FORECASTING TRENDS IN CROP UTILIZATION

CROPS AS BIOFUELS

OVERVIEW

Do the benefits of using crops as biofuels outweigh the costs?

SPECIFIC DATA POINTS

When a crop becomes a source for biofuels, how does that impact the food supply and other commodity vectors (exports, imports, etc)?

What are the changes in inputs such as land, fertilizer and pesticides when crops are grown specifically for biofuels?

What is the effect on a crop's price when a higher percentage of its yield is used for biofuels instead of food?

FOOD & AGRICULTURE ORGANIZATION OF THE UNITED NATIONS

The FAO is a subdivision of the United Nations. They provide data sets that show how countries utilize and process their production of food items. Data is available for the years 1961 to 2017.

The available datasets breakdown the distribution of a particular commodity into subsets such as food, feed, exports, imports and other factors. The subset I am using to designate how much of a commodity was used for biofuels is 'Other uses (non-food)'. For this project, I am only collecting data for 'Maize'. This is to get an idea of how the production of corn as ethanol effects corn as a food item.

Using these data sets, along with exchange rates and pricing data, we should hopefully get a fairly accurate prediction on how changes in the utilization of a crop as a biofuel effect its utilization as food.

1	Domain Code	Domain	Area Code	Area	Element Code	Element	Item Code	Item	Year	Unit	Value	Flag Description
133272	QC	Crops	231	United States of America	5312	Area harvested	56	Maize	2001	ha	27829720	Official data
133273	FBSH	Food Balances (old methodology and population)	231	United States of America	5911	Export Quantity	2514	Maize and products	2001	1000 tonnes	48477	Standardized data
133274	FBSH	Food Balances (old methodology and population)	231	United States of America	5521	Feed	2514	Maize and products	2001	1000 tonnes	148558	Standardized data
133275	FBSH	Food Balances (old methodology and population)	231	United States of America	5142	Food	2514	Maize and products	2001	1000 tonnes	3858	Standardized data
133276	FBSH	Food Balances (old methodology and population)	231	United States of America	5611	Import Quantity	2514	Maize and products	2001	1000 tonnes	335	Standardized data
133277	FBSH	Food Balances (old methodology and population)	231	United States of America	5123	Losses	2514	Maize and products	2001			
133278	FBSH	Food Balances (old methodology and population)	231	United States of America	5154	Other uses (non-food)	2514	Maize and products	2001	1000 tonnes	25024	Standardized data
133279	FBSH	Food Balances (old methodology and population)	231	United States of America	5131	Processing	2514	Maize and products	2001	1000 tonnes	22572	Standardized data
133280	PP	Producer Prices - Annual	231	United States of America	5530	Producer Price (LCU/tonne)	56	Maize	2001	LCU	78	Official data
133281	FBSH	Food Balances (old methodology and population)	231	United States of America	5511	Production	2514	Maize and products	2001	1000 tonnes	241375	Standardized data
133282	FBSH	Food Balances (old methodology and population)	231	United States of America	5527	Seed	2514	Maize and products	2001	1000 tonnes	509	Standardized data
133283	FBSH	Food Balances (old methodology and population)	231	United States of America	5072	Stock Variation	2514	Maize and products	2001	1000 tonnes	7289	Standardized data
133284	QC	Crops	231	United States of America	5419	Yield	56	Maize	2001	hg/ha	86733	Calculated data

DATA CLEAN UP AND PREPROCESSING

Step 1: TRANSPOSE ROW DATA INTO COLUMN DATA

As an example, in the product breakdown data provided by the UN, all the row values in the Element column needed to be set as individual columns, as these will eventually be the model features. So going from this…

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	Domain Code	Domain	Area Code	Area	Element Code	Element	Item Code	Item	Year	Unit	Value	Flag Description	
2	QC	Crops	2	Afghanistan	5312	Area harvested	56	Maize	1961	ha	500000	Official data	
3	FBSH	Food Balances (old m	2	Afghanistan	5911	Export Quantity	2514	Maize and products	1961	1000 tonnes	0	Standardized data	
4	FBSH	Food Balances (old m	2	Afghanistan	5521	Feed	2514	Maize and products	1961	1000 tonnes	210	Standardized data	
5	FBSH	Food Balances (old m	2	Afghanistan	5142	Food	2514	Maize and products	1961	1000 tonnes	403	Standardized data	
6	FBSH	Food Balances (old m	2	Afghanistan	5611	Import Quantity	2514	Maize and products	1961	1000 tonnes	0	Standardized data	
7	FBSH	Food Balances (old m	2	Afghanistan	5123	Losses	2514	Maize and products	1961	1000 tonnes	70	Standardized data	
8	FBSH	Food Balances (old m	2	Afghanistan	5154	Other uses (non-food)	2514	Maize and products	1961				
9	FBSH	Food Balances (old m	2	Afghanistan	5131	Processing	2514	Maize and products	1961				
10	FBSH	Food Balances (old m	2	Afghanistan	5511	Production	2514	Maize and products	1961	1000 tonnes	700	Standardized data	
11	FBSH	Food Balances (old m	2	Afghanistan	5527	Seed	2514	Maize and products	1961	1000 tonnes	18	Standardized data	
12	FBSH	Food Balances (old m	2	Afghanistan	5072	Stock Variation	2514	Maize and products	1961				
13	QC	Crops	2	Afghanistan	5419	Yield	56	Maize	1961	hg/ha	14000	Calculated data	
14	QC	Crops	2	Afghanistan	5312	Area harvested	56	Maize	1962	ha	500000	Official data	
15	FBSH	Food Balances (old m	2	Afghanistan	5911	Export Quantity	2514	Maize and products	1962	1000 tonnes	0	Standardized data	
16	FBSH	Food Balances (old m	2	Afghanistan	5521	Feed	2514	Maize and products	1962	1000 tonnes	210	Standardized data	
17	FBSH	Food Balances (old m	2	Afghanistan	5142	Food	2514	Maize and products	1962	1000 tonnes	403	Standardized data	

…to this

Price USD	Area harvested	Export Quantity	Feed	Food	Import Quantity	Losses	...	Stock Variation	Yield	Domestic Supply	Pesticides
131000e+03	8.973000e+03	8060.000000	8751.000000	9042.000000	9306.000000	8178.000000	...	7051.000000	8969.000000	5935.000000	4.516000e+03
717459e+08	1.039355e+06	549.853598	2670.693406	529.910086	475.736514	208.501101	...	-28.768969	30170.383432	5018.965122	2.918477e+04
097215e+09	3.789772e+06	3856.141411	13565.760482	1444.436750	1570.267834	1034.670545	...	2076.579729	32838.772116	20809.515399	1.632017e+05
00000e+00	0.000000e+00	0.000000	0.000000	0.000000	0.000000	0.000000	...	-47725.000000	343.000000	0.000000	3.000000e-02
320411e+02	5.500000e+03	0.000000	8.000000	3.000000	1.000000	2.000000	...	-6.000000	11006.000000	132.000000	9.300000e+01
588324e+03	8.300000e+04	1.000000	89.000000	58.000000	21.000000	15.000000	...	0.000000	17778.000000	570.000000	1.170585e+03
439703e+05	5.090000e+05	21.000000	680.000000	356.000000	193.000000	67.000000	...	0.000000	38034.000000	2289.000000	7.377250e+03

DATA CLEAN UP AND PREPROCESSING

Step 2: NORMALIZE COLUMN VALUES

The values for many feature columns skewed heavily, due to the differences in each countries total yields. As an example, in 2013 the United States produced 353,699 kilo tonnes of Maize, while Afghanistan only produced 312 kilo tonnes. To make these value amounts usable, I needed to covert the actual amounts to a percentage of total domestic supply for each country. So going from this…

133433	FBSH	Food Balances (old m	231	United States of Ar	5123	Losses	2514	Maize and products	2013			
133434	FBSH	Food Balances (old m	231	United States of Ar	5154	Other uses (non-food)	2514	Maize and products	2013	1000 tonnes	137023	Standardized data
133435	FBSH	Food Balances (old m	231	United States of Ar	5131	Processing	2514	Maize and products	2013	1000 tonnes	23230	Standardized data
133436	PP	Producer Prices - Annu	231	United States of Ar	5530	Producer Price (LCU/tonne)	56	Maize	2013	LCU	176	Official data
133437	FBSH	Food Balances (old m	231	United States of Ar	5511	Production	2514	Maize and products	2013	1000 tonnes	353699	Standardized data
133438	FBSH	Food Balances (old m	231	United States of Ar	5527	Seed	2514	Maize and products	2013	1000 tonnes	582	Standardized data
133439	FBSH	Food Balances (old m	231	United States of Ar	5072	Stock Variation	2514	Maize and products	2013	1000 tonnes	-39863	Standardized data

…to this

eld	Domestic Supply	Pesticides	Fertilizer	food_supply_percentage	feed_supply_percentage	export_supply_percentage	other_use_supply_percentage	import_supply_percentage
1.0	90207.0	NaN	7646500.0	1.629585	90.445309	8.486891	3.638299	0.034365
1.0	88924.0	NaN	8604260.0	1.685709	90.179254	12.183966	3.881967	0.038235

DATA CLEAN UP AND PREPROCESSING

Step 3: CREATE NEW FEATURE COLUMNS FROM EXISTING ONES

The price values given in the original dataset, were given in local currencies. To normalize the prices I was able to get historic US exchange rate data for some of the countries in the dataset. I used the exchange rate to create the 'Price USD' column. Becomes I was not able to adjust for inflation, I also categorized the prices into a column labeled 'Price Levels'. I divided the 'Price USD' into 4 bins and was able to use that feature for decreasing the RMSE.

