Introduction

This is an explanatory analysis of US freight data made with following purposes:

- to make animated visualization with help of matplotlib and pyplot,
- find some interesting trends and search additional explanatory information in the web to support or challenge them.

I was interested in money measurement of data so I took columns with values (value_2012, value_2013 and so on). The code below can be adapted to other measurements like tons or miles.

In [1]:

```
# load libraries and set basic options
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
import shapefile as shp
import plotly.graph_objs as go
import random
import io
import base64

from matplotlib.animation import FFMpegWriter
from matplotlib import gridspec
from matplotlib.patches import Polygon
from matplotlib import animation, rc, rcParams
from matplotlib.collections import PatchCollec
tion
from mpl_toolkits.basemap import Basemap

from plotly import __version__
from plotly.offline import download_plotlyjs,
init_notebook_mode, plot, iplot
from itertools import chain

#%matplotlib inline
from IPython.display import HTML, Image
rc('animation', html='html5')

init_notebook_mode(connected=True)
np.set_printoptions(formatter={'int_kind': '
{:,}'.format})
```

File "FAF4 User Guide" provides detailed explanations for

File "FAF4 User Guide" provides detailed explanations for data in tables. For this analysis, I shall use file "FAF4_Regional" as it provides data by regions which I think is more interesting for visualization on maps.

Data from user guide will be implemented through dictionaries and lists below.

In [2]:

```
# prepare commodity dictionary based on User Gu
ide
com_list = ['Animals and Fish (live)', 'Cereal
Grains (includes seed)',
            'Agricultural Products (excludes A
nimal Feed, Cereal Grains, and Forage Product
s)',
            'Animal Feed, Eggs, Honey, and Oth
er Products of Animal Origin',
            'Meat, Poultry, Fish, Seafood, and
Their Preparations',
            'Milled Grain Products and Prepara
tions, and Bakery Products',
            'Other Prepared Foodstuffs, Fats a
nd Oils',
            'Alcoholic Beverages and Denatured
Alcohol',
            'Tobacco Products', 'Monumental or
Building Stone',
            'Natural Sands', 'Gravel and Crush
ed Stone (excludes Dolomite and Slate)',
            'Other Non-Metallic Minerals not e
lsewhere classified',
            'Metallic Ores and Concentrates',
'Coal', 'Crude Petroleum',
            'Gasoline, Aviation Turbine Fuel,
 and Ethanol (includes Kerosene, and Fuel Alco
hols)',
            'Fuel Oils (includes Diesel, Bunke
r C, and Biodiesel)',
            'Other Coal and Petroleum Product
s, not elsewhere classified',
            'Basic Chemicals', 'Pharmaceutical
Products', 'Fertilizers',
            'Other Chemical Products and Prepa
rations', 'Plastics and Rubber',
            'Logs and Other Wood in the Rough'
, 'Wood Products',
            'Pulp, Newsprint, Paper, and Paper
board', 'Paper or Paperboard Articles',
            'Printed Products', 'Textiles, Lea
ther, and Articles of Textiles or Leather',
            'Non-Metallic Mineral Products',
            'Base Metal in Primary or Semi-Fin
ished Forms and in Finished Basic Shapes',
            'Articles of Base Metal', 'Machine
ry',
            'Electronic and Other Electrical E
quipment and Components, and Office Equipment'
,
            'Motorized and Other Vehicles (inc
ludes parts)', 'Transportation Equipment, not
 elsewhere classified',
            'Precision Instruments and Apparat
us',
            'Furniture, Mattresses and Mattres
s Supports, Lamps, Lighting Fittings, and Illu
minated Signs',
            'Miscellaneous Manufactured Produc
ts', 'Waste and Scrap (excludes of agriculture
or food)',
            'Mixed Freight', 'Commodity unknow
n']

com_dict = {}
for i in range(1,len(com_list)+1):
    com_dict[i] = com_list[i-1]

# make sure that dictionary has same numeration
as User Guide
com_dict[99] = com_dict.pop(len(com_list))
com_dict[43] = com_dict.pop(42)

# dictionary of foreign trading partners
fr_dict = {801: "Canada", 802: "Mexico",
          803: "Rest of Americas", 804: "Europ
e",
          805: "Africa", 806: "SW & Central As
ia",
          807: "Eastern Asia", 808: "SE Asia &
Oceania",
          'total': "Total"}

# transportation mode dictionary and list
mode_dict = {1: "Truck", 2: "Rail", 3: "Water"
, 4: "Air",
             5: "Multimode and mail", 6: "Pipe
line",
             7: "Other/unknown", 8: "No domest
ic mode"}

mode_list = ["Truck", "Rail", "Water", "Air",
             "Multimode and mail", "Pipeline",
             "Other/unknown"]

years = ['2012', '2013', '2014','2015','2020',
         '2025', '2030', '2035', '2040', '204
5']
```

In [3]:

```
regions_df = pd.read_csv('../input/FAF4_Region
al.csv',
                         dtype = {'dms_orig':
str, 'dms_dest': str})
reader = shp.Reader('../input/CFS_AREA_shapefi
le_010215/CFS_AREA_shapefile_010215')
```

Choropleth maps

Choropleth maps provide good visualization for data that can be combined with polygons from shapefiles. Just like this database.

I decided to make choropleth map with animation to show changes from year 2012 to year 2045 (forecasted data was provided).

The following visualizations were made:

- Dynamic of total value of commodities originated in each region. This gives sense of production activity with clear presentation of production clusters around Los-Angeles, Houston, Dallas, Chicago, New-York and other big cities.
- Dynamic of total value of commodities by destination region. This shows consumption potential which may differ from production capabilities if the region primary source of income is services.
- Balance of production and consumption per each region. Here production disbalances can be traced. Once again, these disbalances can be covered by revenue from services provided or capital inflows in any of its forms.

Few notes regarding animated maps.

The best way to present the code for animation is to make function or class and then use it for each set of parameters (dataframe, color, subtitle text etc.). This what I've done in Jupyter notebook. But for some reason it just doesn't work within this kernel. The output is only last frame and it seems that animation function doesn't work if it is nested inside other function. Thus, the code below is overblown. If you would like to use it for some other purposes it can be done properly - through nested function (or maybe class).

Maps below use one colorbar for all years (2012-2045). It is good for showing dynamic through final year, to show how some regions are predicted to grow their productions while others expected to stall. But this approach makes first few years pallid. So, in case you are interested in shorter horizon, list of years should be shortened as well as number of frames.

