn modules. The final combined dataset was displayed using the GeoPandas, MatPlotLib, and ipywidgets module. The python toolboxes that are used in ArcGIS Pro were created using the arcpy python module.

4.2 Workflow

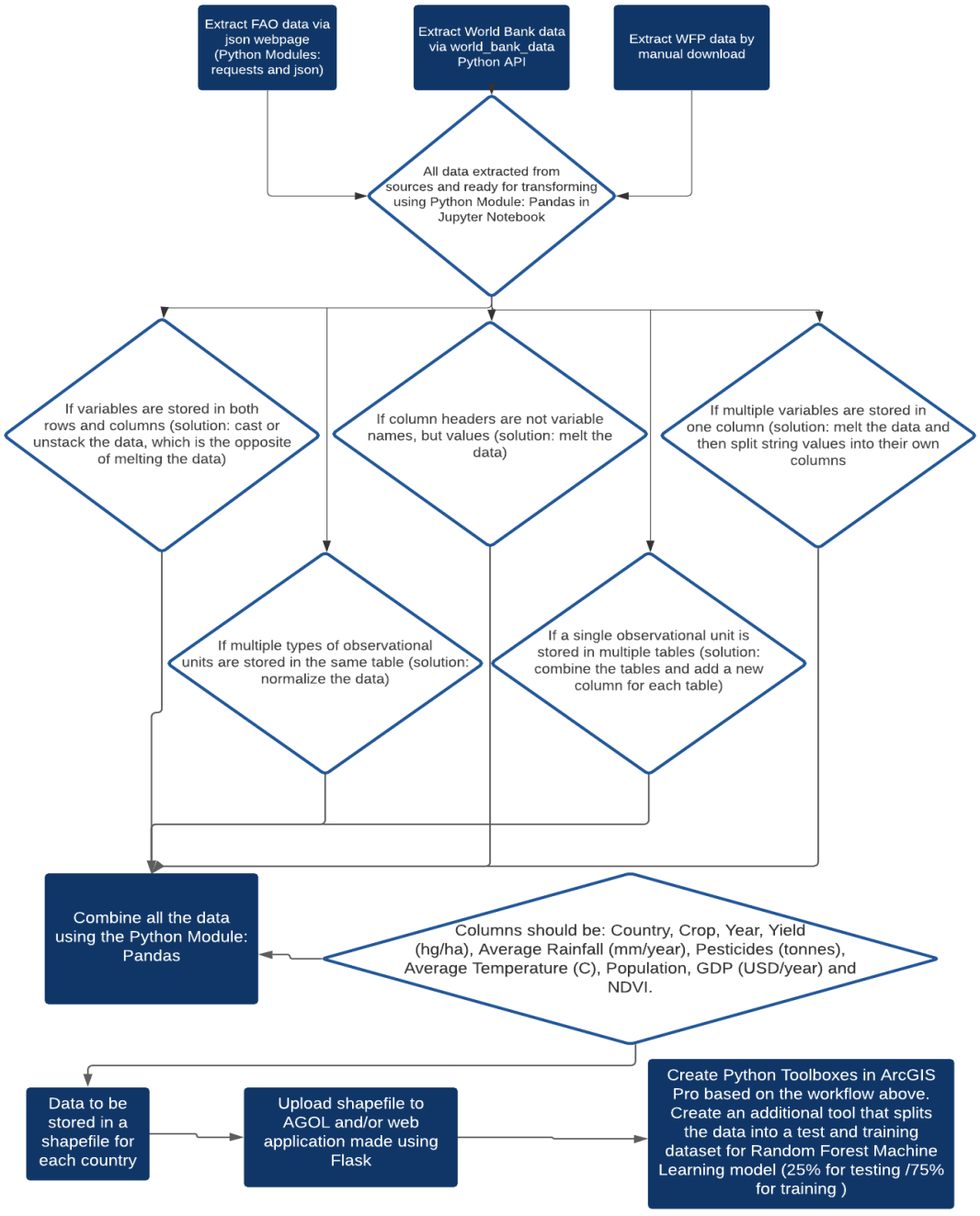

Figure #.

4.3 Python Toolbox: Download Data

Python toolboxes were created for users at the WFP to easily obtain the data. The first step in the workflow is obtaining the data. Three parameters were created in the python toolbox

for the user to choose which datasets they wanted first. The first parameter prompts the user to choose, either the FAO or World Bank. Next the specific datasets from either the FAO or World Bank are listed in the second parameter. Once the dataset is chosen, a folder path that will hold the output is required in the final parameter. The two images in Figure #. show the python toolbox display as well as the datasets that are generated after choosing the first parameter.