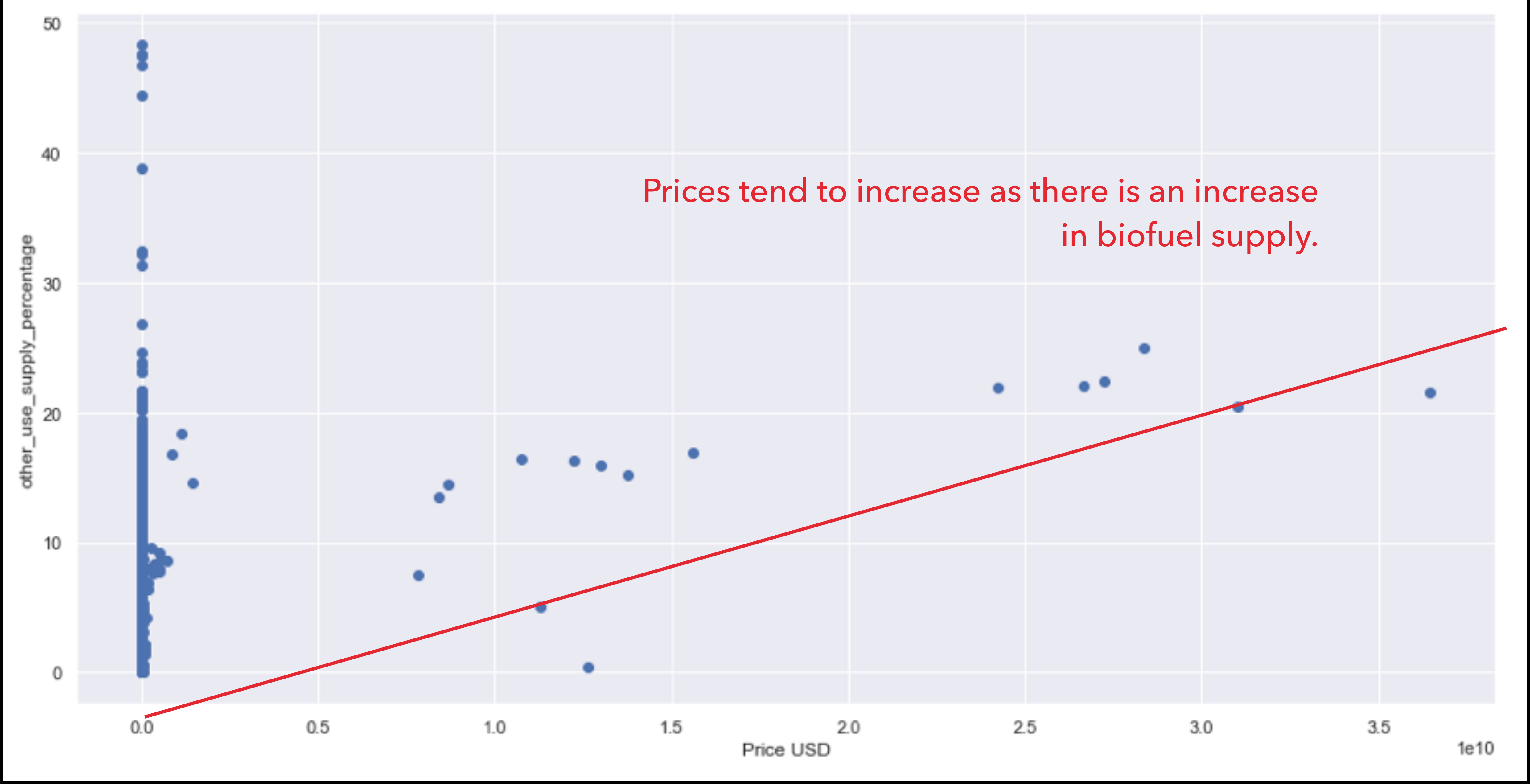
	Producer Price (LCU/tonne)	Price USD	Price Level
0	76.19	68.026775	3.0
1	2178.00	4115.317932	4.0
2	108.00	0.126360	1.0
3	57.87	62.344335	3.0
4	0.29	0.001117	1.0

Step 4: REMOVING NULL VALUES AND ADDING DEFAULT VALUES

Lastly I either dropped rows that did not have certain features or target values needed to train the model properly. I also used a SimpleImputer to fill certain features with '0' when applicable.

EXPLORATORY ANALYSIS: PRICES

After loading in the transformed and cleaned data. I used MatplotLib and Seaborn to create some visuals around the specific points mentioned in the overview. First off, I wanted to see how 'other uses' of a crop effected its price. As shown in the chart below.

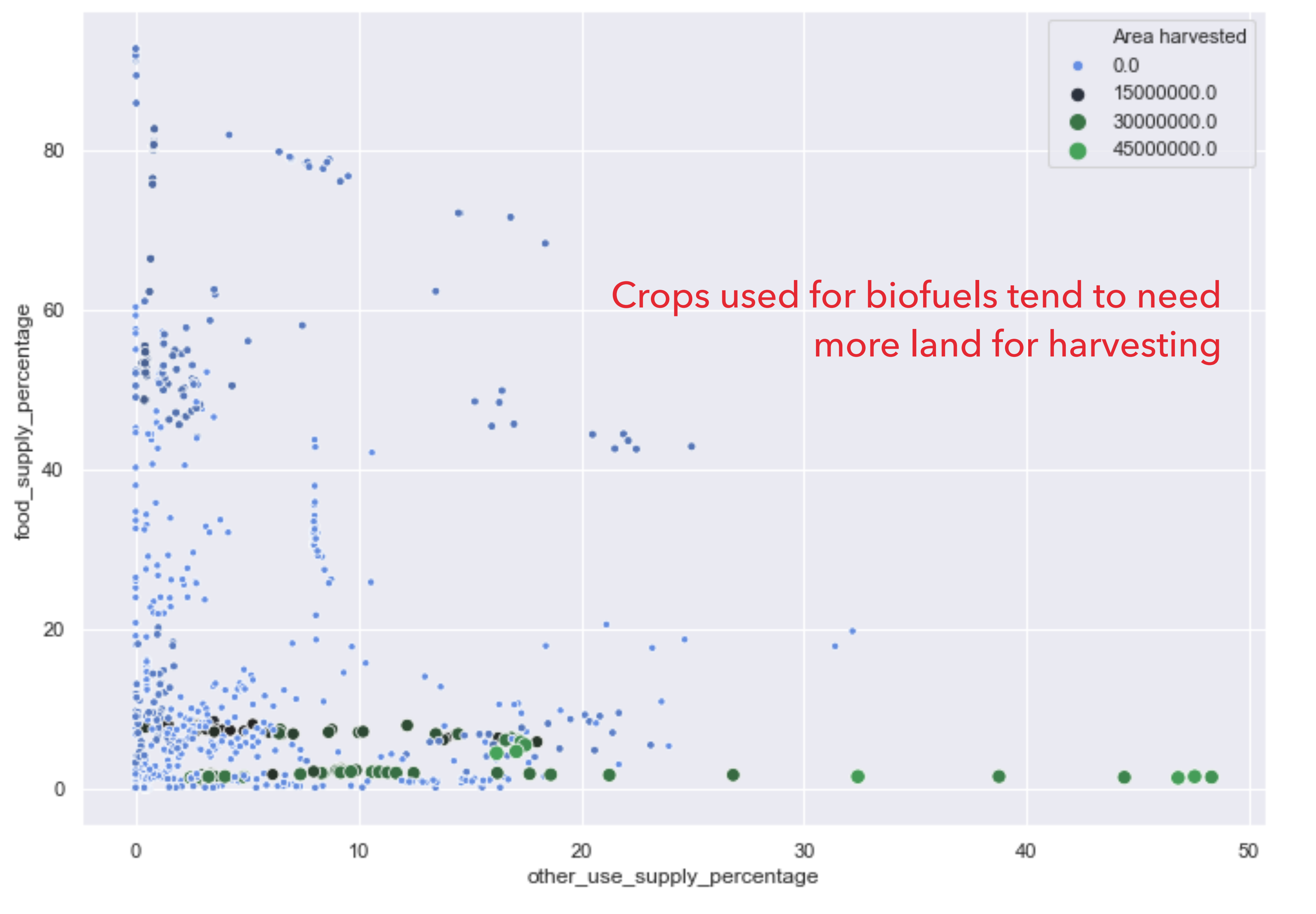


EXPLORATORY ANALYSIS: PRICES

Next I plotted how a crop's use as a food effected it's price. After reviewing this chart and the previous one, there is a definite trend with 'other uses' (i.e. biofuels) and food in relation to pricing.

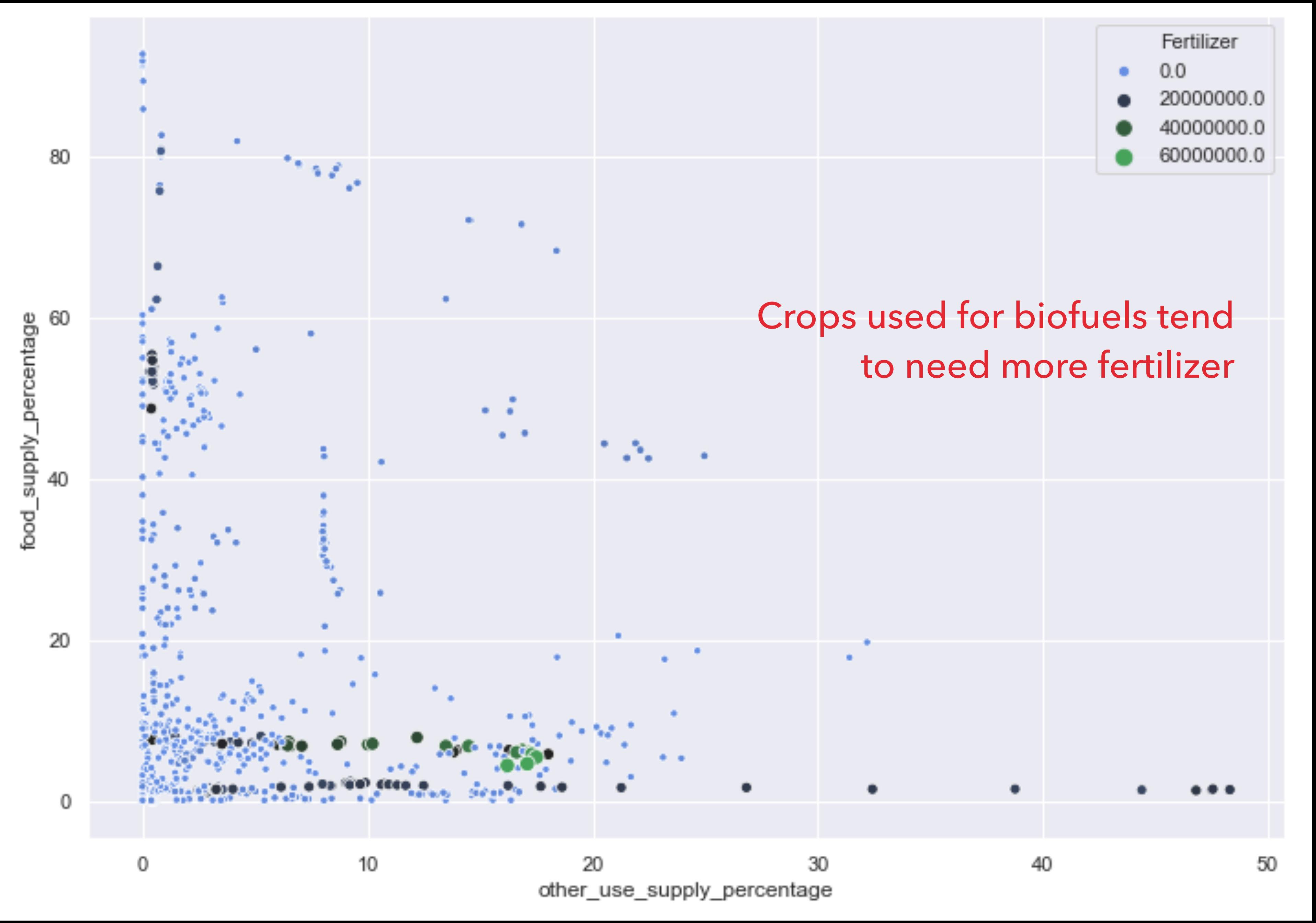
EXPLORATORY ANALYSIS: AREA HARVESTED

Next I plotted the input data as it relates to food supply and 'other uses'. First off was the 'Area harvested'. This scatter matrix shows how land use is effected when a crop is primarily farmed as a biofuel or as a food item.