In [4]:

```
#Following sources helped a lot in making this animation:
#https://jakevdp.github.io/blog/2012/08/18/matplotlib-animation-tutorial/
#http://louistiao.me/posts/notebooks/embedding-matplotlib-animations-in-jupyter-notebooks/
#http://louistiao.me/posts/notebooks/save-matplotlib-animations-as-gifs/
#http://matplotlib.org/api/_as_gen/matplotlib.animation.FuncAnimation.html#matplotlib.animation.FuncAnimation
#https://www.kaggle.com/kostyabahshetsyan/animated-choropleth-map-for-crime-statistic/code
#https://www.kaggle.com/jaeyoonpark/heatmap-animation-us-drought-map/code
```

In [5]:

```
# domestic origin by value 2012-2045
```

In [6]:

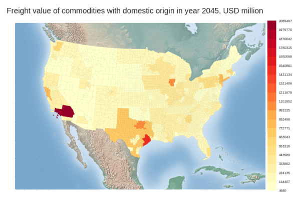
```
dff = dom_origin_df
category = 'dms_orig'
map_color = 'YlOrRd'
num_colors = 20
text = 'commodities with domestic origin'

# animated plot
cat_val_df = dff[[category,'value_2012',
                     'value_2013', 'value_201
4',
                     'value_2015', 'value_202
0',
                     'value_2025', 'value_203
0',
                     'value_2035', 'value_204
0',
                     'value_2045']].groupby(ca
tegory, as_index = False).sum()

cat_val_df.columns = [category] + years

# color map for values origins (applicable to a
ll periods)
values_df = cat_val_df[years]
values = values_df.values.reshape((cat_val_df.
shape[0]*10,1))

# set up steady layer for the map
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(111)

# shapefile data and map setups
m = Basemap(width = 6000000, height = 4000000,
projection = 'lcc',
                resolution = None, lat_1=27.,l
at_2=32,lat_0=37,lon_0=-97.)
m.shadedrelief()
m.readshapefile('../input/CFS_AREA_shapefile_0
10215//CFS_AREA_shapefile_010215',
                    'regions', drawbounds = Tr
ue, linewidth=.01)

# set up steady legend for the map
cm = plt.get_cmap(map_color)
scheme = [cm(q*1.0/num_colors) for q in range(
num_colors)]
bins = np.linspace(values.min(), values.max(),
num_colors)
ax_legend = fig.add_axes([0.8,0.12,0.02,0.77])
cmap = mpl.colors.ListedColormap(scheme)
cb = mpl.colorbar.ColorbarBase(ax_legend, cmap
= cmap, ticks = bins,
                                 boundaries = bi
ns, orientation = 'vertical')
```

```
cb.ax.tick_params(labelsize=9)

# initial set up for polygons
chor_map = ax.plot([],[])[0]
for shape in m.regions:
```

```
        ax.add_collection(pc)

# animated elements of the map
def animate(j):
    year = years[j-1]
    fig.suptitle('Freight value of {} in year
{}, USD million'.format(text, year), fontsize=
20, y=.95)
    dms_orig_val_animated = cat_val_df.loc[:,[
category, year]]
    dms_orig_val_animated.set_index(category,
inplace = True)
    dms_orig_val_animated['bin'] = np.digitize
(dms_orig_val_animated[year], bins) - 1

    for info, shape in zip(m.regions_info, m.r
egions):
        name = info['CFS07DDGEO']
        # some names of the regions are missta
ted in database or in shapefile
        # the purpose of if... loop is to make
this names equal
        # -----------------------------------
------------------------------
        if name == '349':
            name = '342'
        elif name == '100':
            name = '101'
        elif name == '330':
            name = '339'
        elif name == '310':
            name = '311'
        else:
            name
        # -----------------------------------
------------------------------
        # Alaska and Hawaii were excluded from
analysis to make map size reasonable
        if name not in ['020', '159', '151']:
            color = scheme[dms_orig_val_animat
ed.loc[name]['bin'].astype(int)]
            patches = [Polygon(np.array(shape
), True)]
            pc = PatchCollection(patches)
            pc.set_facecolor(color)
            ax.add_collection(pc)
    return chor_map,

anim = animation.FuncAnimation(fig, func = ani
mate, frames = 10,
                               repeat_delay =
2000, interval = 1000)

anim.save('freight_d_d.gif', writer='imagemagi
ck')
```

```
Image(url='freight_d_d.gif', width = 1200, hei
ght = 1000)
```

Out[6]:


Freight value of commodities with domestic origin in year 2045, USD million

In [7]:

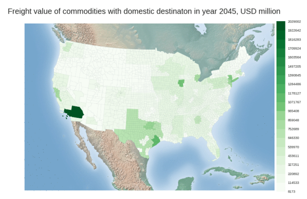
```
# domestic destination by value 2012-2045
dom_dest_df = regions_df.loc[pd.isnull(regions
_df['fr_dest'])]
```