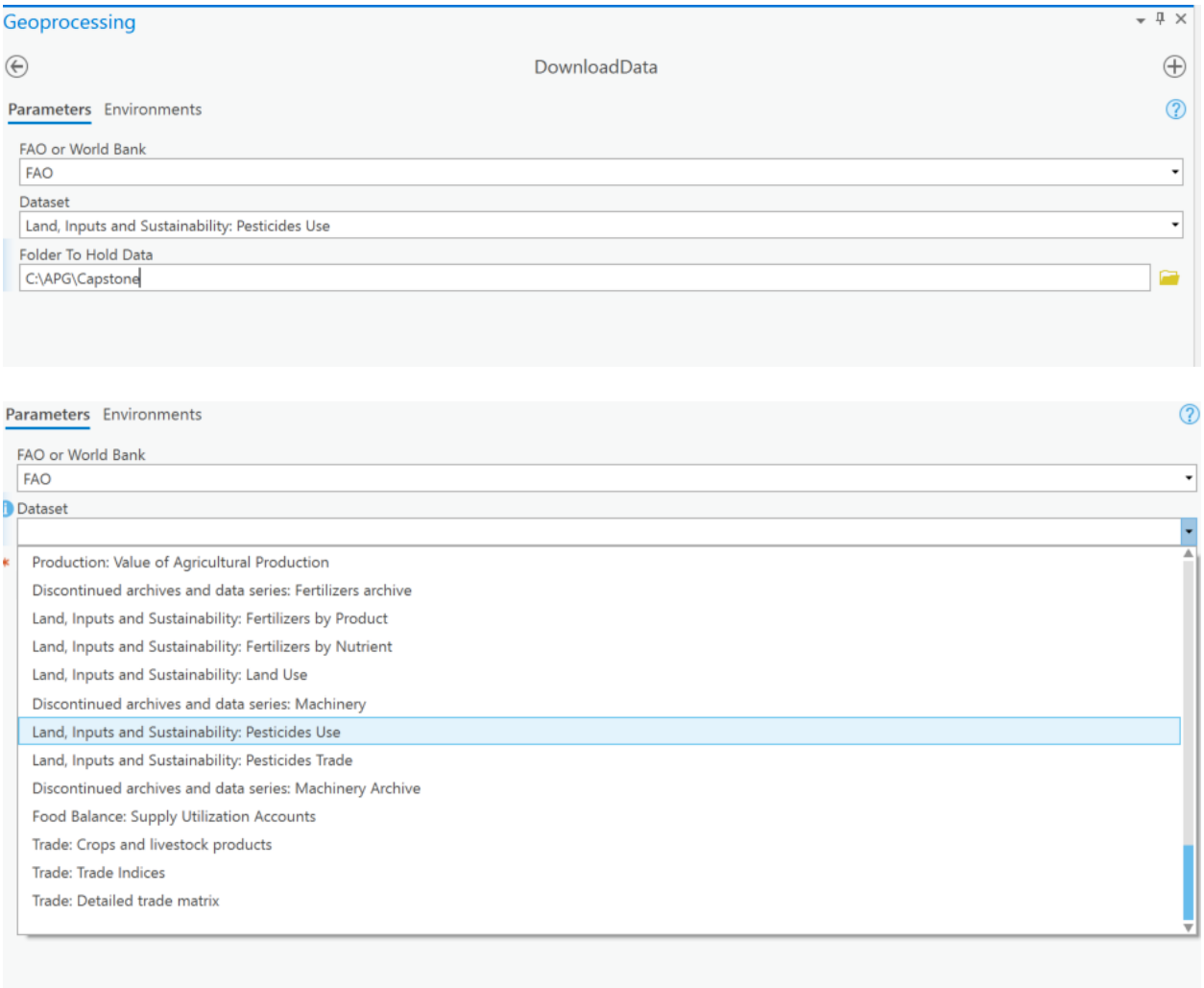

Figure #.

After filling in the parameters and running the tool, the dataset is downloaded as a zip file using the requests module, it is unzipped, and put into a Pandas data frame. Once in the Pandas data frame, country names are matched up so that once merged with the default GeoPandas