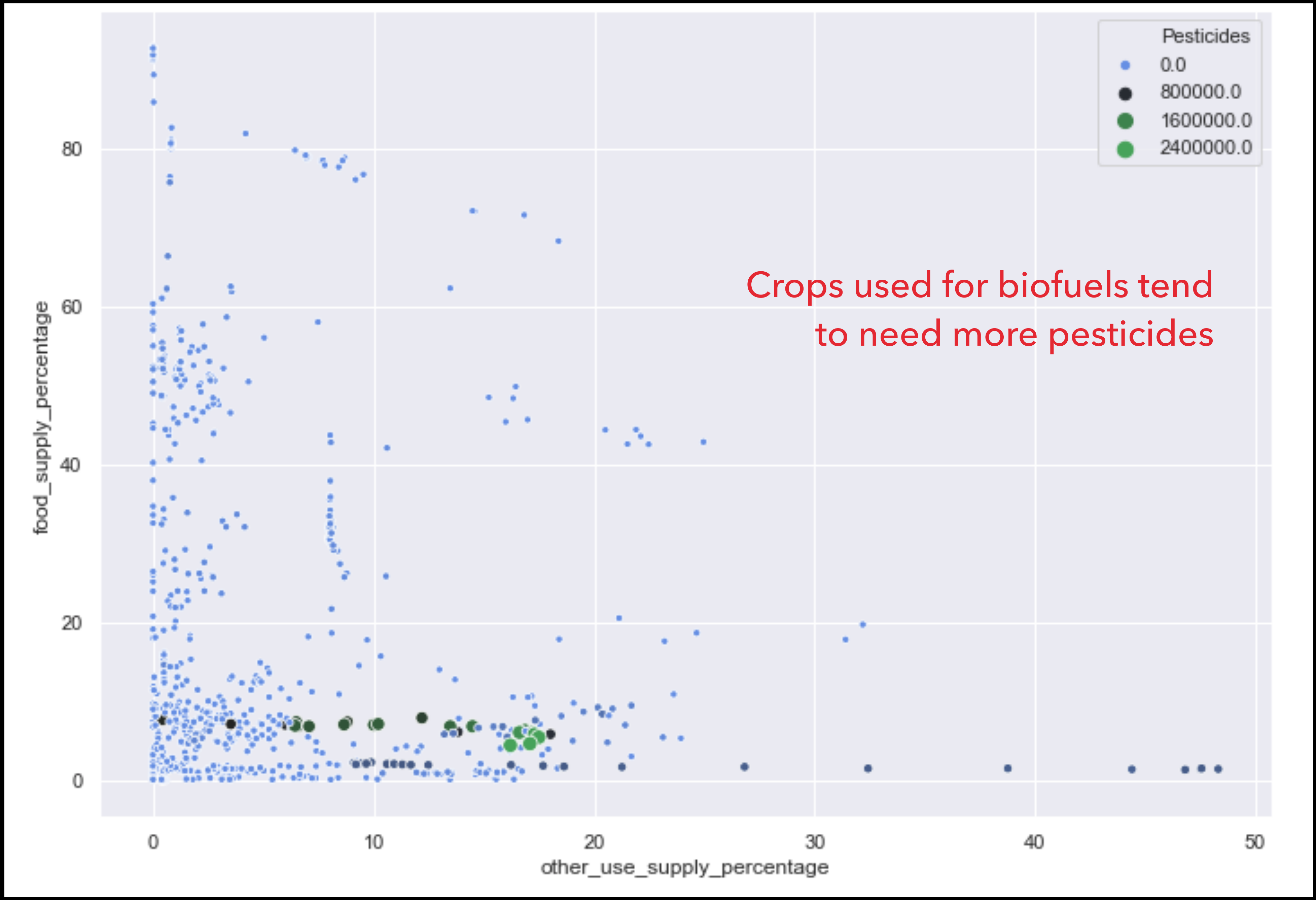
EXPLORATORY ANALYSIS: FERTILIZER

Next is fertilizer usage, this is the amount (in tonnes) used for the crop sector per year.



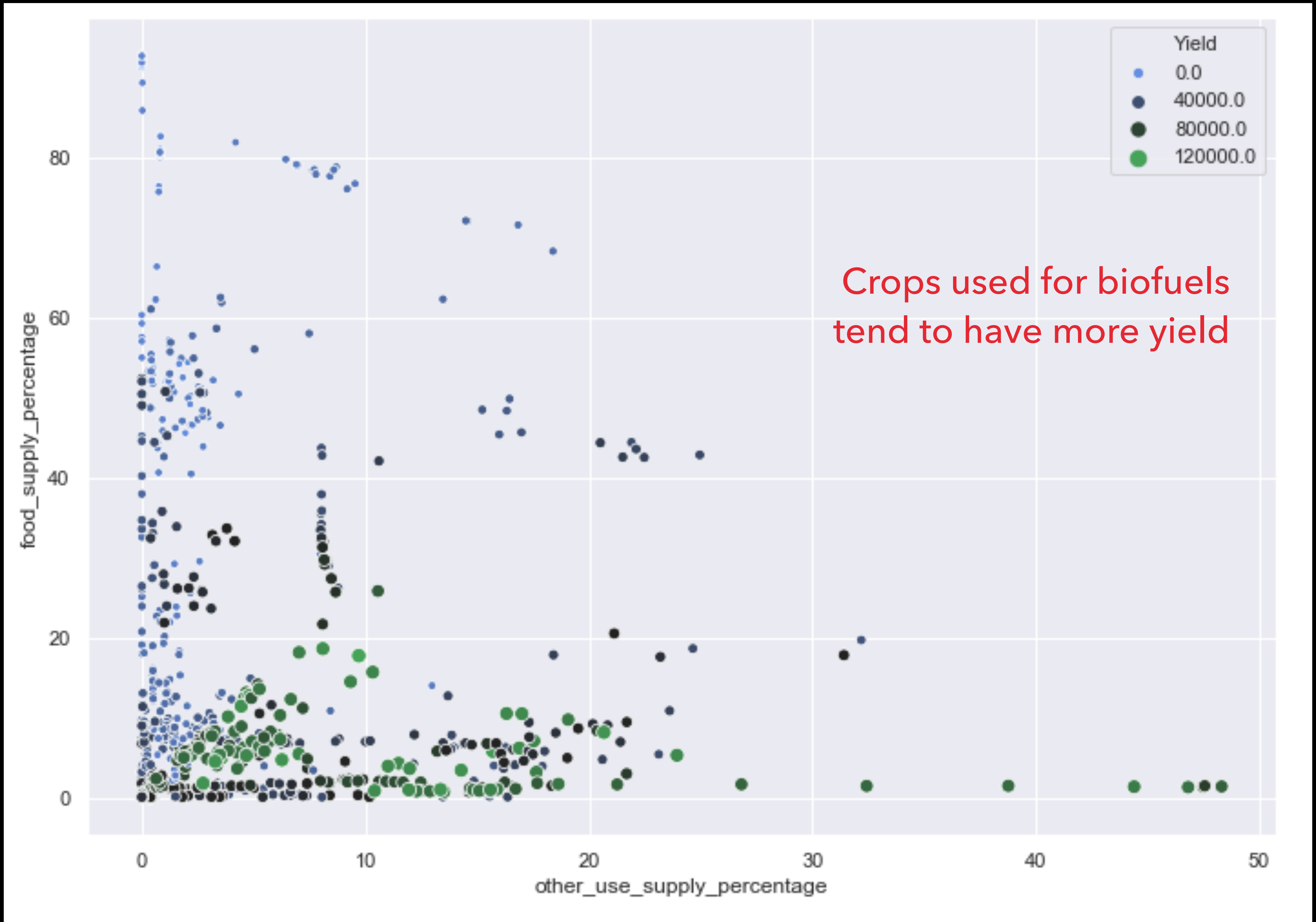
EXPLORATORY ANALYSIS: PESTICIDES

Lastly is pesticide usage, this is the amount (in tonnes) used for the crop sector per year. The pesticide data was a bit thin, as records only went back to 1990, but I was able to gain a decent trend visual from the data.



