Solution: Global Supply Chain

Import Libraries and Dataset

In [1]:
```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

In [31]:
```
df = pd.read_csv('data/global_supply_chain.csv')
```

Investigating Missing Values

First, we'll check for how many missing values are in each feature.

In [3]:
```
# Check for missing values
df.isnull().sum()
```

Out[3]:
```
Area        0
Item        0
Y1961    2696
Y1962    2696
Y1963    2696
Y1964    2696
Y1965    2696
Y1966    2696
Y1967    2696
Y1968    2696
Y1969    2696
Y1970    2696
Y1971    2696
Y1972    2696
Y1973    2696
Y1974    2696
Y1975    2696
Y1976    2696
Y1977    2696
Y1978    2696
Y1979    2696
Y1980    2696
```

```
Y1981    2696
Y1982    2696
Y1983    2696
Y1984    2696
Y1985    2696
Y1986    2696
Y1987    2696
Y1988    2696
Y1989    2696
Y1990    2594
Y1991    2594
Y1992     791
Y1993     485
Y1994     485
Y1995     485
Y1996     485
Y1997     485
Y1998     485
Y1999     485
Y2000     289
Y2001     289
Y2002     289
Y2003     289
Y2004     289
Y2005     289
Y2006      94
Y2007      94
Y2008      94
Y2009      94
Y2010      94
Y2011      94
Y2012       0
Y2013       0
dtype: int64
```

One important insight to note is that the number of missing values drops significantly from 1991 to 1992. This corresponds with the dissolution of the Soviet Union in late 1991.

This leads to a **hypothesis:** The missing values are from countries that did not exist at the time. If so, then a good way to handle them would be to create a new feature called `StartYear` to remember the date that we started collecting data for a country and then fill the missing values with 0.

This way, we can proceed with our analysis while still being able to distinguish the two scenarios a `0.0` would show up in the dataset:

1. A country not existing yet, or simply...
2. A country not producing a certain food item that year.

Let's dig into this a bit.

In [4]:
```
# Missing values in 1991 by country
df[df.Y1991.isnull()].Area.value_counts()
```

Out[4]:
```
Ethiopia                                      107
Russian Federation                            102
Kazakhstan                                    101
Belgium                                       101
Slovenia                                      101
Slovakia                                      100
The former Yugoslav Republic of Macedonia     100
Latvia                                        100
Croatia                                        99
Estonia                                        99
Czechia                                        99
Georgia                                        98
Belarus                                        98
Serbia                                         98
Armenia                                        98
Lithuania                                      98
Bosnia and Herzegovina                         97
Republic of Moldova                            97
Montenegro                                     97
Azerbaijan                                     96
Ukraine                                        96
Luxembourg                                     95
Sudan                                          94
Kyrgyzstan                                     93
Uzbekistan                                     91
Tajikistan                                     73
Turkmenistan                                   66
Name: Area, dtype: int64
```

Ok, so it looks like many of the countries are Eastern European countries that were established after the dissolution of the Soviet Union.

But what about a country like Luxembourg? Or, more importantly, what if we don't have the time nor knowledge of history to be able to check each country individually?

Based on the above output, it looks like we're on the right track, but let's refine our hypothesis slightly.

- **Old hypothesis:** The missing values are from countries that did not exist at the time.
- **New hypothesis:** The missing values are from countries that the FAO couldn't collect data from, for some reason or another.

Now, the second hypothesis might seem like cheating. After all, doesn't that cover any possible reason for missing values?

Not quite. In fact, there's a very specific corollary that we can test. The corollary to the updated hypothesis is that if we were to check for missing values *after* a specific year for each country (i.e. `StartYear`), we should not find missing values.

Put in another way, if the new hypothesis is correct, the available data for *all* food categories should start on the same year for Luxembourg or for any other country with missing data. Then, after that `StartYear` , there should be no other missing values.

In [5]:
```
# Drop missing columns if ALL values in the column are missing
df[df.Area == 'Luxembourg'].dropna(how='all', axis=1).head()
```

Out[5]:

	Area	Item	Y2000	Y2001	Y2002	Y2003	Y2004	Y2005	Y2006	Y2007	Y2008	Y2009
9457	Luxembourg	Wheat and products	32.0	31.0	32.0	32.0	34.0	39.0	43.0	47.0	49.0	53.0
9458	Luxembourg	Rice (Milled Equivalent)	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	2.0	2.0
9459	Luxembourg	Barley and products	1.0	2.0	2.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
9460	Luxembourg	Maize and products	3.0	4.0	4.0	4.0	4.0	3.0	3.0	3.0	3.0	3.0
9461	Luxembourg	Rye and products	3.0	3.0	3.0	3.0	2.0	2.0	2.0	2.0	2.0	2.0

As you can see, for Luxembourg, the `StartYear` should be 2000. All of the years before 2000 had ALL missing values for ALL food categories.

Let's systematize this and confirm if our hypothesis holds weight.

In [6]:
```
# Get the earliest year for each country's data
def extract_earliest_data_year(data, country):
    # Drop if ALL values in the column are missing
    keep = data[data.Area == country].dropna(how='all', axis=1)