shapefile, all the data will be combined correctly. The arcpy mapping function is put into use with the python toolbox to locate the current map, return the first map, create a layer, and finally add the chosen dataset to the current map. Figure #. shows the bulk of the python toolbox.

```
def execute(self, params, messages):
    """The source code of the tool."""
    dataset = params[1].valueAsText
    file = params[2].valueAsText

    #retrieve shapefile from geopandas
    world = gpd.read_file(gpd.read_file(gpd.datasets.get_path('naturalearth_lowres')))

    #fao_json file exploration
    fao_json = "http://fenixservices.fao.org/faostat/static/bulkdownloads/datasets_E.json"
    response = requests.get(fao_json)
    data = response.json()
    dist = len(data['Datasets']['Dataset'])
    for x in range(dist):
        if data['Datasets']['Dataset'][x]['DatasetName'] == dataset:
            num = x
    fileLoc = data['Datasets']['Dataset'][num]['FileLocation']

    #request the zip
    data_r = requests.get(fileLoc)

    #combine zip with file
    local_zip = file + '/datazip.zip'
    if os.path.exists(local_zip) == True:
        os.remove(local_zip)

    #open the zip and unzip the file
    with open(local_zip, 'wb') as fd:
        for chunk in data_r.iter_content(chunk_size=128):
            fd.write(chunk)
    with ZipFile(local_zip, 'r') as zipObj:
        zipObj.extractall()
```

```
#read unzipped data
df_data = pd.read_csv(local_zip)
#make sure all the names match in both the shp and the master dataframe
df_data['Area'] = df_data['Area'].replace(['Bosnia and Herzegovina','Brunei Darussalam'
world = world.merge(df_data ,left_on='name' ,right_on='Area',how='inner')

#create a new shapefile path and create shapefile from merged data
newshp = file + '/something.shp'
world.to_file(newshp)

#locate current map
aprx = arcpy.mp.ArcGISProject("Current")
#return the first map
map = aprx.listMaps()[0]
#create a layer
lyr = arcpy.MakeFeatureLayer_management(newshp, "layer")
#add data to current map
map.addDataFromPath(lyr)

return
```

Figure #.

4.4 Python Toolbox: Combine and Test Data

There are only two parameters needed for this tool and they are all the feature classes created from the first tool and the folder path to where the output combined data should be placed/held. The first few steps are using the arcpy Merge function to combine all the data, this is a very simple process because all the data are in separate feature classes, but they all use the same shapefile. The merge function takes all the feature classes and creates a new combined output. That merged output is then placed into a NumPy array using the arcpy function FeatureClassToNumPyArray. This array is then placed into a Pandas data frame to remove all null values. This process is shown in Figure #. below.

```
def execute(self, params, messages):
    """The source code of the tool."""
    fc = params[0].valueAsText

    #merge all the fc
    master = arcpy.management.Merge(fc, "df_master")

    #create a numpy array
    array_master = arcpy.da.FeatureClassToNumPyArray(in_table=master)

    #create a df based on array
    df_master = pd.DataFrame(array_master).dropna()
```

Figure #.

The combined dataset is used to perform preliminary tests before the random forest machine learning model. Two indicators are returned from the tool, an R^2 score and a list of all the variables in the model in order of importance to the output. Data needs to be characterized by four elements: results, crop yields, indicators, and “all other” datasets using the loc Pandas function. These data are then assigned to a training class (70%) or a testing class (30%). The model is run, and the variable results are returned for the R^2 score and the ordered list of variables based on importance. The code to complete this is shown below in Figure #.

```
#split the dataframe into two sections: the results and the indicators
col_ind = list(df_rf.columns)
col_ind.remove('yield(tonnes)')
X = df_rf[col_ind]
y = df_rf['yield(tonnes)']
```

```
#split the data into testing and training data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
#run the model
rf = RandomForestRegressor(n_estimators=200, random_state=42, n_jobs=-1, verbose=1)
```

```
#fit the model based on training data
rf.fit(X_train, y_train)
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:    6.5s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:   28.4s
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:   29.4s finished
```

```
RandomForestRegressor(n_estimators=200, n_jobs=-1, random_state=42, verbose=1)
```

```
#actually predict outputs using the test data
y_pred = rf.predict(X_test)
```

```
[Parallel(n_jobs=4)]: Using backend ThreadingBackend with 4 concurrent workers.
[Parallel(n_jobs=4)]: Done  42 tasks      | elapsed:    0.1s
[Parallel(n_jobs=4)]: Done 192 tasks      | elapsed:    0.4s
[Parallel(n_jobs=4)]: Done 200 out of 200 | elapsed:    0.4s finished
```

```
#return r^2 score
r2_score(y_pred=y_pred,y_true=y_test)
```

```
0.9912822432277254
```

```
#obtain data for plot
plot_list = df_master.columns
features = X.columns
importances = rf.feature_importances_
```

```
#sort features by importance and only return top 15
feat_imp = pd.Series(importances, features).sort_values(ascending=False)
feat_imp = feat_imp[:14]
```

```
plt.style.use('fivethirtyeight')
ax = feat_imp.plot(kind='bar', title='Feature Importance')
```