In [8]:

```
dff = dom_dest_df
category = 'dms_dest'
map_color = 'Greens'
num_colors = 20
text = 'commodities with domestic destinaton'

# animated plot
cat_val_df = dff[[category,'value_2012',
                     'value_2013', 'value_201
4',
                     'value_2015', 'value_202
0',
                     'value_2025', 'value_203
0',
                     'value_2035', 'value_204
0',
                     'value_2045']].groupby(ca
tegory, as_index = False).sum()

cat_val_df.columns = [category] + years

# color map for values origins (applicable to a
ll periods)
values_df = cat_val_df[years]
values = values_df.values.reshape((cat_val_df.
shape[0]*10,1))

# set up steady layer for the map
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(111)

# shapefile data and map setups
m = Basemap(width = 6000000, height = 4000000,
projection = 'lcc',
                resolution = None, lat_1=27.,l
at_2=32,lat_0=37,lon_0=-97.)
m.shadedrelief()
m.readshapefile('../input/CFS_AREA_shapefile_0
10215//CFS_AREA_shapefile_010215',
                    'regions', drawbounds = Tr
ue, linewidth=.01)
```

```
# set up steady legend for the map
cm = plt.get_cmap(map_color)
scheme = [cm(q*1.0/num_colors) for q in range(
num_colors)]
bins = np.linspace(values.min(), values.max(),
num_colors)
ax_legend = fig.add_axes([0.8,0.12,0.02,0.77])
cmap = mpl.colors.ListedColormap(scheme)
cb = mpl.colorbar.ColorbarBase(ax_legend, cmap
= cmap, ticks = bins,
                               boundaries = bi
ns, orientation = 'vertical')
cb.ax.tick_params(labelsize=9)

# initial set up for polygons
chor_map = ax.plot([],[])[0]
for shape in m.regions:
        patches = [Polygon(np.array(shape), Tr
ue)]
        pc = PatchCollection(patches)
        ax.add_collection(pc)

# animated elements of the map
def animate(j):
    year = years[j-1]
    fig.suptitle('Freight value of {} in year
{}, USD million'.format(text, year), fontsize=
20, y=.95)
    dms_orig_val_animated = cat_val_df.loc[:,[
category, year]]
    dms_orig_val_animated.set_index(category,
inplace = True)
    dms_orig_val_animated['bin'] = np.digitize
(dms_orig_val_animated[year], bins) - 1

    for info, shape in zip(m.regions_info, m.r
egions):
        name = info['CFS07DDGEO']
        # some names of the regions are missta
ted in database or in shapefile
        # the purpose of if... loop is to make
this names equal
        # ------------------------------------
-------------------------------
        if name == '349':
            name = '342'
        elif name == '100':
            name = '101'
        elif name == '330':
            name = '339'
        elif name == '310':
            name = '311'
        else:
            name
        # ------------------------------------
-------------------------------
        # Alaska and Hawaii were excluded from
analysis to make map size reasonable
        if name not in ['020', '159', '151']:
            color = scheme[dms_orig_val_animat
ed.loc[name]['bin'].astype(int)]
            patches = [Polygon(np.array(shape
), True)]
            pc = PatchCollection(patches)
            pc.set_facecolor(color)
            ax.add_collection(pc)
    return chor_map,

anim = animation.FuncAnimation(fig, func = ani
mate, frames = 10,
                               repeat_delay =
2000, interval = 1000)

anim.save('freight_d_o.gif', writer='imagemagi
ck')
plt.close()
Image(url='freight_d_o.gif', width = 1200, hei
ght = 1000)
```

Out[8]:


Freight value of commodities with domestic destinaton in year 2045, USD million

In [9]:

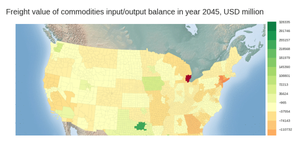
```
# dataframe for freight balance (outflow from r
egoin minus inflow to region)
dom_origin_bal_df = dom_origin_df[['dms_orig',
'value_2012',
                                'value_2013',
'value_2014',
                                'value_2015',
'value_2020',
                                'value_2025',
'value_2030',
                                'value_2035',
'value_2040',
                                'value_2045']]
.groupby('dms_orig', as_index = True).sum()

dom_dest_bal_df = dom_dest_df[['dms_dest','val
ue_2012',
                                'value_2013',
'value_2014',
                                'value_2015',
'value_2020',
                                'value_2025',
'value_2030',
                                'value_2035',
'value_2040',
                                'value_2045']]
.groupby('dms_dest', as_index = True).sum()

dom_dest_bal_df = dom_dest_bal_df.apply(lambda
x: x*(-1))

balance_df = dom_origin_bal_df.add(dom_dest_ba
l_df, fill_value = 0.0)
balance_df.reset_index(inplace = True)
```

In [10]:

```
dff = balance_df
category = 'dms_orig'
map_color = 'RdYlGn'
num_colors = 20
text = 'commodities input/output balance'

# animated plot
cat_val_df = dff[[category,'value_2012',
                     'value_2013', 'value_201
4',
                     'value_2015', 'value_202
0',
                     'value_2025', 'value_203
0',
                     'value_2035', 'value_204
0',
                     'value_2045']].groupby(ca
tegory, as_index = False).sum()

cat_val_df.columns = [category] + years

# color map for values origins (applicable to a
ll periods)
values_df = cat_val_df[years]
values = values_df.values.reshape((cat_val_df.
shape[0]*10,1))

# set up steady layer for the map
fig = plt.figure(figsize=(16, 8))
ax = fig.add_subplot(111)

# shapefile data and map setups
m = Basemap(width = 6000000, height = 4000000,
projection = 'lcc',
                resolution = None, lat_1=27.,l
at_2=32,lat_0=37,lon_0=-97.)
m.shadedrelief()
m.readshapefile('../input/CFS_AREA_shapefile_0
10215//CFS_AREA_shapefile_010215',
                    'regions', drawbounds = Tr
ue, linewidth=.01)

# set up steady legend for the map
cm = plt.get_cmap(map_color)
scheme = [cm(q*1.0/num_colors) for q in range(
num_colors)]
bins = np.linspace(values.min(), values.max(),
num_colors)
ax_legend = fig.add_axes([0.8,0.12,0.02,0.77])
cmap = mpl.colors.ListedColormap(scheme)
cb = mpl.colorbar.ColorbarBase(ax_legend, cmap
= cmap, ticks = bins,
                           boundaries = bi
ns, orientation = 'vertical')
cb.ax.tick_params(labelsize=9)

# initial set up for polygons
chor_map = ax.plot([],[])[0]
for shape in m.regions:
        patches = [Polygon(np.array(shape), Tr
ue)]
        pc = PatchCollection(patches)
        ax.add_collection(pc)
```

```
# animated elements of the map
def animate(j):
    year = years[j-1]
    fig.suptitle('Freight value of {} in year
{}, USD million'.format(text, year), fontsize=
20, y=.95)
    dms_orig_val_animated = cat_val_df.loc[:,[
category, year]]
    dms_orig_val_animated.set_index(category,
inplace = True)
    dms_orig_val_animated['bin'] = np.digitize
(dms_orig_val_animated[year], bins) - 1

    for info, shape in zip(m.regions_info, m.r
egions):
        name = info['CFS07DDGEO']
        # some names of the regions are missta
ted in database or in shapefile
        # the purpose of if... loop is to make
this names equal
        # -----------------------------------
-------------------------------
        if name == '349':
            name = '342'
        elif name == '100':
            name = '101'
        elif name == '330':
            name = '339'
        elif name == '310':
            name = '311'
        else:
            name
        # -----------------------------------
-------------------------------
        # Alaska and Hawaii were excluded from
analysis to make map size reasonable
        if name not in ['020', '159', '151']:
            color = scheme[dms_orig_val_animat
ed.loc[name]['bin'].astype(int)]
            patches = [Polygon(np.array(shape
), True)]
            pc = PatchCollection(patches)
            pc.set_facecolor(color)
            ax.add_collection(pc)
    return chor_map,

anim = animation.FuncAnimation(fig, func = ani
mate, frames = 10,
                               repeat_delay =
2000, interval = 1000)

anim.save('freight_bal.gif', writer='imagemagi
ck')
plt.close()
Image(url='freight_bal.gif', width = 1200, hei
ght = 1000)
```