    # Years of data available
    years = [int(col[1:]) for col in keep.columns
             if col.startswith('Y')]

    # Check if any values are missing after StartYear
    still_missing = keep.isnull().sum().any()

    return min(years), still_missing
```

In [7]:
```
# Example
extract_earliest_data_year(df, 'Luxembourg')
```

Out[7]:
```
(2000, False)
```

The first output is the earliest year for which data is available for a country. The second output is `True` if any values beyond that year are missing. Otherwise, it is `False`, which would support our new hypothesis.

Next, we'll loop through each country and store the year of the earliest data available. We'll also print an alert if that country still has missing values after that start year.

In [8]:
```
# Record the StartYear for each country
start_year_dict = {}

for country in df.Area:
    start_year, still_missing = extract_earliest_data_year(df, country)
    start_year_dict[country] = start_year

    # If we still have missing values, alert
    if still_missing:
        print( country, 'still has missing values.' )
```

No alerts. Our hypothesis is most likely correct, and from an empirical point of view, we can proceed with the analysis assuming it is correct.

The key for take-home challenges (and data science in general) is to **state your assumptions clearly.** This way, you can always review and revise them later, if needed or if new information arises.

Now we also have a handy dictionary that has the data collection start years for each country.

In [9]:
```
start_year_dict['Luxembourg']
```

Out[9]:
```
2000
```

We'll record the `StartYear` for each country, then fill missing values.

In [10]:
```
# Create StartYear feature
df['StartYear'] = df.Area.apply(lambda x: start_year_dict[x] )

# Fill missing values
df.fillna(0.0, inplace=True)
```

Data Wrangling

Next, we'll build an table with the following specifications:

- Total food production (across all food categories)
- Time-series format with years as indices.
- Only contains data from the 12 South American countries.

We'll take it step-by-step. First, we'll need to **aggregate, then transpose** our dataset.

In [11]:
```
# Get the names of "year" columns
year_cols = [col for col in df.columns if col.startswith('Y')]

# Create intermediary table
abt = df[['Area'] + year_cols]

# Aggregate, then transpose
total_df = abt.groupby('Area').sum().transpose()

# Set year as index
total_df.index = [int(col[1:]) for col in year_cols]
```

This is what the new table currently looks like:

In [12]:
```
total_df.head()
```

Out[12]:

Area	Afghanistan	Albania	Algeria	Angola	Antigua and Barbuda	Argentina	Armenia	Australia	Austria	Azerbaijar
1961	8761.0	1612.0	7405.0	4716.0	90.0	33850.0	0.0	17982.0	13003.0	0.0
1962	8694.0	1641.0	7141.0	4657.0	92.0	33231.0	0.0	18636.0	12820.0	0.0
1963	8458.0	1643.0	6798.0	5124.0	103.0	33692.0	0.0	19346.0	13408.0	0.0
1964	9430.0	1767.0	7157.0	5154.0	93.0	34628.0	0.0	19754.0	13499.0	0.0
1965	9753.0	1789.0	7425.0	5399.0	82.0	36863.0	0.0	20087.0	13247.0	0.0

5 rows × 174 columns

Next, we'll keep only the countries from South America. One thing to watch out for is that the country names could be written slightly differently, depending on the source.

This is common thing to watch out for in data science, especially when combining information from different source. In this case, we are using country names from Wikipedia (source B) to filter data from our table (source A).

As a result, we should check to make sure the names are written correctly:

In [13]:
```
# List of country names from Wikipedia
countries_list = ['Brazil', 'Colombia', 'Argentina', 'Venezuela',
                  'Peru', 'Chile', 'Ecuador', 'Bolivia',
                  'Paraguay', 'Uruguay', 'Guyana', 'Suriname']
```

In [14]:
```
# Check that each country is in the dataset
for country in countries_list:
    if country not in df.Area.unique():
        print( country, 'not in dataset' )
```

```
Venezuela not in dataset
Bolivia not in dataset
```

Good thing we checked! Now the question is if those countries are actually in the dataset.

There are more advanced regex ways to check, but those are overkill for this scenario. All we really need to do is print the country names out in alphabetical order and scan them quickly.

In [15]:
```
# Country names in the dataset
df.Area.unique()
```

Out[15]:
```
array(['Afghanistan', 'Albania', 'Algeria', 'Angola',
       'Antigua and Barbuda', 'Argentina', 'Armenia', 'Australia',
       'Austria', 'Azerbaijan', 'Bahamas', 'Bangladesh', 'Barbados',
       'Belarus', 'Belgium', 'Belize', 'Benin', 'Bermuda',
       'Bolivia (Plurinational State of)', 'Bosnia and Herzegovina',
       'Botswana', 'Brazil', 'Brunei Darussalam', 'Bulgaria',
       'Burkina Faso', 'Cabo Verde', 'Cambodia', 'Cameroon', 'Canada',
       'Central African Republic', 'Chad', 'Chile',
       'China, Hong Kong SAR', 'China, Macao SAR', 'China, mainland',
       'China, Taiwan Province of', 'Colombia', 'Congo', 'Costa Rica',
       "Côte d'Ivoire", 'Croatia', 'Cuba', 'Cyprus', 'Czechia',
       "Democratic People's Republic of Korea", 'Denmark', 'Djibouti',
       'Dominica', 'Dominican Republic', 'Ecuador', 'Egypt',
       'El Salvador', 'Estonia', 'Ethiopia', 'Fiji', 'Finland', 'France'
       'French Polynesia', 'Gabon', 'Gambia', 'Georgia', 'Germany',
       'Ghana', 'Greece', 'Grenada', 'Guatemala', 'Guinea',
       'Guinea-Bissau', 'Guyana', 'Haiti', 'Honduras', 'Hungary',
       'Iceland', 'India', 'Indonesia', 'Iran (Islamic Republic of)',
       'Iraq', 'Ireland', 'Israel', 'Italy', 'Jamaica', 'Japan', 'Jordan
       'Kazakhstan', 'Kenya', 'Kiribati', 'Kuwait', 'Kyrgyzstan',
       "Lao People's Democratic Republic", 'Latvia', 'Lebanon', 'Lesothc
       'Liberia', 'Lithuania', 'Luxembourg', 'Madagascar', 'Malawi',
       'Malaysia', 'Maldives', 'Mali', 'Malta', 'Mauritania', 'Mauritius
       'Mexico', 'Mongolia', 'Montenegro', 'Morocco', 'Mozambique',
```