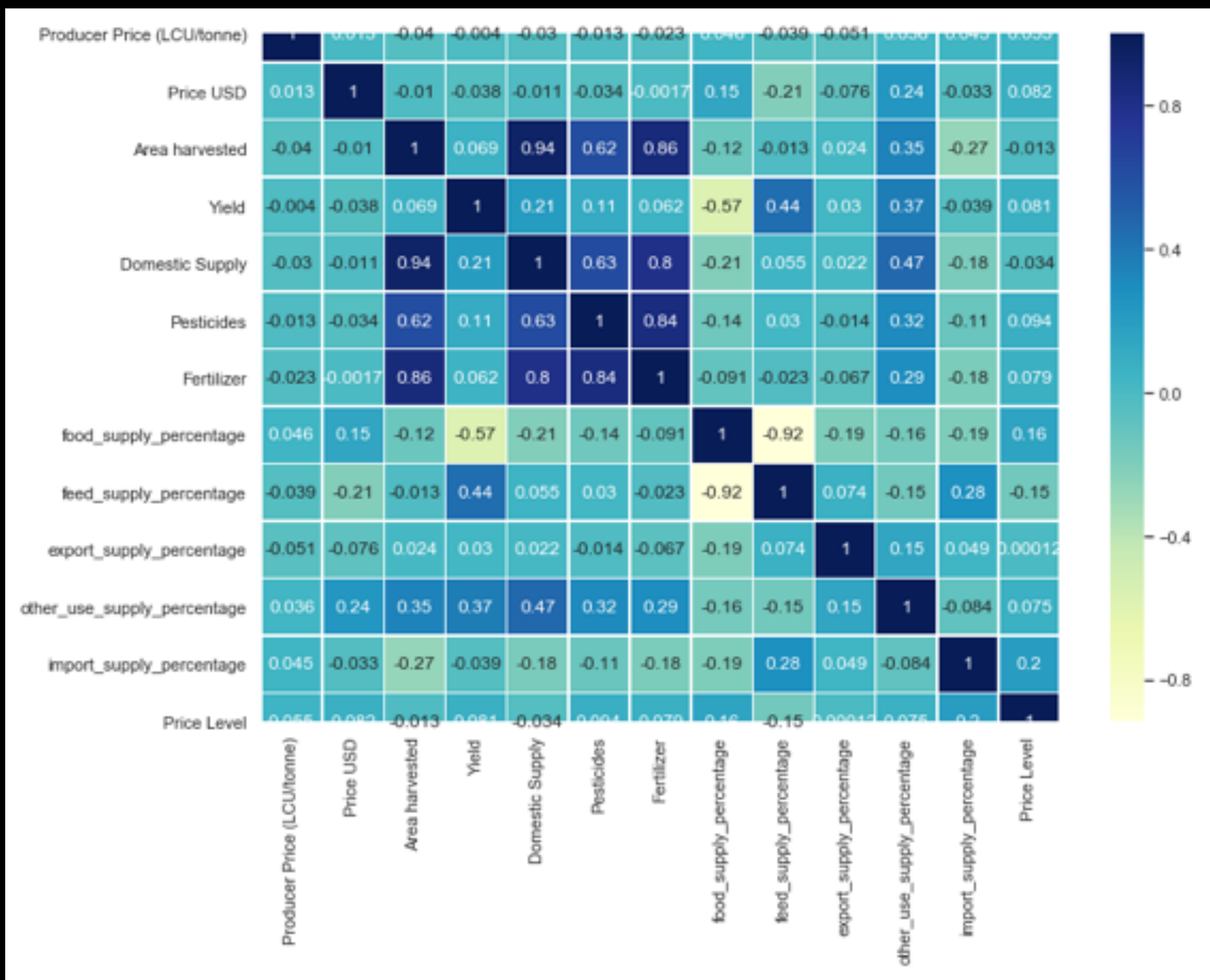
EXPLORATORY ANALYSIS: YIELD

Lastly I plotted how total yield as it relates to crop utilization as food and as biofuel. Yield data is obtained by dividing the production data by the data on area harvested. The higher the yield, the more dense the crop harvest.



EXPLORATORY ANALYSIS: CROP UTILIZATION

Using Seaborn, I created a heat map to show how the different crop utilizations effect one other. I also added the inputs to see how they effect a crop's utilization (inputs such as area harvested, fertilizer, pesticides). This helps in figuring out what features are important in training the model accurately. Feed, yield, import, price and 'other use' all play significant roles in predicting the crops food supply percentage.



EXPLORATORY ANALYSIS: CROP UTILIZATION

Looking directly at each individual corr_matrix set helped me get a better understanding of how utilization sectors effect one another. My thought was that the data here can be used as to properly transform the feature values when feeding a model predictive values. As an example, if I take the second set of percentages in the corr_matrix['other_use_supply_percentage'], my final predictive inputs could be something like this:

other_use_supply_change = 1
feed_supply_change = -0.150412
export_supply_change = 0.152328
import_supply_change = -0.084344
food_supply_change = -0.158367
food_supply_model.predict([[existing_other_use_supply_percentage + other_use_supply_change,
existing_feed_supply_percentage+ (other_use_supply_change*feed_supply_change),
existing_export_supply_percentage + (other_use_supply_change*export_supply_change),
existing_import_supply_percentage + (other_use_supply_change* import_supply_change)]])

```
corr_matrix['food_supply_percentage'].sort_values(ascending=False)

food_supply_percentage          1.000000
Price Level                     0.157386
Price USD                       0.152746
Producer Price (LCU/tonne)      0.045850
Fertilizer                     -0.090605
Area harvested                 -0.115741
Pesticides                     -0.135172
other_use_supply_percentage    -0.158367
import_supply_percentage       -0.194038
export_supply_percentage       -0.194109
Domestic Supply                -0.209100
Yield                          -0.567638
feed_supply_percentage         -0.916291
Name: food_supply_percentage, dtype: float64
```

```
corr_matrix['other_use_supply_percentage'].sort_values(ascending=False)

other_use_supply_percentage    1.000000
Domestic Supply                0.473425
Yield                          0.373510
Area harvested                 0.348011
Pesticides                     0.317041
Fertilizer                     0.294942
Price USD                      0.243772
export_supply_percentage       0.152328
Price Level                    0.074859
Producer Price (LCU/tonne)     0.036049
import_supply_percentage      -0.084344
feed_supply_percentage        -0.150412
food_supply_percentage        -0.158367
Name: other_use_supply_percentage, dtype: float64
```

```
corr_matrix['export_supply_percentage'].sort_values(ascending=False)

export_supply_percentage        1.000000
other_use_supply_percentage     0.152328
feed_supply_percentage          0.073681
import_supply_percentage        0.049263
Yield                           0.030399
Area harvested                  0.023651
Domestic Supply                 0.022103
Price Level                     0.000123
Pesticides                     -0.014132
Producer Price (LCU/tonne)     -0.050784
Fertilizer                     -0.066859
Price USD                      -0.075637
food_supply_percentage         -0.194109
Name: export_supply_percentage, dtype: float64
```

```
corr_matrix['import_supply_percentage'].sort_values(ascending=False)

import_supply_percentage        1.000000
feed_supply_percentage          0.280651
Price Level                     0.202651
export_supply_percentage        0.049263
Producer Price (LCU/tonne)      0.045285
Price USD                      -0.032847
Yield                          -0.039409
other_use_supply_percentage    -0.084344
Pesticides                     -0.109344
Domestic Supply                -0.176287
Fertilizer                     -0.184323
food_supply_percentage         -0.194038
Area harvested                 -0.270409
Name: import_supply_percentage, dtype: float64
```

```
corr_matrix['feed_supply_percentage'].sort_values(ascending=False)

feed_supply_percentage          1.000000
Yield                           0.438859
import_supply_percentage        0.280651
export_supply_percentage        0.073681
Domestic Supply                 0.054624
Pesticides                      0.029704
Area harvested                 -0.013438
Fertilizer                     -0.022715
Producer Price (LCU/tonne)     -0.038709
Price Level                    -0.146673
other_use_supply_percentage    -0.150412
Price USD                      -0.207063
food_supply_percentage         -0.916291
Name: feed_supply_percentage, dtype: float64
```

TRAINING: TESTING DIFFERENT MODELS

The solution is a continuous value and requires a Regression model. Using an 80/20 train_test_split, I tested three models: Linear Regression, Decision Tree Regressor, Random Forest Regressor

LINEAR REGRESSION – CROSS VALIDATION SCORES:

Root Mean Squared Error: 6.10
Mean: 5.97
STD: 0.67

DECISION TREE REGRESSOR – CROSS VALIDATION SCORES:

Root Mean Squared Error: 4.11
Mean: 5.01
STD: 2.11

RANDOM FOREST REGRESSOR – CROSS VALIDATION SCORES:

Root Mean Squared Error: 3.71
Mean: 4.35
STD: 1.40

The Random Forest Regressor had the best validation scores, so next I used GridSearchCV to fine tune the best hyper parameters for the model. Based off the GridSearch fit, the best estimator ended up being:

```
RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=None,
           max_features=3, max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=40,
           n_jobs=None, oob_score=False, random_state=0, verbose=0,
           warm_start=False)
```

After doing a final round of predictions with the test set, the final cross validation scores ended up at:
Root Mean Squared Error: 2.69
Mean: 6.67
STD: 2.64

OVERALL FINDINGS

For the final model, I entered in the 2017 US entry for corn utilization as a base for predicting the amount of corn used as food. After validating that the initial values gave me a close prediction on the actual food supply percentage, I started to adjust the numbers fed into the prediction. The model shows that currently there is such a high percentage of corn being used for biofuels in the US, that until there is a significant reduction in biofuel utilization, the percentage of corn used as food will continue to be extremely low. The model predicted that no real change to the percentage of corn used as food would occur, until there was a decrease of over 8% in biofuel utilization.

The exploratory data shows that crops used as biofuels tend to have much higher amounts of inputs (fertilizer, pesticides, and land area). These inputs increase the overall cost of producing these crops, and would have a negative impact on the net gains by farmers.

- The Random Forest Ensemble Model using GridSearch CV performed the best

- Based off the GridSearch feature importance method, the top features to predict food supply percentage are: 'feed_supply_percentage', other_use_supply_percentage', and 'export_supply_percentage'

- Although the RMSE was fairly low (2.70), the dataset is not as robust as it should be. After removing rows with null values, I was left with less than 1,000 entries. An increase in the dataset will allow for a more valid predictive algorithm.

- Finally the inputs needed to make predictions are all dependent on one another. As biofuel usage increases, so does a crop's export percentage. The increase in biofuel usage and export usage both effect the usage of corn as a food item. How these values relate to one another must also be understood in order to have a better predictive model for crop utilization.

JUPYTER NOTEBOOKS

The following pages contain the code used to collect the data, evaluate the data and potential model types, than train the final algorithm. The datasets are available if needed.

In [1]:

```
import os
from pathlib import Path
import pandas as pd
import numpy as np
import pickle
from sklearn.impute import SimpleImputer
from sklearn.pipeline import Pipeline
from sklearn.base import BaseEstimator, TransformerMixin
```

GET ALL DATA FROM UNITED NATIONS FOOD AND AGRICULTURE ORGANIZATION CSVs

In [2]:

```
#Load the crop data
ag_production_filepath = Path('data','production_breakdown.csv')
ag_df = pd.read_csv(ag_production_filepath)
# Load the country code data
cc_filepath = Path('data', 'country-codes.csv')
cc_df = pd.read_csv(cc_filepath)
# Load exchange rate data
xr_filepath = Path('data', 'exchange_rates.csv')
xr_df = pd.read_csv(xr_filepath)
#Load the fertilizer data
ft_filepath = Path('data', 'fertilizer_total.csv')
ft_df = pd.read_csv(ft_filepath)
ftn_filepath = Path('data', 'fertilizer_by_nutrients.csv')
ftn_df = pd.read_csv(ftn_filepath)
#Load the pesticide data
pt_filepath = Path('data', 'pesticides.csv')
pt_df = pd.read_csv(pt_filepath)
```

In [22]:

```
#create a new dataframe to house all the data from the csvs
df = pd.DataFrame()
```

In [3]:

```
# Get a List of All Countries
countries_list = ag_df['Area'].drop_duplicates().sort_values()
# Get a List of All Years
years_list = ag_df['Year'].drop_duplicates().sort_values()
# Get a List of All Elements in the Production Data
elements_list = ag_df['Element'].drop_duplicates().sort_values()
```

In [4]:

```
years_series = pd.Series(years_list)
country_series = pd.Series(countries_list)
elements_series = pd.Series(elements_list)
```