Out[10]:


Freight value of commodities input/output balance in year 2045, USD million

Thoughts and conclusions:

(1). Us production is concentrated around big cities:

- New-York (CSA population 23.7 million people)
- Los-Angeles (CSA population 18.7 million people
- Chicago (CSA population 9.9 million people)
- Dallas (CSA population 7.7 million people)
- Houston (CSA population 7 million people)

[CSA - is composed of adjacent metropolitan (MSA) and micropolitan statistical areas (μSA) in the United States and Puerto Rico that can demonstrate economic or social linkage (source: Wikipedia).]

(2). Consumption concentration is similar to production's one. I see the two main reasons behind this:

- population concentration around production facilities
- production centers consume row materials for following production.

(3). According to forecasts provided in the database Detroit and New-York CSA are heading into production deficit in following 30 years. At the same time Houston and Dallas CSAs are expected to have production surplus.

(4). Texas has many regions with higher production of commodities than nearby states. Information is provided per each small region. Brief web search results say that Texas has low tax burden and business friendly regulatory environment.

Analysis by commodities

The next section is analysis by commodities. I made analysis by top 10 commodities in terms of consumption, production, and balance between them per commodities.

I selected plotly as tool for this part as it provides very clear way to make buttons and sliders. They are very convenient for presenting multiperiod data.

In [11]:

```
# origination dataframe
dom_origin_comm_df = dom_origin_df[['sctg2','v
alue_2012',
                                    'value_2013',
'value_2014',
                                'value_2015',
'value_2020',
                                'value_2025',
'value_2030',
                                'value_2035',
'value_2040',
                                'value_2045']]
.groupby('sctg2', as_index = False).sum()

dom_origin_comm_df.columns = ['sctg2'] + years
dom_origin_comm_df.loc['total'] = dom_origin_c
omm_df.sum()
```

In [12]:

```
# destination dataframe
dom_dest_comm_df = dom_dest_df[['sctg2','value
_2012',
                                'value_2013',
'value_2014',
                                'value_2015',
'value_2020',
                                'value_2025',
'value_2030',
                                'value_2035',
'value_2040',
                                'value_2045']]
.groupby('sctg2', as_index = False).sum()

dom_dest_comm_df.columns = ['sctg2'] + years
dom_dest_comm_df.loc['total'] = dom_dest_comm_
df.sum()
```

top 10 originations/destinations in 2012 and 2045

In [13]:

```
# list of top 10 commodities by consumption (20
12 and 2045)
top_10_dest_2012 = dom_dest_comm_df[['sctg2',
'2012','2045']].sort_values('2012', axis = 0,
ascending = False)
top_10_dest_2012 = top_10_dest_2012.head(11)
top_10_dest_2012.drop('total', inplace = True)

top_10_dest_2045 = dom_dest_comm_df[['sctg2',
'2012','2045']].sort_values('2045', axis = 0,
ascending = False)
top_10_dest_2045 = top_10_dest_2045.head(11)
top_10_dest_2045.drop('total', inplace = True)
```

```
top_2012_dest = top_10_dest_2012['sctg2'].valu
es.flatten().tolist()
top_2045_dest = top_10_dest_2045['sctg2'].valu
es.flatten().tolist()
top_dest_list = [x for x in top_2012_dest or t
op_2045_dest]

top_dest_2012_2045 = dom_dest_comm_df.loc[dom_
dest_comm_df['sctg2'].isin(top_dest_list),
                                          ['sc
tg2']+years]
top_dest_2012_2045.reset_index(inplace = True,
drop = True)
top_dest_2012_2045['Commodity'] = top_dest_201
2_2045['sctg2'].apply(lambda x: com_dict[x])
```

In [14]:

```
# list of top 10 commodities by production (201
2 and 2045)
top_10_origin_2012 = dom_origin_comm_df[['sctg
2','2012','2045']].sort_values('2012', axis =
0, ascending = False)
top_10_origin_2012 = top_10_origin_2012.head(1
1)
top_10_origin_2012.drop('total', inplace = Tru
e)

top_10_origin_2045 = dom_origin_comm_df[['sctg
2','2012','2045']].sort_values('2045', axis =
0, ascending = False)
top_10_origin_2045 = top_10_origin_2045.head(1
1)
top_10_origin_2045.drop('total', inplace = Tru
e)

top_2012_origin = top_10_origin_2012['sctg2'].
values.flatten().tolist()
top_2045_origin = top_10_origin_2045['sctg2'].
values.flatten().tolist()
top_origin_list = [x for x in top_2012_origin
or top_2045_origin]

top_origin_2012_2045 = dom_origin_comm_df.loc[
dom_origin_comm_df['sctg2'].isin(top_origin_li
st),

['sctg2'] + years]
top_origin_2012_2045.reset_index(inplace = Tru
e, drop = True)
top_origin_2012_2045['Commodity'] = top_origin
_2012_2045['sctg2'].apply(lambda x: com_dict[x
])
```

In [15]:

```
# check assumption that top origin and top dest
ination commodities lists contains same items
print (len([x for x in top_origin_list and top
_dest_list]))
# As there are only 10 items in final list we c
an say that they contain same items.
# Thus, they can be used interchangeably as par
```

```
ameters of the function "line_plot".
# This will help to preserve color scheme durin
g making different plots.
```