```
       'Myanmar', 'Namibia', 'Nepal', 'Netherlands', 'New Caledonia',
       'New Zealand', 'Nicaragua', 'Niger', 'Nigeria', 'Norway', 'Oman',
       'Pakistan', 'Panama', 'Paraguay', 'Peru', 'Philippines', 'Poland'
       'Portugal', 'Republic of Korea', 'Republic of Moldova', 'Romania'
       'Russian Federation', 'Rwanda', 'Saint Kitts and Nevis',
       'Saint Lucia', 'Saint Vincent and the Grenadines', 'Samoa',
       'Sao Tome and Principe', 'Saudi Arabia', 'Senegal', 'Serbia',
       'Sierra Leone', 'Slovakia', 'Slovenia', 'Solomon Islands',
       'South Africa', 'Spain', 'Sri Lanka', 'Sudan', 'Suriname',
       'Swaziland', 'Sweden', 'Switzerland', 'Tajikistan', 'Thailand',
       'The former Yugoslav Republic of Macedonia', 'Timor-Leste', 'Togc
       'Trinidad and Tobago', 'Tunisia', 'Turkey', 'Turkmenistan',
       'Uganda', 'Ukraine', 'United Arab Emirates', 'United Kingdom',
       'United Republic of Tanzania', 'United States of America',
       'Uruguay', 'Uzbekistan', 'Vanuatu',
       'Venezuela (Bolivarian Republic of)', 'Viet Nam', 'Yemen',
       'Zambia', 'Zimbabwe'], dtype=object)
```

As you can see, "Venezual" and "Bolivia" are written as `'Venezuela (Bolivarian Republic of)'` and `'Bolivia (Plurinational State of)'` in our dataset.

Let's revise our list.

In [16]:
```
countries_list = ['Brazil', 'Colombia', 'Argentina',