In [5]:

```
countries_set = []
for index, value in years_series.items():
    for c_index, c_value in country_series.items():
            countries_set.append(c_value)
```

In [6]:

```
years_set = []
for y_index, y_value in years_series.items():
    for c_index, c_value in country_series.items():
        years_set.append(y_value)
```

In [155]:

```
df['Country'] = countries_set
```

In [156]:

```
df['Year'] = years_set
```

In [ ]:

```
#set the country codes - needed for exchange rate
for index, row in cc_df.iterrows():
    df.loc[df['Country'] == row['Country'], 'Country Code'] = row['Country_Code']
```

In [ ]:

```
#set the exchange rate - needed for Price USD
for index, row in xr_df.iterrows():
    df.loc[(df["Year"] == row["TIME"]) & (df["Country Code"] == row["LOCATION"]), 'Exchange Rate'] = row['Value']
```

In [26]:

```
#set up pesticides
for year in range(1990,2018):
    for country in country_series:
        if(pt_df.loc[(pt_df['Year'] == year) & (pt_df['Area'] == country), "Value"].values.size > 0):
            df.loc[(df['Year'] == year) & (df['Country'] == country), "Pesticides"] = pt_df.loc[(pt_df['Year'] == year) & (pt_df['Area'] == country), "Value"].values[0]
```

In [19]:

```
#set up fertilizer
for year in range(df['Year'].min(),(df['Year'].max()+1)):
    for country in country_series:
        if(ft_df.loc[(ft_df['Year'] == year) & (ft_df['Country'] == country), "Value"].values.size > 0):
                df.loc[(df['Year'] == year) & (df['Country'] == country), "Fertilizer"] = ft_df.loc[(ft_df['Year'] == year) & (ft_df['Country'] == country), "Value"].value
s[0]
```

In [ ]:

```python
def element_data(element):
    for year in range(df['Year'].min(),(df['Year'].max()+1)):
        for country in country_series:
            if(ag_df.loc[(ag_df['Year'] == year) & (ag_df['Area'] == country) & (ag_df['Element'] == element), "Value"].values.size > 0):
                df.loc[(df['Year'] == year) & (df['Country'] == country), element] = ag_df.loc[(ag_df['Year'] == year) & (ag_df['Area'] == country) & (ag_df['Element'] ==
element), "Value"].values[0]
```

In [ ]:

```python
for index, value in elements_series.items():
    df[value] = 0
```

In [ ]:

```python
#set domestic supply to allow percentage columns
df["Domestic Supply"] = df["Production"] + df["Import Quantity"] - df["Export Quantity"] + df["Stock Variation"]
```

In [14]:

```python
ftn_group = ftn_df.groupby(["Area", "Year"])["Value"].sum()
for index, value in ftn_group.items():
    df.loc[(df['Year'] == index[1]) & (df['Country'] == index[0]), "Fertilizer"] = value
```

In [172]:

```python
# This gets most of the features/column info
for e_index, e_value in elements_series.items():
    element_data(e_value)
```

In [ ]:

```python
df["Price USD"] = df['Producer Price (LCU/tonne)'] * df['Exchange Rate']
```

In [7]:

```python
df['food_supply_percentage'] = df['Food']/df['Domestic Supply']*100
df['feed_supply_percentage'] = df['Feed']/df['Domestic Supply']*100
df['other_use_supply_percentage'] = df['Other uses (non-food)']/df['Domestic Supply']*100

# Exports need to be a percentage of domestic production, as you can only export what you actually grew
df['export_supply_percentage'] = df['Export Quantity']/df['Production']*100
df['import_supply_percentage'] = df['Import Quantity']/df['Domestic Supply']*100
```

In [ ]:

```python
df.loc[(df['Exchange Rate'].notnull()) & (df['Producer Price (LCU/tonne)'].notnull())]
```

In [8]:

```python
# Create Pickle Files
df.to_pickle("df.pkl")
```

Use Pickle File To Populate DataFrame

In [3]:

```python
with open("df.pkl", "rb") as file:
    df = pickle.load(file)
```

In [6]:

```python
df.tail()
```

Out[6]:

	Country	Year	Country Code	Exchange Rate	Producer Price (LCU/tonne)	Price USD	Area harvested	Export Quantity	Feed	Food	...	Stock Variation	Yield	Domestic Supply	Pesticides	Fertilizer	food_supply_percentage	feed_supply_percentage	export_supply
14089	Western Sahara	2018	ESH	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
14090	Yemen	2018	YEM	NaN	NaN	NaN	37231.0	NaN	NaN	NaN	...	NaN	11546.0	NaN	NaN	NaN	NaN	NaN	
14091	Yugoslav SFR	2018	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
14092	Zambia	2018	ZMB	NaN	1856.8	NaN	1086006.0	NaN	NaN	NaN	...	NaN	22052.0	NaN	NaN	NaN	NaN	NaN	
14093	Zimbabwe	2018	ZWE	NaN	NaN	NaN	1191425.0	NaN	NaN	NaN	...	NaN	6131.0	NaN	NaN	NaN	NaN	NaN	

5 rows × 26 columns

In [46]:

```
import os
from pathlib import Path
import pandas as pd
from pandas.plotting import scatter_matrix
import numpy as np
import pickle
import seaborn as sns
from sklearn.impute import SimpleImputer
import matplotlib.pyplot as plt
from matplotlib import pyplot
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
%matplotlib inline
```

In [47]:

```
with open("df.pkl", "rb") as file:
    df_agriculture = pickle.load(file)
```

In [48]:

```
df_agriculture.dropna(subset=["Price USD"], inplace=True)
df_agriculture.drop(['Country', 'Country Code', 'Seed','Stock Variation','Export Quantity','Exchange Rate', 'Year', 'Feed', 'Food', 'Import Quantity', 'Production','Losses', 'Other uses (non-food)', 'Processing'], axis=1, inplace=True)
```

In [49]:

```
df_agriculture.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1131 entries, 1226 to 13596
Data columns (total 12 columns):
Producer Price (LCU/tonne)     1131 non-null float64
Price USD                      1131 non-null float64
Area harvested                 1062 non-null float64
Yield                          1062 non-null float64
Domestic Supply                976 non-null float64
Pesticides                     600 non-null float64
Fertilizer                     1131 non-null float64
food_supply_percentage         928 non-null float64
feed_supply_percentage         976 non-null float64
export_supply_percentage       1062 non-null float64
other_use_supply_percentage    839 non-null float64
import_supply_percentage       976 non-null float64
dtypes: float64(12)
memory usage: 114.9 KB
```

In [50]:

```
df_agriculture.dropna(subset=["other_use_supply_percentage"], inplace=True)
df_agriculture.dropna(subset=["food_supply_percentage"], inplace=True)
df_agriculture.dropna(subset=["export_supply_percentage"], inplace=True)
df_agriculture.dropna(subset=["import_supply_percentage"], inplace=True)
df_agriculture.dropna(subset=["feed_supply_percentage"], inplace=True)
```

In [51]:

```
imputer = SimpleImputer(strategy='constant', fill_value=0)
imputer.fit(df_agriculture)
```

Out[51]:

```
SimpleImputer(add_indicator=False, copy=True, fill_value=0, missing_values=nan,
              strategy='constant', verbose=0)
```

In [52]:

```
X = imputer.transform(df_agriculture)
df_agriculture_cleaned = pd.DataFrame(X,columns=df_agriculture.columns)
```

In [53]:

```
df_agriculture_cleaned["Price Level"] = pd.cut(x=df_agriculture_cleaned['Price USD'], bins=[0, 10.5, 55, 274, np.inf], labels=[1,2,3,4])
df_agriculture_cleaned["Price Level"] = pd.to_numeric(df_agriculture_cleaned["Price Level"])
```

In [54]:

```
df_agriculture_cleaned["Price Level"].value_counts() / len(df_agriculture_cleaned)
```

Out[54]:

```
4.0    0.648546
3.0    0.208597
1.0    0.091024
2.0    0.045512
Name: Price Level, dtype: float64
```

In [55]:

```
df_agriculture_cleaned.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 791 entries, 0 to 790
Data columns (total 13 columns):
Producer Price (LCU/tonne)     791 non-null float64
Price USD                      791 non-null float64
Area harvested                 791 non-null float64
Yield                          791 non-null float64
Domestic Supply                791 non-null float64
Pesticides                     791 non-null float64
Fertilizer                     791 non-null float64
food_supply_percentage         791 non-null float64
feed_supply_percentage         791 non-null float64
export_supply_percentage       791 non-null float64
other_use_supply_percentage    791 non-null float64
import_supply_percentage       791 non-null float64
Price Level                    786 non-null float64
dtypes: float64(13)
memory usage: 80.5 KB
```

In [56]:

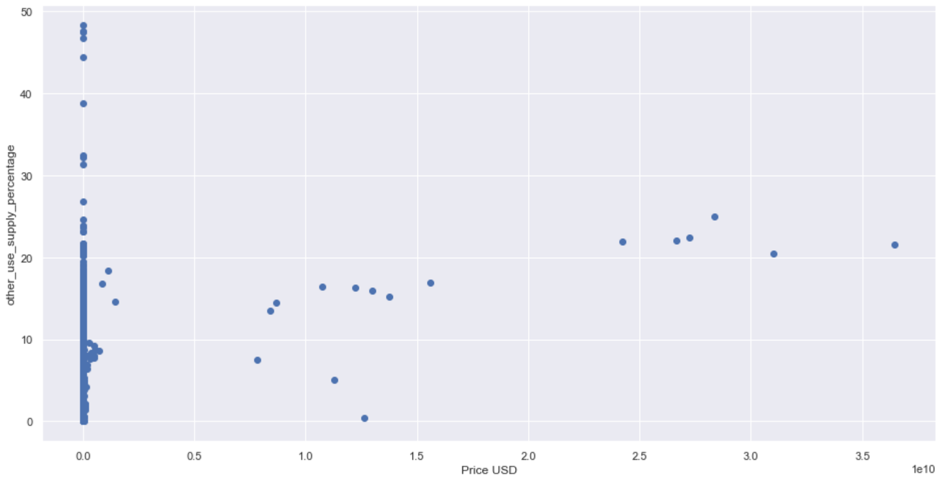
```
df_agriculture_cleaned.head()
```

Out[56]:

	Producer Price (LCU/tonne)	Price USD	Area harvested	Yield	Domestic Supply	Pesticides	Fertilizer	food_supply_percentage	feed_supply_percentage	export_supply_percentage	other_use_supply_percentage	import_supply_percentage	P L
0	76.19	68.026775	79600.0	15694.0	137.0	0.0	1128490.0	29.197080	65.693431	1.600000	1.459854	2.189781	
1	2178.00	4115.317932	55317.0	49644.0	514.0	0.0	216903.0	5.447471	89.688716	0.363636	0.194553	67.315175	
2	108.00	0.126360	574148.0	38431.0	1796.0	0.0	476842.0	7.405345	74.109131	8.473040	0.000000	4.231626	
3	57.87	62.344335	326415.0	51615.0	2267.0	0.0	812258.0	2.161447	70.313189	0.534125	0.970446	26.202029	
4	0.29	0.001117	80700.0	35357.0	315.0	0.0	111596.0	8.888889	86.666667	0.000000	2.222222	9.523810	

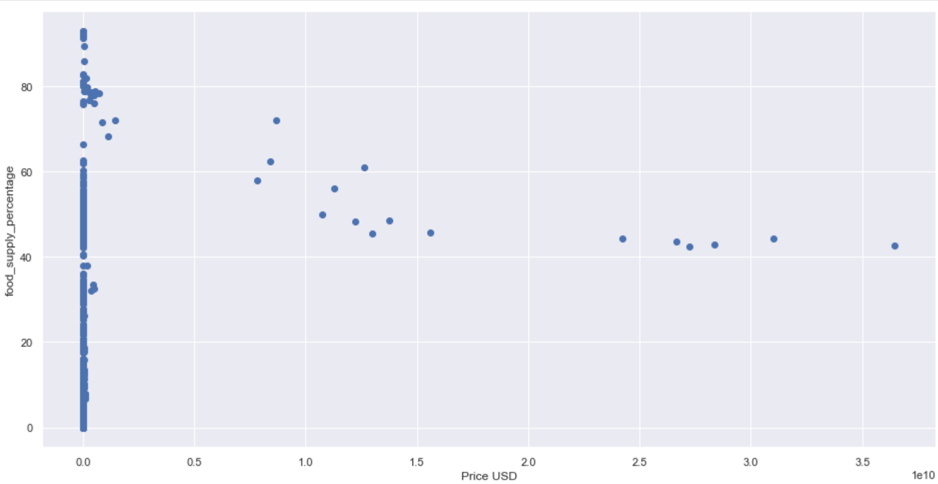
In [57]:

```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_agriculture_cleaned['Price USD'], df_agriculture_cleaned['other_use_supply_percentage'])
ax.set_xlabel('Price USD')
ax.set_ylabel('other_use_supply_percentage')
plt.show()
```



In [58]:

```
fig, ax = plt.subplots(figsize=(16,8))
ax.scatter(df_agriculture_cleaned['Price USD'], df_agriculture_cleaned['food_supply_percentage'])
ax.set_xlabel('Price USD')
ax.set_ylabel('food_supply_percentage')
plt.show()
```

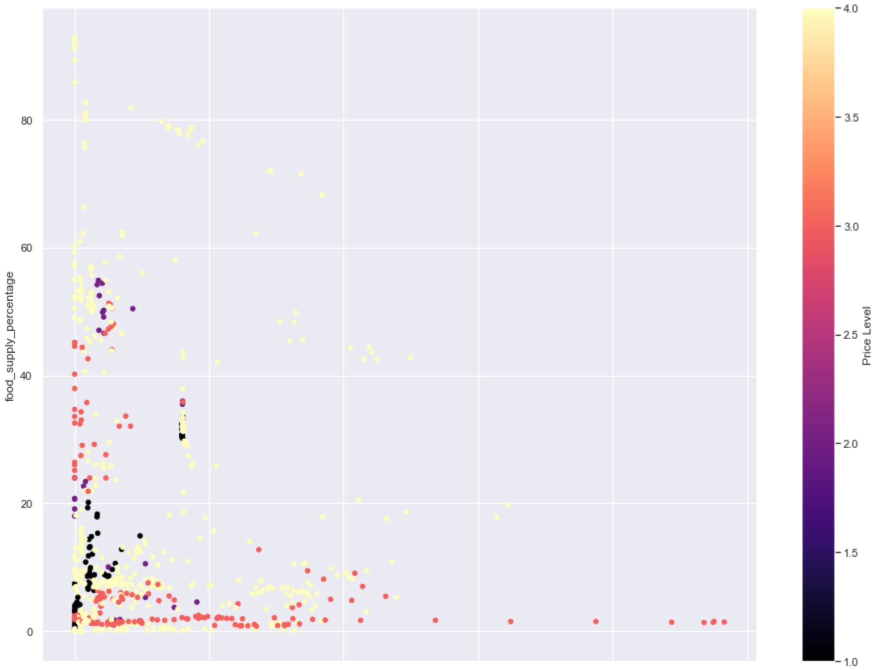


In [59]:

```
df_agriculture_cleaned.plot(kind='scatter', x='other_use_supply_percentage', y='food_supply_percentage', c='Price Level', colormap="magma", figsize=(16,12) )
```

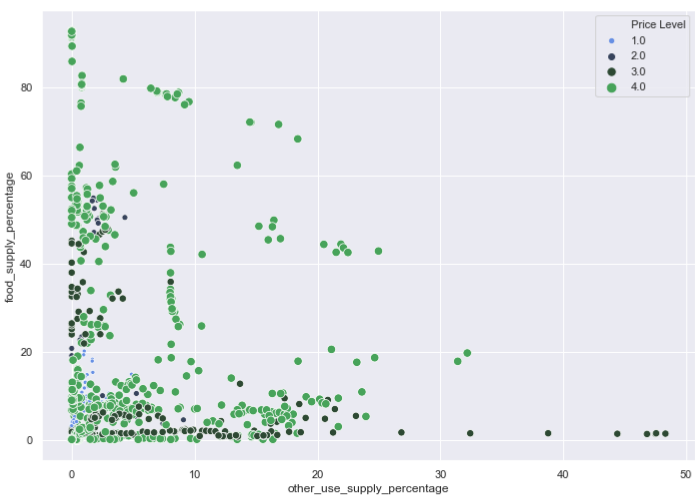
Out[59]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a25fc40b8>
```



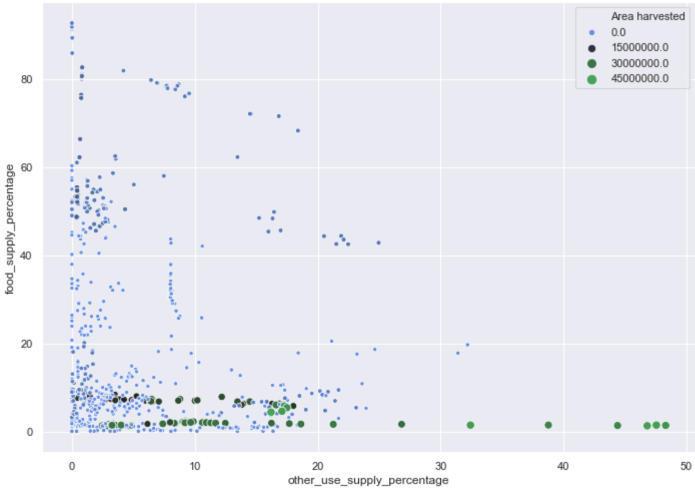
In [60]:

```
cmap = sns.diverging_palette(255, 133, l=60, n=7, center="dark", as_cmap=True)
sns.set(rc={'figure.figsize':(11.7,8.27)})
ax = sns.scatterplot(x="other_use_supply_percentage", y="food_supply_percentage", size="Price Level",
                     hue="Price Level",
                     palette=cmap, data=df_agriculture_cleaned)
```



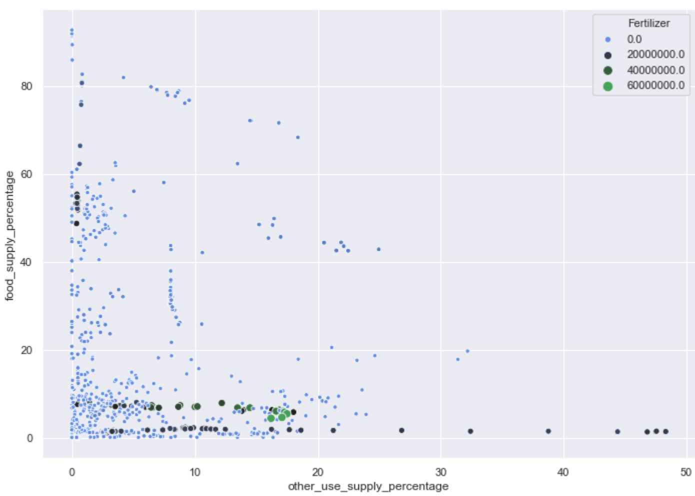
In [61]:

```
cmap = sns.diverging_palette(255, 133, l=60, n=7, center="dark", as_cmap=True)
sns.set(rc={'figure.figsize':(11.7,8.27)})
ax = sns.scatterplot(x="other_use_supply_percentage", y="food_supply_percentage", size="Area harvested",
                     hue="Area harvested",
                     palette=cmap, data=df_agriculture_cleaned)
```



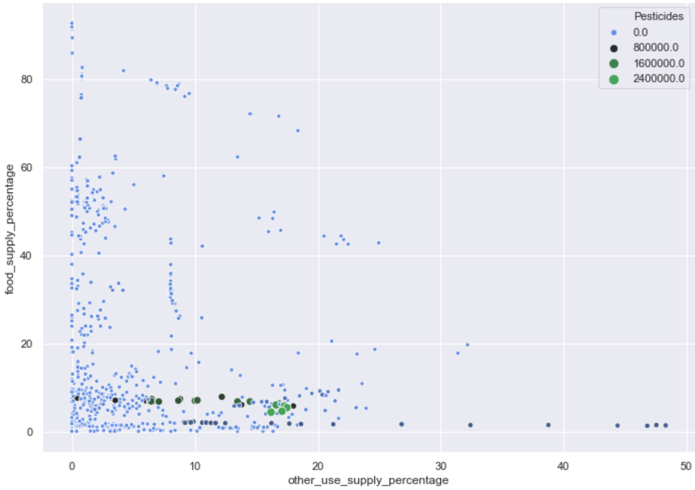
In [62]:

```
cmap = sns.diverging_palette(255, 133, l=60, n=7, center="dark", as_cmap=True)
sns.set(rc={'figure.figsize':(11.7,8.27)})
ax = sns.scatterplot(x="other_use_supply_percentage", y="food_supply_percentage", size="Fertilizer",
                     hue="Fertilizer",
                     palette=cmap, data=df_agriculture_cleaned)
```



In [63]:

```
cmap = sns.diverging_palette(255, 133, l=60, n=7, center="dark", as_cmap=True)
sns.set(rc={'figure.figsize':(11.7,8.27)})
ax = sns.scatterplot(x="other_use_supply_percentage", y="food_supply_percentage", size="Pesticides",
                     hue="Pesticides",
                     palette=cmap, data=df_agriculture_cleaned)
```



In [64]:

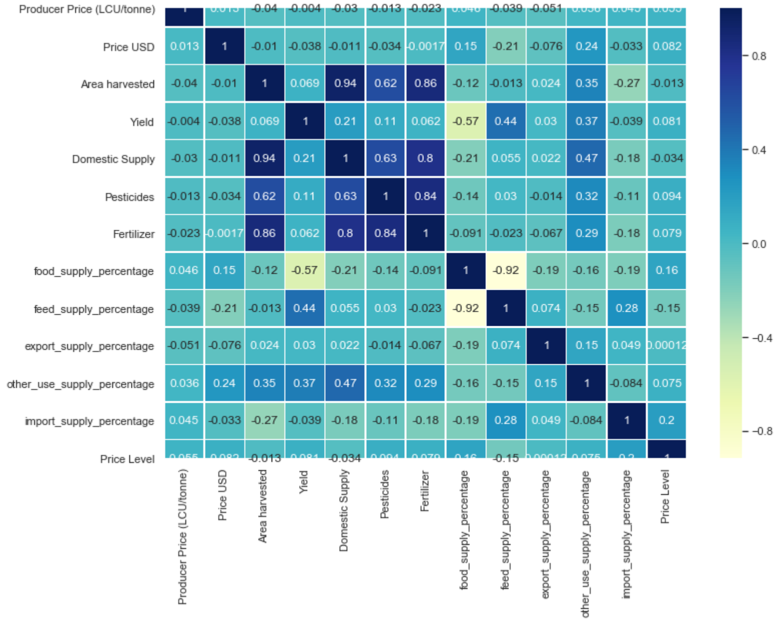
```
cmap = sns.diverging_palette(255, 133, l=60, n=7, center="dark", as_cmap=True)
sns.set(rc={'figure.figsize':(11.7,8.27)})
ax = sns.scatterplot(x="other_use_supply_percentage", y="food_supply_percentage", size="Yield",
                     hue="Yield",
                     palette=cmap, data=df_agriculture_cleaned)
```

In [65]:

```
corr_matrix = df_agriculture_cleaned.corr()
```

In [74]:

```
ax = sns.heatmap(corr_matrix, linewidths=.5, annot=True, cmap="YlGnBu")
```



In [203]:

```
corr_matrix['food_supply_percentage'].sort_values(ascending=False)
```

Out[203]:

```
food_supply_percentage         1.000000
Price Level                    0.157386
Price USD                      0.152746
Producer Price (LCU/tonne)     0.045850
Fertilizer                    -0.090605
Area harvested                -0.115741
Pesticides                    -0.135172
other_use_supply_percentage   -0.158367
import_supply_percentage      -0.194038
export_supply_percentage      -0.194109
Domestic Supply               -0.209100
Yield                         -0.567638
feed_supply_percentage        -0.916291
Name: food_supply_percentage, dtype: float64
```

In [204]:

```
corr_matrix['other_use_supply_percentage'].sort_values(ascending=False)
```

Out[204]:

```
other_use_supply_percentage    1.000000
Domestic Supply                0.473425
Yield                          0.373510
Area harvested                 0.348011
Pesticides                     0.317041
Fertilizer                     0.294942
Price USD                      0.243772
export_supply_percentage       0.152328
Price Level                    0.074859
Producer Price (LCU/tonne)     0.036049
import_supply_percentage      -0.084344
feed_supply_percentage        -0.150412
food_supply_percentage        -0.158367
Name: other_use_supply_percentage, dtype: float64
```

In [205]:

```
corr_matrix['export_supply_percentage'].sort_values(ascending=False)
```

Out[205]:

```
export_supply_percentage       1.000000
other_use_supply_percentage    0.152328
feed_supply_percentage         0.073681
import_supply_percentage       0.049263
Yield                          0.030399
Area harvested                 0.023651
Domestic Supply                0.022103
Price Level                    0.000123
Pesticides                    -0.014132
Producer Price (LCU/tonne)    -0.050784
Fertilizer                    -0.066859
Price USD                     -0.075637
food_supply_percentage        -0.194109
Name: export_supply_percentage, dtype: float64
```

In [206]:

```
corr_matrix['import_supply_percentage'].sort_values(ascending=False)
```

Out[206]:

```
import_supply_percentage       1.000000
feed_supply_percentage         0.280651
Price Level                    0.202651
export_supply_percentage       0.049263
Producer Price (LCU/tonne)     0.045285
Price USD                     -0.032847
Yield                         -0.039409
other_use_supply_percentage   -0.084344
Pesticides                    -0.109344
Domestic Supply               -0.176287
Fertilizer                    -0.184323
food_supply_percentage        -0.194038
Area harvested                -0.270409
Name: import_supply_percentage, dtype: float64
```

In [207]:

```
corr_matrix['feed_supply_percentage'].sort_values(ascending=False)
```

Out[207]:

```
feed_supply_percentage         1.000000
Yield                          0.438859
import_supply_percentage       0.280651
export_supply_percentage       0.073681
Domestic Supply                0.054624
Pesticides                     0.029704
Area harvested                -0.013438
Fertilizer                    -0.022715
Producer Price (LCU/tonne)    -0.038709
Price Level                   -0.146673
other_use_supply_percentage   -0.150412
Price USD                     -0.207063
food_supply_percentage        -0.916291
Name: feed_supply_percentage, dtype: float64
```

In [208]:

```
corr_matrix['Price Level'].sort_values(ascending=False)
```

Out[208]:

```
Price Level                    1.000000
import_supply_percentage       0.202651
food_supply_percentage         0.157386
Pesticides                     0.094179
Price USD                      0.082264
Yield                          0.081167
Fertilizer                     0.079476
other_use_supply_percentage    0.074859
Producer Price (LCU/tonne)     0.054777
export_supply_percentage       0.000123
Area harvested                -0.012630
Domestic Supply               -0.034066
feed_supply_percentage        -0.146673
Name: Price Level, dtype: float64
```

In [209]:

```
corr_matrix['Price USD'].sort_values(ascending=False)
```

Out[209]:

```
Price USD                      1.000000
other_use_supply_percentage    0.243772
food_supply_percentage         0.152746
Price Level                    0.082264
Producer Price (LCU/tonne)     0.012936
Fertilizer                    -0.001656
Area harvested                -0.010489
Domestic Supply               -0.011383
import_supply_percentage      -0.032847
Pesticides                    -0.034047
Yield                         -0.038366
export_supply_percentage      -0.075637
feed_supply_percentage        -0.207063
Name: Price USD, dtype: float64
```

In [210]:

```
corr_matrix['Area harvested'].sort_values(ascending=False)
```

Out[210]:

```
Area harvested                 1.000000
Domestic Supply                0.939048
Fertilizer                     0.860021
Pesticides                     0.618115
other_use_supply_percentage    0.348011
Yield                          0.068842
export_supply_percentage       0.023651
Price USD                     -0.010489
Price Level                   -0.012630
feed_supply_percentage        -0.013438
Producer Price (LCU/tonne)    -0.040185
food_supply_percentage        -0.115741
import_supply_percentage      -0.270409
Name: Area harvested, dtype: float64
```

In [211]:

```
corr_matrix['Fertilizer'].sort_values(ascending=False)
```

Out[211]:

```
Fertilizer                     1.000000
Area harvested                 0.860021
Pesticides                     0.843292
Domestic Supply                0.803875
other_use_supply_percentage    0.294942
Price Level                    0.079476
Yield                          0.061645
Price USD                     -0.001656
feed_supply_percentage        -0.022715
Producer Price (LCU/tonne)    -0.022790
export_supply_percentage      -0.066859
food_supply_percentage        -0.090605
import_supply_percentage      -0.184323
Name: Fertilizer, dtype: float64
```

In [212]:

```
corr_matrix['Pesticides'].sort_values(ascending=False)
```

Out[212]:

```
Pesticides                     1.000000
Fertilizer                     0.843292
Domestic Supply                0.630489
Area harvested                 0.618115
other_use_supply_percentage    0.317041
Yield                          0.110639
Price Level                    0.094179
feed_supply_percentage         0.029704
Producer Price (LCU/tonne)    -0.012831
export_supply_percentage      -0.014132
Price USD                     -0.034047
import_supply_percentage      -0.109344
food_supply_percentage        -0.135172
Name: Pesticides, dtype: float64
```

In [213]:

```
df_agriculture_cleaned.to_pickle("data/df_agriculture_cleaned.pkl")
```

In [1]:

```
import os
from pathlib import Path
import pandas as pd
from pandas.plotting import scatter_matrix
import numpy as np
import pickle
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
%matplotlib inline
```

In [2]:

```
with open("data/df_agriculture_cleaned.pkl", "rb") as file:
    df_agriculture_cleaned = pickle.load(file)
```

In [3]:

```
train_set, test_set = train_test_split(df_agriculture_cleaned, test_size=0.2, random_state=42)
```

In [4]:

```
train_set = train_set[~train_set.isin([np.nan, np.inf, -np.inf]).any(1)]
test_set = test_set[~test_set.isin([np.nan, np.inf, -np.inf]).any(1)]
```

In [5]:

```
food_supply_train_features = train_set.loc[:,['other_use_supply_percentage', 'feed_supply_percentage', 'export_supply_percentage', 'import_supply_percentage']]
food_supply_train_target = train_set.loc[:,'food_supply_percentage']

food_supply_test_features = test_set.loc[:,['other_use_supply_percentage', 'feed_supply_percentage', 'export_supply_percentage', 'import_supply_percentage']]
food_supply_test_target = test_set.loc[:,'food_supply_percentage']
```

In [6]:

```
def useLinearRegression(train_features, train_target, test_features, test_target):
    lr = LinearRegression()
    lr.fit(train_features, train_target)
    lr_predictions = lr.predict(test_features)
    lr_mse = mean_squared_error(test_target, lr_predictions)
    lr_rmse = np.sqrt(lr_mse)
    lr_cvs_scores = cross_val_score(lr, train_features, train_target, scoring="neg_mean_squared_error", cv=10)
    lr_rmse_scores = np.sqrt(-lr_cvs_scores)
    lr_rmse_mean_score = lr_rmse_scores.mean()
    lr_rmse_std_score = lr_rmse_scores.std()
    return [lr_rmse, lr_rmse_mean_score, lr_rmse_std_score]
```

In [7]:

```
lr_food_supply_scores = useLinearRegression(food_supply_train_features, food_supply_train_target, food_supply_test_features, food_supply_test_target)
```

In [8]:

```
print("Food Supply - Linear Regression - Root Mean Squared Error: {}".format(lr_food_supply_scores[0]))
print("Food Supply - Linear Regression - Mean: {}".format(lr_food_supply_scores[1]))
print("Food Supply - Linear Regression - STD: {}".format(lr_food_supply_scores[2]))
```