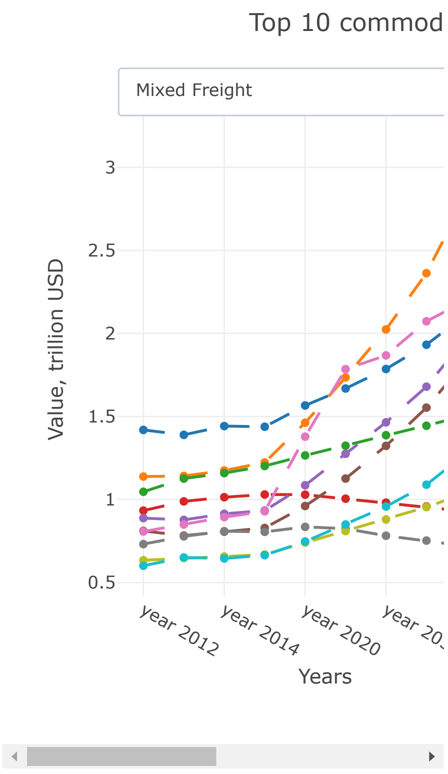
```
10
```

In [16]:

```
# function for plotting top 2012/2045 destinati
ons/originations by commodity
# plotly web page with examples was the best so
urce of information https://plot.ly/python/
def line_plot(category, sel_list, df, sel_dict
, clustering_criterion, xAxis):
    data = []
    buttons = []
    for i in sel_list:
        r = random.randint(1,256)
        g = random.randint(1,256)
        b = random.randint(1,256)
        rgb = 'rgb({}, {}, {})'.format(r, g, b
)
        trace = go.Scatter(x = ["year {}".form
at(x) for x in xAxis],
                           y = df.loc[df[categ
ory] == i, xAxis].apply(lambda x: x/1000000).v
alues.flatten(),
                           name = '{}_{}'.form
at(" ".join(sel_dict[i].split(" ")[:2]), i),
                           line = dict(width =
2,
                                       dash =
'longdash'))
        data.extend([trace])

        buttons_upd = list([dict(label = '{}'.
format(sel_dict[i]),
                                 method = 'upd
ate',
                                 args = [{'vis
ible': [x==i for x in sel_list]}])])
        buttons.extend(buttons_upd)

    # button for reset / all items
    buttons_all = list([dict(label = 'All sele
cted items',
                             method = 'upd
ate',
                             args = [{'vis
ible': [True for x in sel_list]}])])
    buttons.extend(buttons_all)

    # set menues inside the plot
    update_menus = list([dict(active=-5,
                              buttons = button
s,
                              direction = 'dow
n',
                              pad = {'r': 10,
't': 10},
                              showactive = Tru
e,
                                        x = 0.001,
                                        xanchor = 'left'
,
                                        y = 1.1,
                                        yanchor = 'top'
)])
    # Edit the layout
    layout = dict(title = '{}'.format(clusteri
ng_criterion),
                  xaxis = dict(title = 'Years'
,
                               nticks = len(xA
xis)),
                  yaxis = dict(title = 'Value,
trillion USD'),
                  updatemenus = update_menus,
                  showlegend = True)

    fig_top_10 = dict(data = data, layout = la
yout)
    iplot(fig_top_10)
```

In [17]:

```
# plot top 2012 vs 2045 destination by commodit
y
line_plot('sctg2', top_dest_list, top_origin_2
012_2045, com_dict,
          'Top 10 commodities by production',
years)
```



This chart gives interesting results that are worth further analysis.

I ignored mixed freight as it contains many other commodities that are just not significant enough to get their own group.

Two related groups - "**Machinery**" and "**Electronic and Other Electrical Equipment and Components, and Office Equipment**" are expected to drastically grow up to 2045. According to estimation provided, production of these commodities is going to be tripled (taking year 2012 as the basis). Considering automatization trend this forecast seems very reasonable. Also, it shows that Department of Transportation expects that USA will take significant share of world's new equipment production.

"**Pharmaceutical Products**" projected growth seems to be supported by increase in population and longer life span.

Growth of "**Other Prepared Foodstuffs, Fats and Oils**" may have many reasons behind it - population growth, shift to higher quality thus higher priced food, increase in export of food.

All trends from last three passages have their logic but what perplexed me is that all of them start booming right after factual period ends (2015) and forecast starts. I think these expectations are overestimated and probably gets so high because of compound annual growth rate. But this is my speculation. It would be interesting to see what forecasting methodology was applied and how it was done.

In contrast, steady increase of "**Motorized and Other Vehicles (includes parts)**" continues trend of factual periods (2012-2015). Because of this it looks much more plausible.

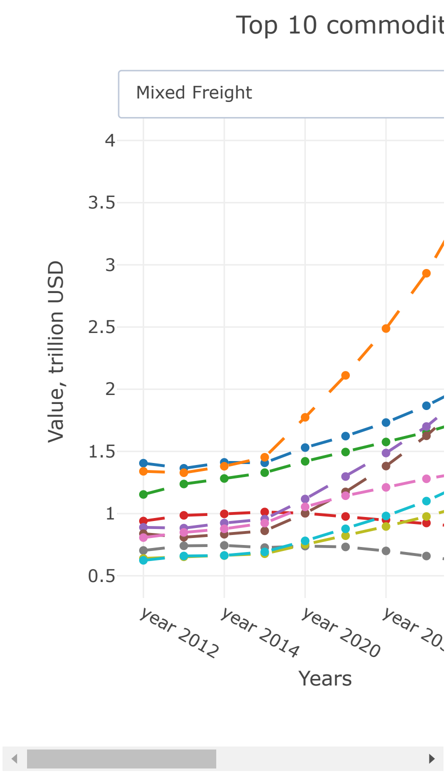
Now let's look on counterintuitive trends:

"**Gasoline, Aviation Turbine Fuel, and Ethanol (includes Kerosene, and Fuel Alcohols)**" and "**Fuel Oils (includes Diesel, Bunker C, and Biodiesel)**". Both are expected to climb up first and then drop to the level below year 2012. This is perfectly in line with anticipated domination of electric cars and probably trucks. Another big fuel consumer, planes, improves fuel efficiency with every new model.

At the same time we see that "**Other Coal and Petroleum Products, not elsewhere classified**" shows strong upward trend. Clue can be found in old version of User guide
(https://ops.fhwa.dot.gov/freight/freight_analysis/faf/faf3/u
(https://ops.fhwa.dot.gov/freight/freight_analysis/faf/faf3/u
In year 2012 this group was named "Coal and petroleum products, n.e.c. (includes Natural gas)". So the answer is quite simple - the group includes natural gas. And this growth reflects soar of fracking industry, as well as USA plans to become one of the world's top gas exporter.