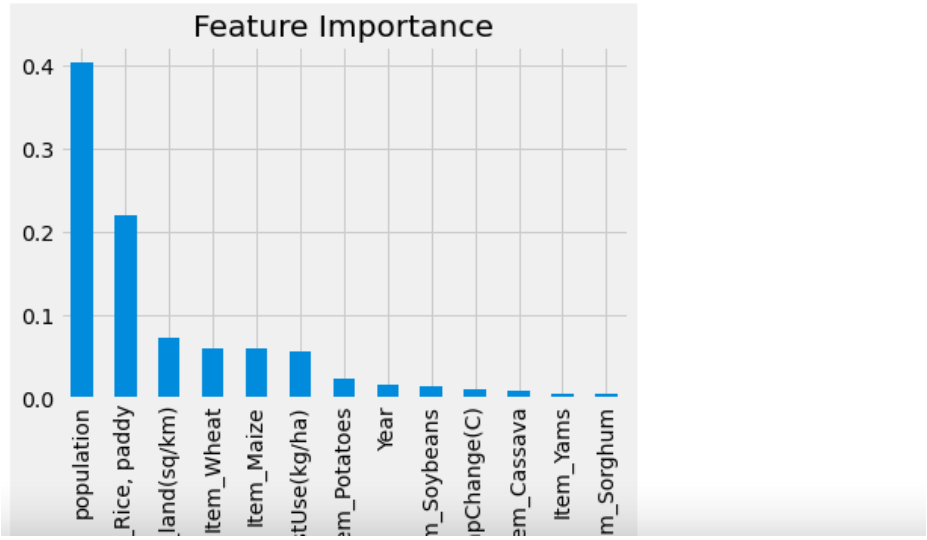

Figure #.

4.5 Literature Review on Concepts Used:

Tidy Data discusses an important part of data cleaning, which is data tidying or the structuring of datasets to facilitate analysis (Wickham, 2014). Wickham (2014) discusses and lays out a framework for developing datasets that are easy to manipulate, model, and visualize. Within tidy data, each variable forms a column, each observation forms a row, and each type of observational unit forms a table. Ordering variables and observations is key for tidy data to make your data even more easy to understand and analyze according to Wickham (2014).

Wickham assess the five most common issues with unorganized datasets and the broad solution listed below:

1. Column headers are values, not variable names (solution: melt the data)
2. Multiple variables are stored in one column (solution: melt the data and then split string values into their own columns)
3. Variables are stored in both rows and columns (solution: cast or unstack the data, which is the opposite of melting the data)
4. Multiple types of observational units are stored in the same table (solution: normalize the data)
5. A single observational unit is stored in multiple tables (solution: combine the tables and add a new column for each table)

Wickham (2014) provides guidance on the tools that can be used on both tidy and messy data. It describes three areas that these tools can be grouped into one being data manipulation, which includes tools like filtering, transforming, aggregating, and sorting, visualization, and modeling.

Once the data has been cleaned following the principles above the data should be split into a training and testing set to be implemented into the Random forest machine learning model

(Jeong, 2016). The article suggested a split around 70% training and 30% testing or 60% training and 40% testing, saying “including more data for model training is likely to improve the predictability of random forest regression” (Jeong, 2016, p. 11).

5. Results/Conclusion

The ETL workflow showed the ability to predict yields in a quick and efficient manner. The python toolboxes that replicate this ETL will be used to update all the data needed to predict crop yields when needed. This project can form the framework to provide additional workflows to make sure governments, farmers, policymakers, and many more have the information needed to prevent food insecurity in a timely manner. The key observation from this study is that one of the most important indicators for predicting crop yields is pesticide and fertilizer use. This should be carefully interpreted as our climate continues to suffer because yes more fertilizer and pesticide use may result in higher crop yields but in the long run it can devastate the land and the other natural factors that allow crops to grow.

There are many enhancements that can be made to this ETL process. One limitation discovered is the processing power requirements for modeling imagery. A larger processing computer might have allowed extracted satellite imagery from either USGS or Google Earth to be analyzed. Running a normalized difference vegetation index (NDVI) on the imagery would provide an average value for each year in each country. This could provide crop yield updates bi-weekly depending on the satellite.

6. Bibliography:

Jiang D, Yang X, Clinton N, Wang N. 2004. An artificial neural network model for estimating crop yields using remotely sensed information. 25: 1723–1732.

Jeong, J. H., Resop, J. P., Mueller, N. D., Fleisher, D. H., Yun, K., Butler, E. E., Timlin, D. J., Shim, K., Gerber, J. S., Reddy, V. R., & Kim, S. (2016). Random forests for global and regional crop yield predictions. *PLOS ONE*, *11*(6), e0156571. https://doi.org/10.1371/journal.pone.0156571

Wickham, Hadley. "Tidy Data." *Journal of Statistical Software*, vol. 59, no. 10, 2014, https://doi.org/10.18637/jss.v059.i10.