```
Food Supply - Linear Regression - Root Mean Squared Error: 6.108048483033716
Food Supply - Linear Regression - Mean: 5.9768286615664845
Food Supply - Linear Regression - STD: 0.6758107508172784
```

In [9]:

```
def useDecisionTreeRegressor(train_features, train_target, test_features, test_target):
    dtr = DecisionTreeRegressor()
    dtr.fit(train_features, train_target)
    dtr_predictions = dtr.predict(test_features)
    dtr_mse = mean_squared_error(test_target, dtr_predictions)
    dtr_rmse = np.sqrt(dtr_mse)
    dtr_scores = cross_val_score(dtr, train_features, train_target, scoring="neg_mean_squared_error", cv=10)
    dtr_rmse_scores = np.sqrt(-dtr_scores)
    dtr_rmse_mean_score = dtr_rmse_scores.mean()
    dtr_rmse_std_score = dtr_rmse_scores.std()
    return [dtr_rmse, dtr_rmse_mean_score, dtr_rmse_std_score]
```

In [10]:

```
dtr_food_supply_scores = useDecisionTreeRegressor(food_supply_train_features, food_supply_train_target, food_supply_test_features, food_supply_test_target)
```

In [11]:

```
print("Food Supply - Decision Tree Regression - Root Mean Squared Error: {}".format(dtr_food_supply_scores[0]))
print("Food Supply - Decision Tree Regression - Mean: {}".format(dtr_food_supply_scores[1]))
print("Food Supply - Decision Tree Regression - STD: {}".format(dtr_food_supply_scores[2]))
```

```
Food Supply - Decision Tree Regression - Root Mean Squared Error: 3.9676878035278405
Food Supply - Decision Tree Regression - Mean: 5.075420276579883
Food Supply - Decision Tree Regression - STD: 2.0469622844887394
```

In [12]:

```
def useRandomForestRegressor(train_features, train_target, test_features, test_target):
    rfr = RandomForestRegressor()
    rfr.fit(train_features, train_target)
    rfr_predictions = rfr.predict(test_features)
    rfr_mse = mean_squared_error(test_target, rfr_predictions)
    rfr_rmse = np.sqrt(rfr_mse)
    rfr_cvs_scores = cross_val_score(rfr, train_features, train_target, scoring="neg_mean_squared_error", cv=10)
    rfr_rmse_scores = np.sqrt(-rfr_cvs_scores)
    rfr_rmse_mean_score = rfr_rmse_scores.mean()
    rfr_rmse_std_score = rfr_rmse_scores.std()
    return [rfr_rmse, rfr_rmse_mean_score, rfr_rmse_std_score]
```

In [13]:

```
rfr_food_supply_scores = useRandomForestRegressor(food_supply_train_features, food_supply_train_target, food_supply_test_features, food_supply_test_target)
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/ensemble/forest.py:245: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 1
00 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [14]:

```
print("Food Supply - Random Forest Regression - Root Mean Squared Error: {}".format(rfr_food_supply_scores[0]))
print("Food Supply - Random Forest Regression - Mean: {}".format(rfr_food_supply_scores[1]))
print("Food Supply - Random Forest Regression - STD: {}".format(rfr_food_supply_scores[2]))
```

```
Food Supply - Random Forest Regression - Root Mean Squared Error: 3.9081853412462886
Food Supply - Random Forest Regression - Mean: 4.240573854104936
Food Supply - Random Forest Regression - STD: 1.5160876849538794
```

In [15]:

```
hparams_grid= [
    {'n_estimators': [20, 30, 40, 50], 'max_features': [2, 4]},
    {'bootstrap': [False], 'n_estimators': [30, 40, 50, 60], 'max_features': [3, 4]}
]
```

In [16]:

```
rfr_model = RandomForestRegressor(random_state=0)

gs = GridSearchCV(rfr_model, hparams_grid, cv=5,
                 scoring="neg_mean_squared_error",
                 return_train_score=True)
```

In [17]:

```
gs.fit(food_supply_train_features, food_supply_train_target)
```

```
/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_search.py:814: DeprecationWarning: The default of the `iid` parameter will change from True to F
alse in version 0.22 and will be removed in 0.24. This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```

Out[17]:

```
GridSearchCV(cv=5, error_score='raise-deprecating',
             estimator=RandomForestRegressor(bootstrap=True, criterion='mse',
                                             max_depth=None,
                                             max_features='auto',
                                             max_leaf_nodes=None,
                                             min_impurity_decrease=0.0,
                                             min_impurity_split=None,
                                             min_samples_leaf=1,
                                             min_samples_split=2,
                                             min_weight_fraction_leaf=0.0,
                                             n_estimators='warn', n_jobs=None,
                                             oob_score=False, random_state=0,
                                             verbose=0, warm_start=False),
             iid='warn', n_jobs=None,
             param_grid=[{'max_features': [2, 4],
                          'n_estimators': [20, 30, 40, 50]},
                         {'bootstrap': [False], 'max_features': [3, 4],
                          'n_estimators': [30, 40, 50, 60]}],
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring='neg_mean_squared_error', verbose=0)
```

In [18]:

```
gs.best_params_
```

Out[18]:

```
{'bootstrap': False, 'max_features': 3, 'n_estimators': 40}
```

In [19]:

```
gs.best_estimator_
```

Out[19]:

```
RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=None,
                      max_features=3, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=40,
                      n_jobs=None, oob_score=False, random_state=0, verbose=0,
                      warm_start=False)
```

In [20]:

```
cvres = gs.cv_results_
for mean_score, params in zip(cvres['mean_test_score'], cvres['params']):
    print(np.sqrt(-mean_score), params)
```

```
4.776660562992186 {'max_features': 2, 'n_estimators': 20}
4.535641919674319 {'max_features': 2, 'n_estimators': 30}
4.550536123847723 {'max_features': 2, 'n_estimators': 40}
4.456221809469817 {'max_features': 2, 'n_estimators': 50}
4.576296410855213 {'max_features': 4, 'n_estimators': 20}
4.553151967212738 {'max_features': 4, 'n_estimators': 30}
4.504727448579877 {'max_features': 4, 'n_estimators': 40}
4.496003882090672 {'max_features': 4, 'n_estimators': 50}
4.064396041011997 {'bootstrap': False, 'max_features': 3, 'n_estimators': 30}
4.027579602653438 {'bootstrap': False, 'max_features': 3, 'n_estimators': 40}
4.079688012837597 {'bootstrap': False, 'max_features': 3, 'n_estimators': 50}
4.058532621695126 {'bootstrap': False, 'max_features': 3, 'n_estimators': 60}
5.225559903561239 {'bootstrap': False, 'max_features': 4, 'n_estimators': 30}
5.222627636709599 {'bootstrap': False, 'max_features': 4, 'n_estimators': 40}
5.219667983149594 {'bootstrap': False, 'max_features': 4, 'n_estimators': 50}
5.215042840450928 {'bootstrap': False, 'max_features': 4, 'n_estimators': 60}
```

In [21]:

```
food_supply_feature_importances = gs.best_estimator_.feature_importances_
```

In [22]:

```
sorted(zip(food_supply_feature_importances, ['other_use_supply_percentage', 'feed_supply_percentage', 'export_supply_percentage', 'import_supply_percentage']), reverse=Tru
e)
```

Out[22]:

```
[(0.8616603989548646, 'feed_supply_percentage'),
 (0.07441377152083993, 'other_use_supply_percentage'),
 (0.03870460128767835, 'export_supply_percentage'),
 (0.02522122823661716, 'import_supply_percentage')]
```

In [23]:

```
food_supply_model = RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=None,
                      max_features=3, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=40,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [24]:

```
food_supply_model.fit(food_supply_train_features, food_supply_train_target)
```

Out[24]:

```
RandomForestRegressor(bootstrap=False, criterion='mse', max_depth=None,
                      max_features=3, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, n_estimators=40,
                      n_jobs=None, oob_score=False, random_state=None,
                      verbose=0, warm_start=False)
```

In [25]:

```
food_supply_predictions = food_supply_model.predict(food_supply_test_features)
```

In [26]:

```
food_supply_mse = mean_squared_error(food_supply_test_target, food_supply_predictions)
```

In [27]:

```
food_supply_rmse = np.sqrt(food_supply_mse)
```

In [28]:

```
food_supply_rmse
```

Out[28]:

```
2.623906020523008
```

In [29]:

```
food_supply_scores = cross_val_score(food_supply_model, food_supply_test_features, food_supply_test_target,
                           scoring="neg_mean_squared_error", cv=10)
```

In [30]:

```
food_supply_rmse_scores = np.sqrt(-food_supply_scores)
```

In [31]:

```
print("Final Model Mean Scores: {}".format(food_supply_rmse_scores.mean()))
print("Final Model Scores STD: {}".format(food_supply_rmse_scores.std()))
```

```
Final Model Mean Scores: 6.280656022138692
Final Model Scores STD: 2.166475928013373
```

In [32]:

```
df_agriculture_cleaned.tail()
```

Out[32]:

	Producer Price (LCU/tonne)	Price USD	Area harvested	Yield	Domestic Supply	Pesticides	Fertilizer	food_supply_percentage	feed_supply_percentage	export_supply_percentage	other_use_supply_percentage	import_supply_percenta
786	1000.0	3327.917000	2515541.0	44941.0	7708.0	6947.9	491831.0	11.390763	85.236118	28.863335	0.038921	4.060
787	2006.0	19368.042336	2781200.0	42466.0	10481.0	26857.0	721481.0	50.405496	44.986165	25.704851	0.000000	0.534
788	198.9	149.803325	442300.0	110445.0	10421.0	54197.0	1749149.0	0.892429	87.515594	6.526187	10.411669	54.678
789	611.9	1164.915639	659222.0	89499.0	7053.0	39440.0	2312134.0	25.818801	58.131292	6.728814	10.562881	21.976
790	176.0	176.000000	35390550.0	99256.0	292776.0	407779.2	20994055.0	1.337883	43.727628	6.970616	46.801309	1.227

In [42]:

```
increase = -20
export_supply_percentage = 0.152328
import_supply_percentage = -0.084344
feed_supply_percentage = -0.150412
food_supply_percentage = -0.158367
#'other_use_supply_percentage', 'feed_supply_percentage', 'export_supply_percentage', 'import_supply_percentage'
#food_supply_model.predict([[46.8 + increase, 43.7 + (increase*feed_supply_percentage), 8.42 + (increase*export_supply_percentage), 1.22 + (increase*import_supply_percenta
ge)]])
food_supply_model.predict([[46.8, 43.7, 15, 0.5]])
```

Out[42]:

```
array([7.14126168])
```