In [18]:

```
# plot top 2012 vs 2045 origination by commodit
y
line_plot('sctg2', top_dest_list, top_dest_201
2_2045, com_dict,
          'Top 10 commodities by consumption',
years)
```



Consumption trends are very similar to production trends and seems to be guided by the same forecasting logic.

The visible difference are in consumption trend for "**Other Coal and Petroleum Products, not elsewhere classified**" - it is much less aggressive than production trend. It supports expectation of exporting significant part of natural gas produced.

origination / destination balance

The following chart represent comparison of domestic origination and domestic consumption for each commodity group. Imbalances are covered by export or import. I took all imbalance instances higher than 150 billion USD.

In [19]:

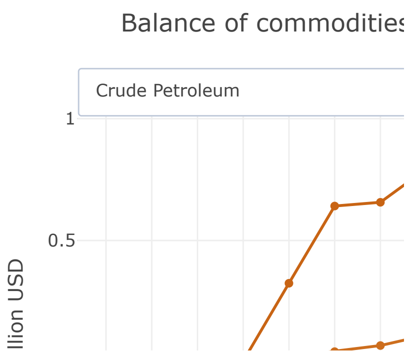
```
# plot biggest commodity deficits and surpluses
dom_dest_comm_df_2 = dom_dest_comm_df.set_inde
x(['sctg2'])
dom_origin_comm_df_2 = dom_origin_comm_df.set_
index(['sctg2'])
dom_dest_comm_df_neg = dom_dest_comm_df_2.appl
y(lambda x: x*(-1))

comm_balance_df = dom_origin_comm_df_2.add(dom
_dest_comm_df_neg, fill_value = 0.0)
comm_balance_df['max'] = comm_balance_df[years
].max(axis = 1)
comm_balance_df['min'] = comm_balance_df[years
].min(axis = 1)
comm_balance_df['abs_max'] = comm_balance_df[[
'max','min']].apply(lambda x: abs(x)).max(axis
= 1)

selected_comm_bal = comm_balance_df.loc[comm_b
alance_df['abs_max'] >= 150000, years]
selected_comm_bal.reset_index(inplace = True)

bal = selected_comm_bal['sctg2'].values.flatte
n().tolist()
com_dict[1003] = "Total for all commodities"
```

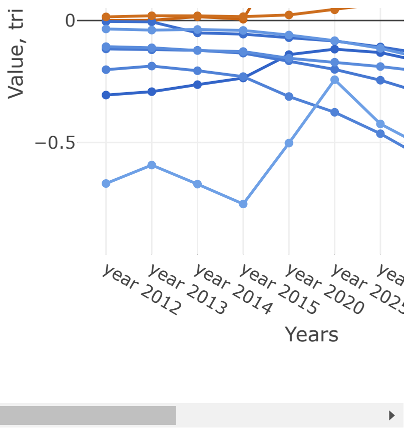
In [20]:

```
# function for plotting balance
def balance_plot(category, sel_list, df, sel_d
ict, heading):
    data = []
    buttons = []

    # rgb for surplus
    r_s = 200
    g_s = 100
    b_s = 20

    # rgb for deficite
    r_d = 50
    g_d = 100
    b_d = 200

    for i in sel_list:
        y = df.loc[df[category] == i,years].ap
ply(lambda x: x/1000000).values
        if np.sum(y)>=0:
            rgb = 'rgb({}, {}, {})'.format(r_s
, g_s, b_s)
            r_s += 5
            g_s += 10
            b_s += 10
        else:
            rgb = 'rgb({}, {}, {})'.format(r_d
, g_d, b_d)
            r_d += 10
            g_d += 10
            b_d += 5
        trace = go.Scatter(x = ["year {}".form
at(x) for x in years],
                       y = y.flatten(),
                       name = '{}_{}'.format("
".join(sel_dict[i].split(" ")[:2]), i),
                       line = dict(color = (rg
b),
                                   width = 2))


        data.append(trace)

        buttons_upd = list([dict(label = '{}'.
format(sel_dict[i]),
                                 method = 'upd
ate',
                                 args = [{'vis
ible': [x==i for x in sel_list]}])])
        buttons.extend(buttons_upd)

    buttons_all = list([dict(label = 'All exce
pt total',
                          method = 'update',
                          args = [{'visible': [
True for x in bal[:-1]]+[False]}])])

    buttons.extend(buttons_all)

    # set menues inside the plot
    update_menus = list([dict(active=-5,
                              buttons = button
s,
                              direction = 'dow
n',
                              pad = {'r': 10,
't': 10},
                              showactive = Tru
e,
                              x = 0.001,
                              xanchor = 'left'
,
                              y = 1.1,
                              yanchor = 'top'
)])

    # Edit the layout
    layout = dict(title = heading,
                  xaxis = dict(title = 'Years'
),
                  yaxis = dict(title = 'Value,
trillion USD'),
                  updatemenus = update_menus,
                  showlegend = True)

    fig = dict(data = data, layout = layout)
    iplot(fig)
```

In [21]:

```
balance_plot('sctg2', bal, selected_comm_bal,
com_dict,
             'Balance of commodities consumed
 and produced')
```





Plot above shows that most of selected commodity groups with production imbalances have and will have deficit. Only "**Other Coal and Petroleum Products, not elsewhere classified**" and "**Precision Instruments and Apparatus**" expected to have surplus.

To see total amount of deficit for all commodities you can select button "Total for all commodities". The chart shows that US production deficit is anticipated to decrease up to year 2025. Using trends by each selected commodity it becomes clear that main reason behind this is export oriented natural gas production.

Overall conclusion is that US is going to continue its trend of producing less goods than its population consume. Such imbalance can be covered by production of services and non-tangible goods (like software). Another way to cover it is to get capital inflows from other countries (purchase of corporate / government debt and investments in ownership rights).

Analysis by foreign trading partners

This part is breakdown of provided data by foreign trading partners. Most of countries are grouped by geographical regions (see User Guide or "fr_dict"). Only Canada and Mexico are presented as single countries.