                   'Venezuela (Bolivarian Republic of)',
                  'Peru', 'Chile', 'Ecuador',
                   'Bolivia (Plurinational State of)',
                  'Paraguay', 'Uruguay', 'Guyana', 'Suriname']
```

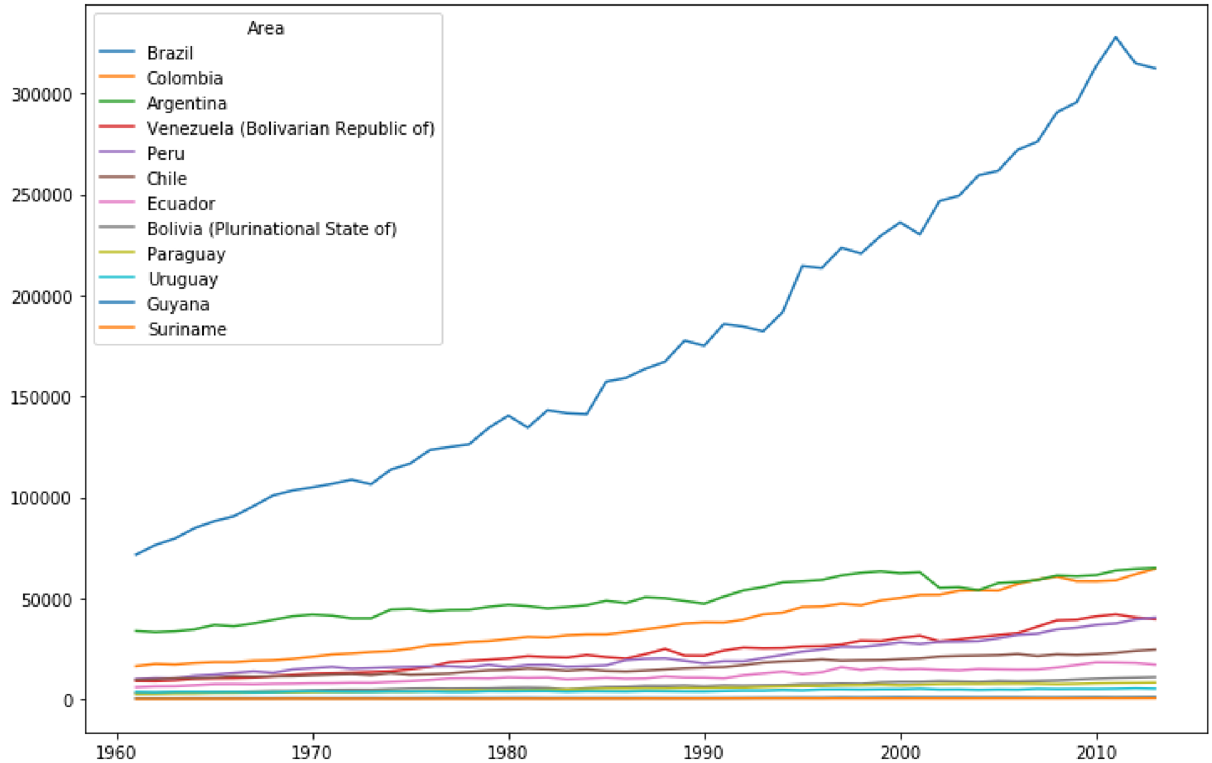
Now we can filter to South American countries.

In [17]:
```
sa_df = total_df[countries_list]
```

Data Visualization

Plotting the initial time series chart is straightforward:

In [18]:
```
# Time series plot
sa_df.plot(figsize=(12,8))
plt.show()
```



Whoah, look at Brazil's growth! We're done here, right? We've answered the question... let's go home.

But wait!

Instead of just looking at absolute growth, let's dig a bit deeper and consider the question from two other angles:

- Year-over-year (YoY) growth
- Overall percent growth

We can calculate and plot **year-over-year growth** like so.

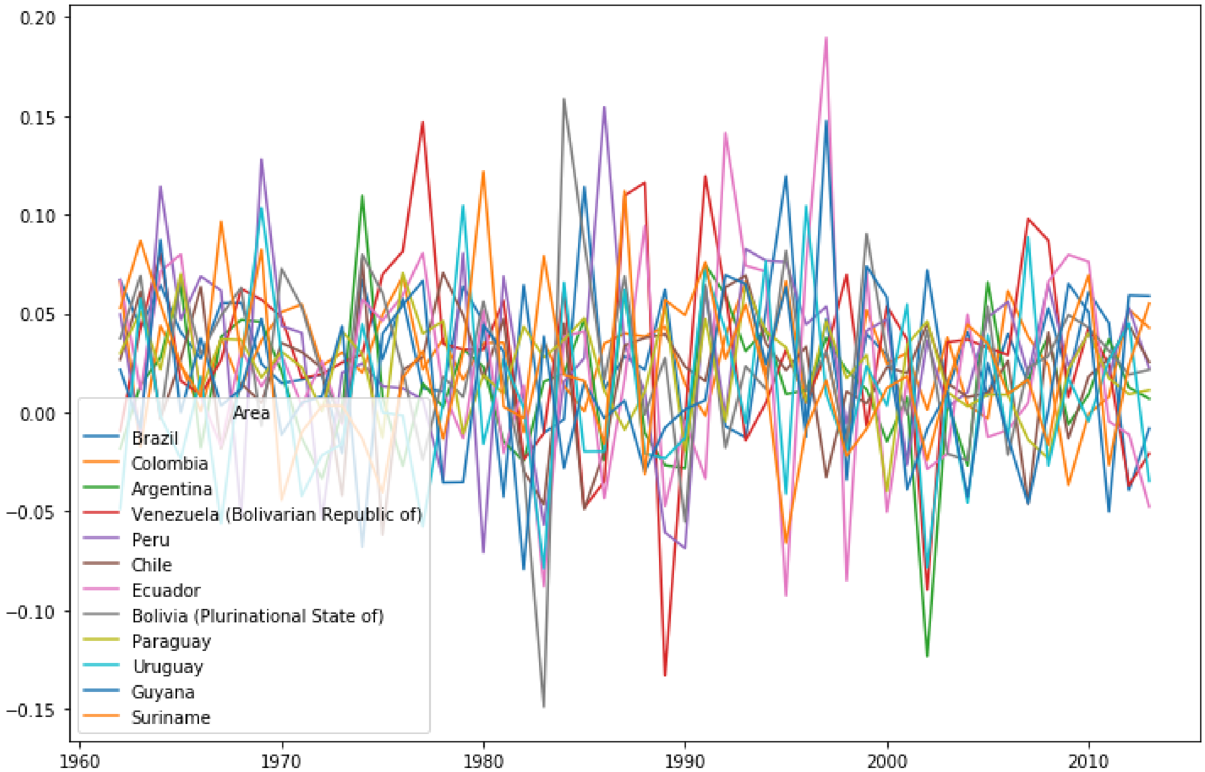
Tip: If you come from a finance background, you may be inclined to calculate something like **log-returns**. Log-returns have several useful properties, such as alleviating affects from *auto-correlation*. However, for this exercise, they are unnecessary because of the challenge's objective. We opt for the more easily interpretable year-over-year growth instead. In the Crypto Time Series challenge, we will explore time series analysis with log-returns.

In [19]:
```
# Year-over-year growth
```