In [22]:

```
# domestically originated goods exported to oth
er countries
fr_dest_df = regions_df.loc[pd.notnull(regions
_df['fr_dest'])]
fr_dest_list = fr_dest_df['fr_dest'].unique().
flatten().tolist()
fr_dest_df = fr_dest_df[['fr_dest','value_201
2',
                           'value_2013', 'valu
e_2014',
                           'value_2015', 'valu
e_2020',
                           'value_2025', 'valu
e_2030',
                           'value_2035', 'valu
e_2040',
                           'value_2045']].grou
pby('fr_dest', as_index = True).sum()

fr_dest_df.columns = years
```

In [23]:

```
# goods imported from other countries
fr_orig_df = regions_df.loc[pd.notnull(regions
_df['fr_orig'])]
fr_orig_list = fr_orig_df['fr_orig'].unique().
flatten().tolist()
fr_orig_df = fr_orig_df[['fr_orig','value_201
2',
                           'value_2013', 'valu
e_2014',
                           'value_2015', 'valu
e_2020',
                           'value_2025', 'valu
e_2030',
                           'value_2035', 'valu
e_2040',
                           'value_2045']].grou
pby('fr_orig', as_index = True).sum()

fr_orig_df.columns = years
```

In [24]:

```
# fr balance df
fr_balance_df = fr_dest_df.add(fr_orig_df.appl
y(lambda x: x*(-1)))
fr_balance_df.loc['total'] = fr_balance_df.sum
()
fr_balance_df.reset_index(inplace = True)
```

In [25]:

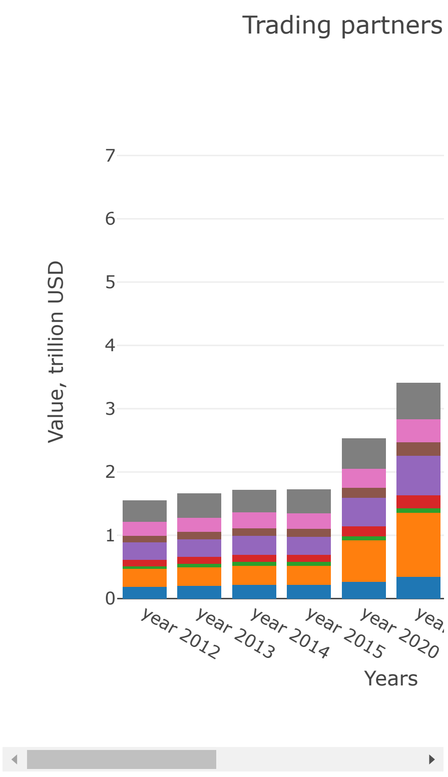
```
# reset index for fr dataframes
fr_dest_df.reset_index(inplace = True)
fr_orig_df.reset_index(inplace = True)
```

In [26]:

```
def fr_dest_orig_plot(category, list_fr, df, c
lustering_criterion):
    data = []
    for i in list_fr:

        trace = go.Bar(x = ["year {}".format(x
) for x in years],
                       y = df.loc[df[category]
== i,years].apply(lambda x: x/1000000).values.
flatten(),
                       name = '{}'.format(fr_d
ict[i]))

        data.append(trace)

    # Edit the layout
    layout = dict(title = 'Trading partners gr
ouped by {}'.format(clustering_criterion),
                  xaxis = dict(title = 'Years'
),
                  yaxis = dict(title = 'Value,
trillion USD'),
                  showlegend = True,
                  barmode='stack')

    fig = dict(data = data, layout = layout)
    iplot(fig, filename='stacked-bar')
```

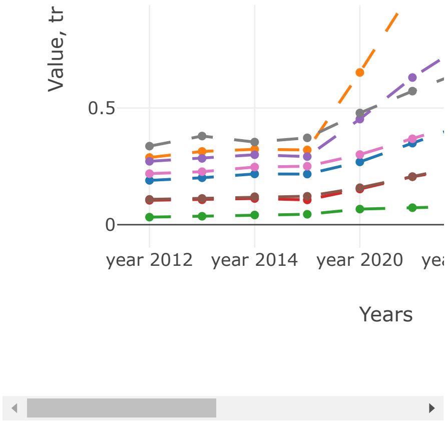
In [27]:

```
# plot overall change and structure of trading
 partners by export
fr_dest_orig_plot('fr_dest', fr_dest_list, fr_
dest_df, 'export')
```



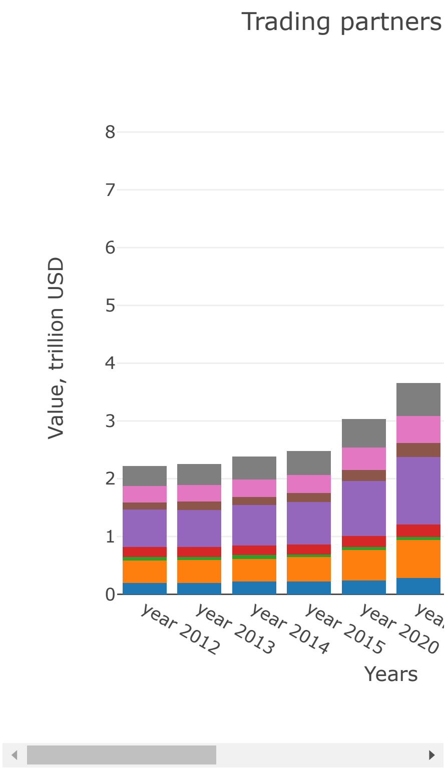
In [28]:

```
# plot trading partners by export
line_plot('fr_dest', fr_dest_list, fr_dest_df,
fr_dict,
          'Trading partners by export', years)
```



In [29]:

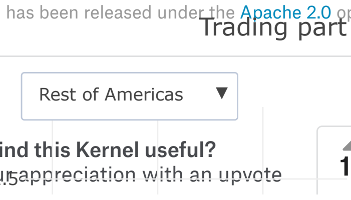
```
# plot overall change and structure of trading
 partners by export
fr_dest_orig_plot('fr_orig', fr_dest_list, fr_
orig_df, 'import') # fr_dest_list was applied
 to match colors with previous plots
```



In [30]:

```
# plot trading partners by export
line_plot('fr_orig', fr_dest_list, fr_orig_df,
fr_dict,
          'Trading partners by import', years)
# fr_dest_list was applied to match colors with
previous plots
```

This kernel has been released under the Apache 2.0 open source license.



Did you find this Kernel useful?
Show your appreciation with an upvote
17

Data

Data Sources

- Freight Analysis Fram...
  - FA... 42 columns
  - FA... 42 columns
  - FAF4 User Guide.pdf
  - CFS_AREA_shapefile.