```
yoy_df = sa_df / sa_df.shift(1) - 1.0
```

In [20]:

```
yoy_df.plot(figsize=(12,8))
plt.show()
```

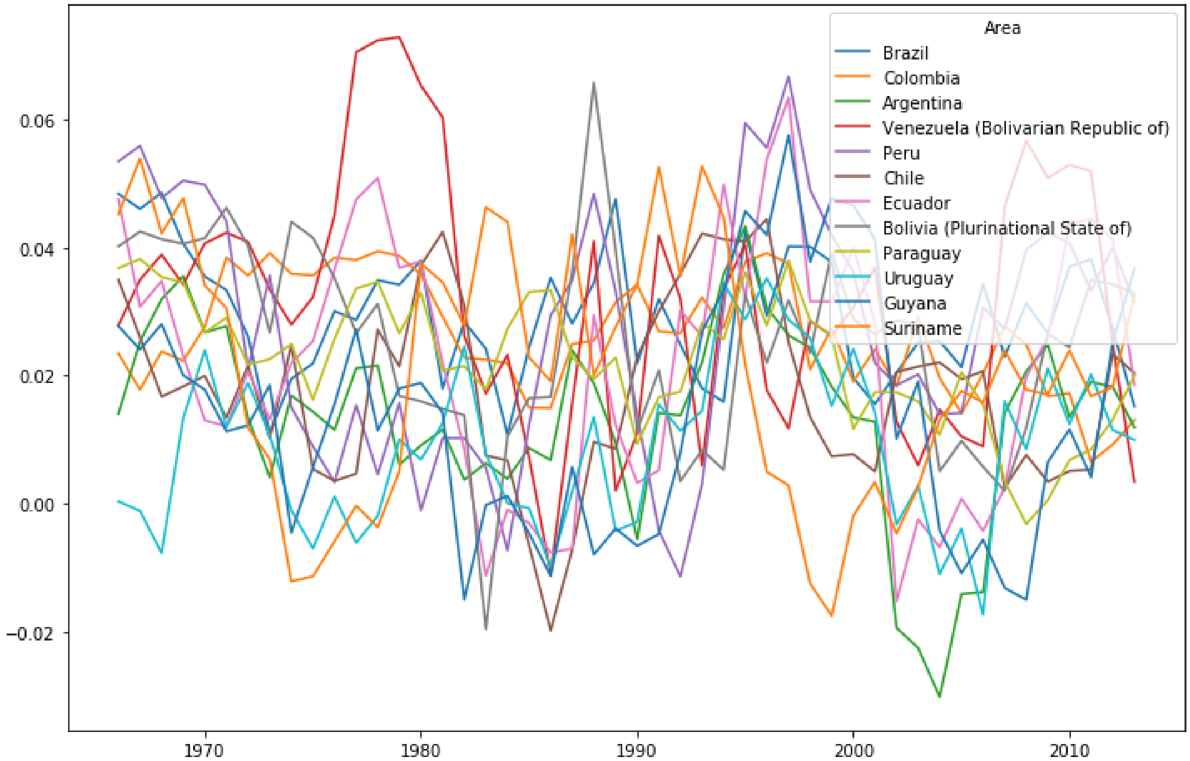


There's much higher variance, and we can see that the answer doesn't look quite so clear-cut.

We can see if looking at 5-year moving averages for YoY growth make things any clearer:

In [21]:

```
rolling_df = yoy_df.rolling(window=5).mean()
rolling_df.plot(figsize=(12,8))
plt.show()
```



Still too noisy, but it's still helpful overall to look at these plots before digging into the numbers and statistics. A quick look at charts can often help you spot outiers, errors, or anything else that looks plain weird.

But to better understand what's really going on, we should look at the summary statistics.

In [22]:
```
# YoY growth summary statistics
yoy_df.describe()
```

Out[22]:

Area	Brazil	Colombia	Argentina	Venezuela (Bolivarian Republic of)	Peru	Chile	Ecuador	Bolivia (Plurinational State of)	Para
count	52.000000	52.000000	52.000000	52.000000	52.000000	52.000000	52.000000	52.000000	52.0
mean	0.029309	0.026952	0.013286	0.029828	0.028159	0.018638	0.021750	0.026077	0.02
std	0.034705	0.023743	0.036094	0.050269	0.047822	0.032633	0.055567	0.045637	0.02
min	-0.042603	-0.036675	-0.123183	-0.132979	-0.070694	-0.061863	-0.092653	-0.148936	-0.0
25%	0.010147	0.011449	-0.011040	0.005439	0.006361	0.007023	-0.012620	-0.001075	0.00
50%	0.027853	0.030510	0.012108	0.031718	0.030650	0.023029	0.014965	0.025415	0.02
75%	0.055339	0.040702	0.037191	0.057874	0.054201	0.038140	0.066304	0.054257	0.04
max	0.119551	0.068585	0.109601	0.146766	0.154353	0.077653	0.189422	0.158402	0.07

Aha! While Brazil is definitely one of the leaders in average/median YoY growth, we can see that several other countries have grown just as quickly:

- Colombia
- Venezuela
- Peru

And even Bolivia to some extent.

If we do the calculations for overall percentage growth, we should see those 4 countries having comparable numbers to Brazil's.

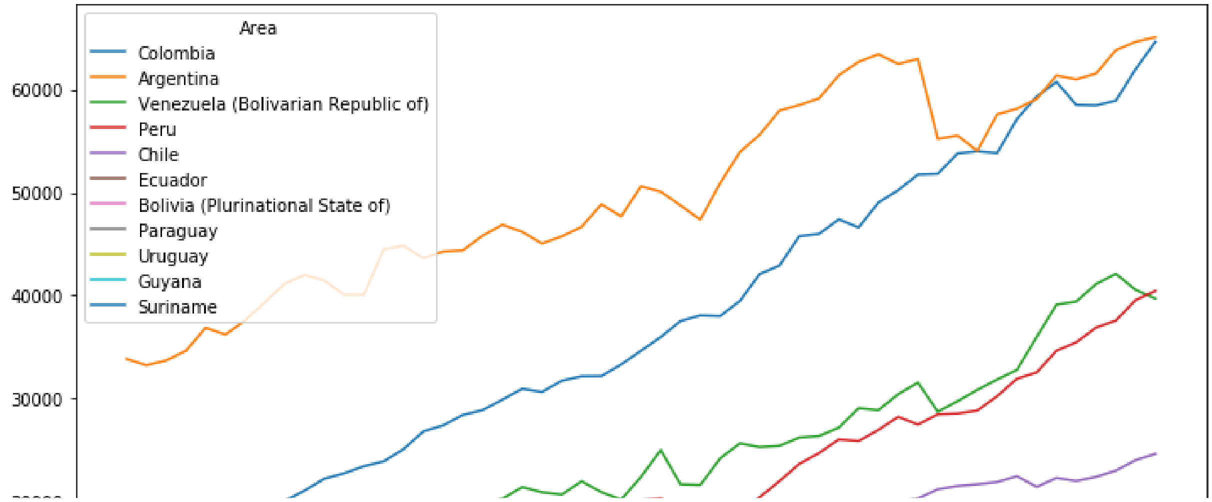
In [23]:
```
# Overall percentage growth
(sa_df.loc[2013] / sa_df.loc[1961] - 1.0) * 100
```

Out[23]:
```
Area
Brazil                                  336.338248
Colombia                                293.239214
Argentina                                92.209749
Venezuela (Bolivarian Republic of)      333.329696
Peru                                    300.901972
Chile                                   154.382162
Ecuador                                 184.017887
Bolivia (Plurinational State of)        262.134647
Paraguay                                215.837448
Uruguay                                  45.404814
Guyana                                  108.644401
Suriname                                203.846154
dtype: float64
```

This is an important lesson for take-home challenges: **Be careful of the seemingly "obvious" answer.** One of the roles of a data scientist is to "be the skeptic." Charts are a good starting point, but sometimes they are not enough. You must be willing to dig a little deeper and double-check using the data.

In fact, if we remove Brazil from the plot, we'll see that several other countries saw fast food production growth as well (they just appeared "squashed" in the original chart due to Brazil's higher starting point).

In [24]:
```
sa_df.drop('Brazil', axis=1).plot(figsize=(12,8))
plt.show()
```



Correlations

One way to look for competition is calculate correlations between different countries' food production.

However, we'll want to investigate absolute production *and* year-over-year growth, as they tell different stories.

Let's calculate and plot everything in one go.

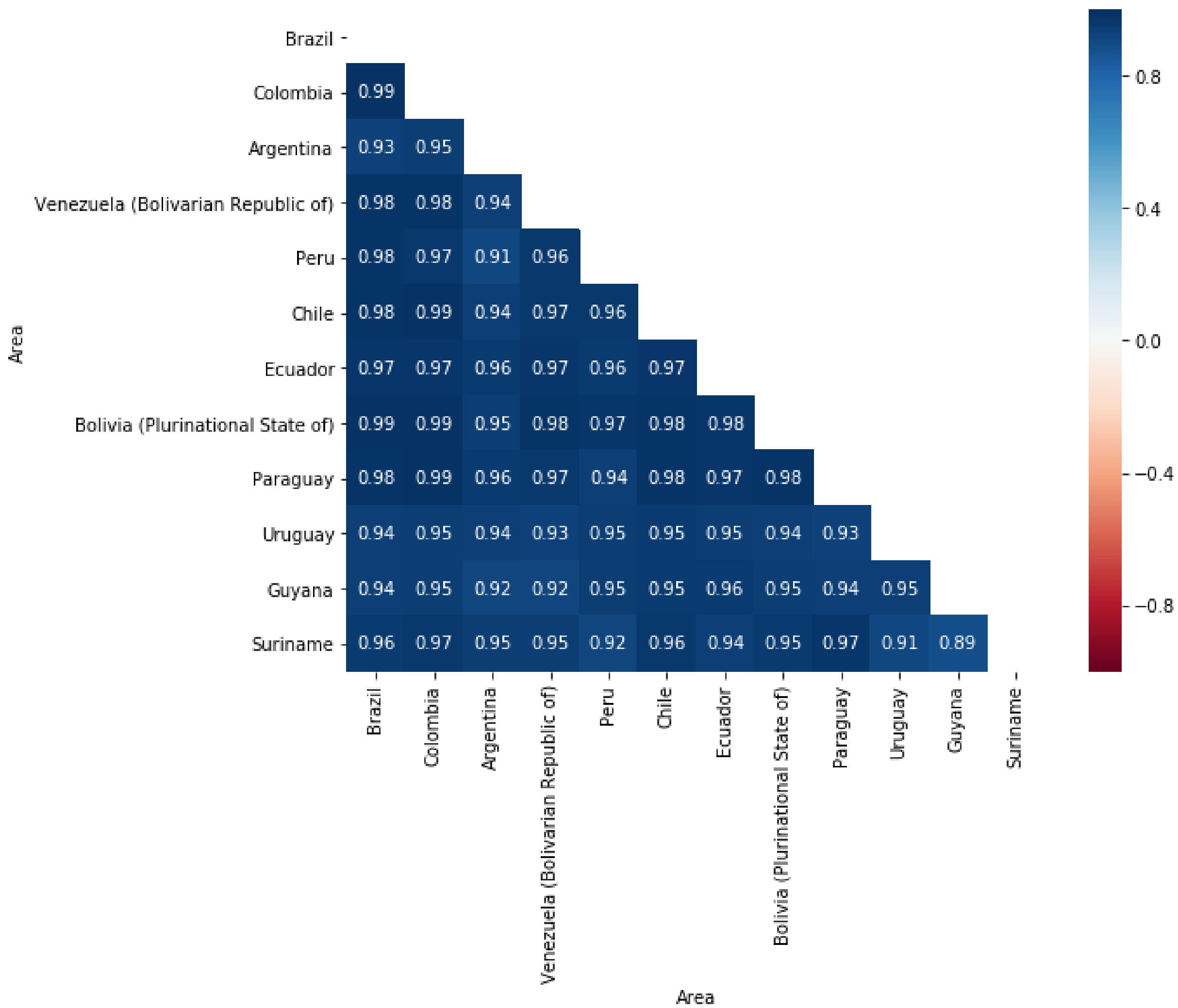
In [25]:
```
# Absolute production
sa_corr = sa_df.corr()

# YoY growth
yoy_corr = yoy_df.corr()

# YoY growth 5-yr window
rolling_corr = rolling_df.corr()
```

In [26]:
```
def plot_corr_heatmap(corr):
    plt.figure(figsize=(9,7))

    # Make to cover top triangle
    mask = np.zeros_like(corr)
    mask[np.triu_indices_from(mask)] = True

    # Plot heatmap
    sns.heatmap(corr,
                vmin=-1.0,
                vmax=1.0,
                mask=mask,
                cmap='RdBu',
                annot=True)
    plt.show()
```

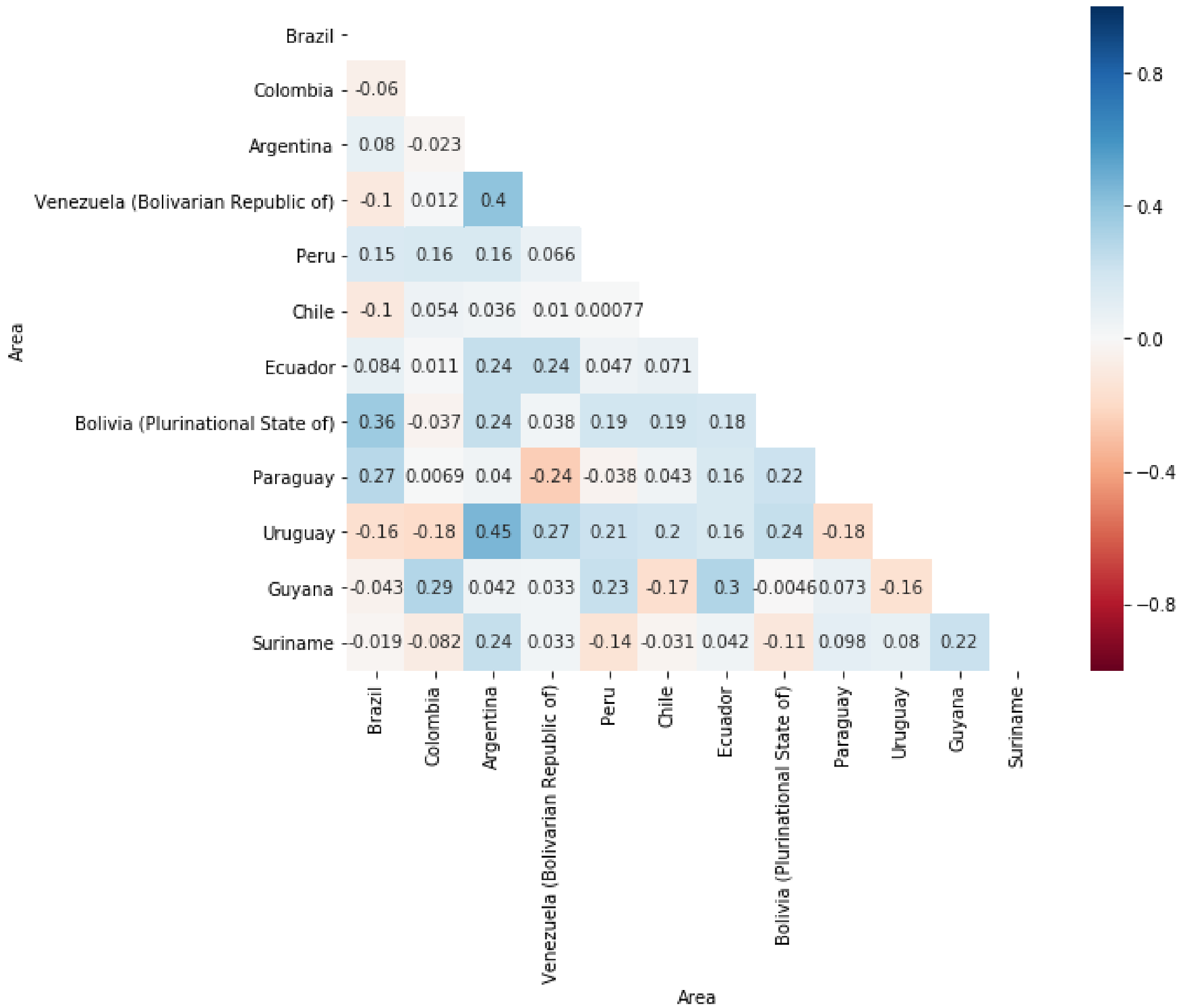
Here we defined our plotting function. For a series of correlation heatmaps, there are a few important parameters to set:

- A diverging color-map (to reflect positive and negative correlations)
- Consistent min of -1.0 and max of 1.0 on of the color-map scale (to reflect the range of possible correlations)
- Annotations of the actual correlation values (just helpful to see in one place)

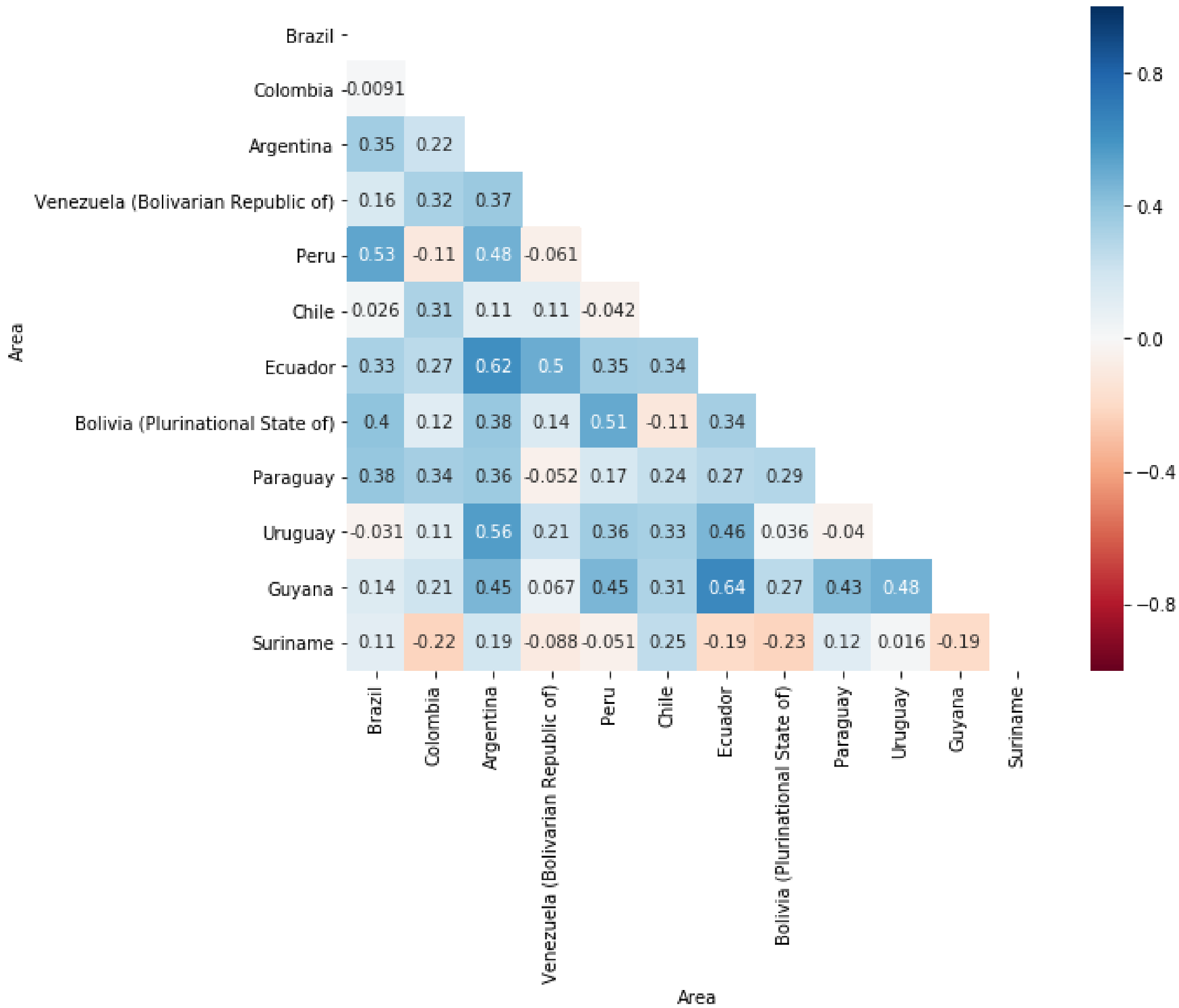
In [27]:
```
# Absolute production
plot_corr_heatmap(sa_corr)
```



In [28]:
```
# YoY Growth
plot_corr_heatmap(yoy_corr)
```



In [29]:
```
# 5-year rolling YoY growth average
plot_corr_heatmap(rolling_corr)
```



The results are very interesting.

First, we can see that the overall absolute food production amounts are highly correlated between all countries in the region. This makes intuitive sense, as these numbers should be heavily influenced by population growth, overall region prosperity, technological advancements, and so on.

On the other hand, the correlations for YoY growth are much weaker (and negative in some cases). This indicates that while production levels for the continent are correlated overall, the year-over-year growth *rates* are not. It could indicate competition or other factors such as different paces of technological adoption.

Finally, the 5-year moving averages for YoY growth appear to be somewhere in the middle, which also makes sense.

Ways to Improve Analysis

There are many correct answers. Here are several ideas:

- **Granularity:** Break down the data into individual food categories. Then, systematically re-run the analysis for each food category to understand what's going on at a deeper level.
- **Scope:** Expand analysis to additional countries (especially when looking at correlations and signs of possible competition). This would be especially useful when combined with improved granularity.
- **External Data:** Layer in macro-economic indicators such as GDP growth, education access, unemployment rates, etc. You can even include proxies for technological adoption (such as mobile coverage) if you cannot find direct data for agricultural tech.