Freight Analysis Framework

Flows of goods among US regions for all modes of transportation

Last Updated: 2 years ago (Version 1)

About this Dataset

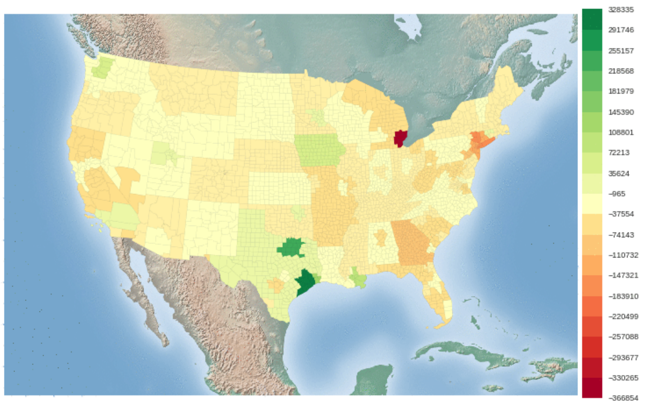
The Freight Analysis Framework (FAF) integrates data from a variety of sources to create a comprehensive picture of freight movement among states and major metropolitan areas by all modes of transportation. Starting with data from the 2012 Commodity Flow Survey (CFS) and international trade data from the Census Bureau, FAF incorporates data from agriculture, extraction, utility, construction, service, and other sectors. FAF version 4 (FAF4) provides estimates for tonnage (in thousand tons) and value (in million dollars) by regions of origin and destination, commodity type, and mode. Data are available for the base year of 2012, the recent years of 2013 - 2015, and forecasts from 2020 through 2045 in 5-year intervals.

Inspiration

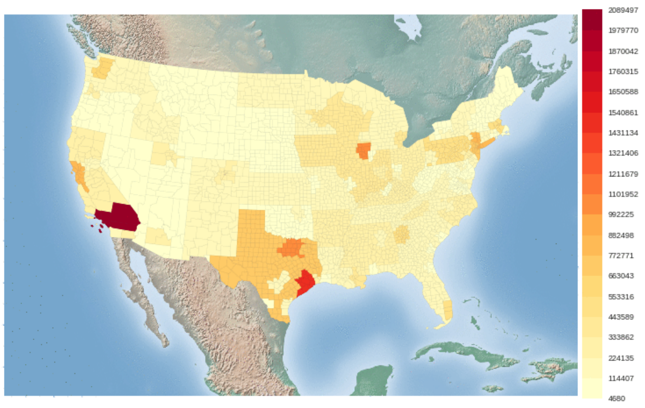
This dataset should be great for map-based visualizations.

Output Visualizations

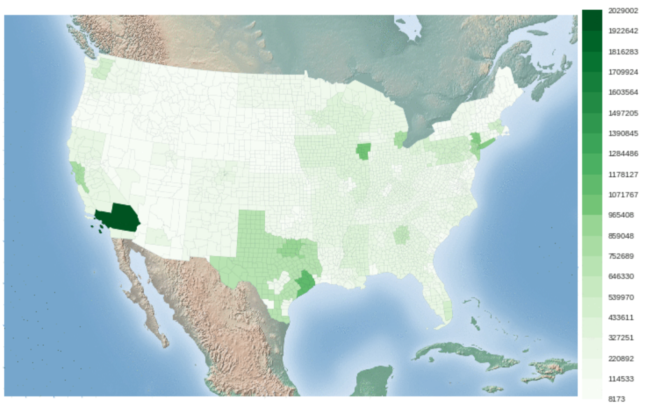
Freight value of commodities input/output balance in year 2045, USD million



Freight value of commodities with domestic origin in year 2045, USD million



Freight value of commodities with domestic destinaton in year 2045, USD million



Run Info

Succeeded	True	Run Time	2669.4 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	kaggle/python(Dockerfile)	Output Size	0
Timeout Exceeded	False	Used All Space	False
Failure Message			

Log

Download Log

```
Time  Line #  Log Message
           1  [{
           2    "data": "[NbConvertApp] Converting notebook
__temp_notebook_source__.ipynb to html\n",
  "stream_name": "stderr",
  "time": 1.7850264600128867
},{
  "data": "[NbConvertApp] Writing 498642 bytes to
__results__.html\n",
  "stream_name": "stderr",
  "time": 2.1127787900040857
}{
  "data": "[NbConvertApp] Converting notebook
__temp_notebook_source__.ipynb to notebook\n",
  "stream_name": "stderr",
  "time": 1.8287799659883603
},{
  "data": "[NbConvertApp] Executing notebook with kernel:
python3\n",
  "stream_name": "stderr",
  "time": 1.85528708199854
},{
  "data": "Fontconfig warning: ignoring C.UTF-8: not a valid
language tag\n",
  "stream_name": "stderr",
  "time": 3.5400210410007276
},{
  "data": "[NbConvertApp] Writing 330855 bytes to
__notebook__.ipynb\n",
  "stream_name": "stderr",
  "time": 1102.3322400680045
}{
  "data": "[NbConvertApp] Converting notebook __notebook__.ipynb to
html\n",
  "stream_name": "stderr",
  "time": 8.885869553982047
},{
  "data": "[NbConvertApp] Writing 498639 bytes to
__results__.html\n",
  "stream_name": "stderr",
  "time": 9.365631280990783
}

Complete. Exited with code 0.
```

Comments (4)

Sort by
All Comments Hotness

Click here to enter a comment...

DSEverything • Posted on Latest Version • 2 years ago • Options • Reply 1

many good insights. Love the charts with user control and timely information.

ievgen Kernel Author • Posted on Latest Version • 2 years ago • Options • Reply 0

Thanks, DSEverything. Plotly is such a great tool for visualizations.

dwyerjonatha... • Posted on Latest Version • a year ago • Options • Reply 0

Does the FAF use bill of lading (BOL) data as the source for shipping records?

ievgen Kernel Author • Posted on Latest Version • a year ago • Options • Reply 0

In this case data source can't be described as one document. It is quite complicated. Details can be found in the following document: http://faf.ornl.gov/fafweb/data/FAF4%20draft%20report%20on%20data%20and%20method.pdf

It is description of data sources and estimation methodologies for freight analysis framework. Hope it will help.

Similar Kernels

© 2019 Kaggle Inc

Our Team Terms Privacy Contact/Support