Introduction to Bokeh:

visualizing data from the USDA Branded Food Products Database

by Espand Jaraha

What is Bokeh

a python package for creating **interactive visualizations and applications**.

What Makes it Special

BokehJS

- JavaScript library that runs in the client's browser
- Generates JavaScript in response to Bokeh commands (Python)

Shareable

- Bokeh Server, Jupyter, html, json

Interactive

- allows users to interact with the visualization using tools

  ex. drop down menu, slider, check boxes

How it's Structured

Bokeh is organized into two **interfaces**:

- **bokeh.plotting**
  - for composing visual glyphs**
  - use to define template
- **bokeh.models**
  - for creating interactive visualizations
  - use to add tools to the template

***glyph*: a marker on a plot (circle, line, triangle) which represents data

This Tutorial

With this tutorial, you can create an **interactive visualization** using data from the **USDA Branded Food Products Database**. Food products from three manufacturers are plotted on a **scatter plot** according to their nutrient rich foods index (NFRI) and energy content per serving size. A **CategoricalColorMapper** is used to color points by manufacturer. A **HoverTool** shows additional product information. A **Select widget** allows the user to control the data displayed on the plot. **Bokeh Server** is used to output the visualization.

Input: data from USDA Branded Food Product Database, with calculated NFRI

- three manufacturers: Wal-Mart, Meijer, Inc., and Target

Output: interactive visualization using Bokeh Server

Let's Get Started

1. **Open** *viz.py*

   notice that most of the script is commented out we will be uncommenting as we move through the tutorial

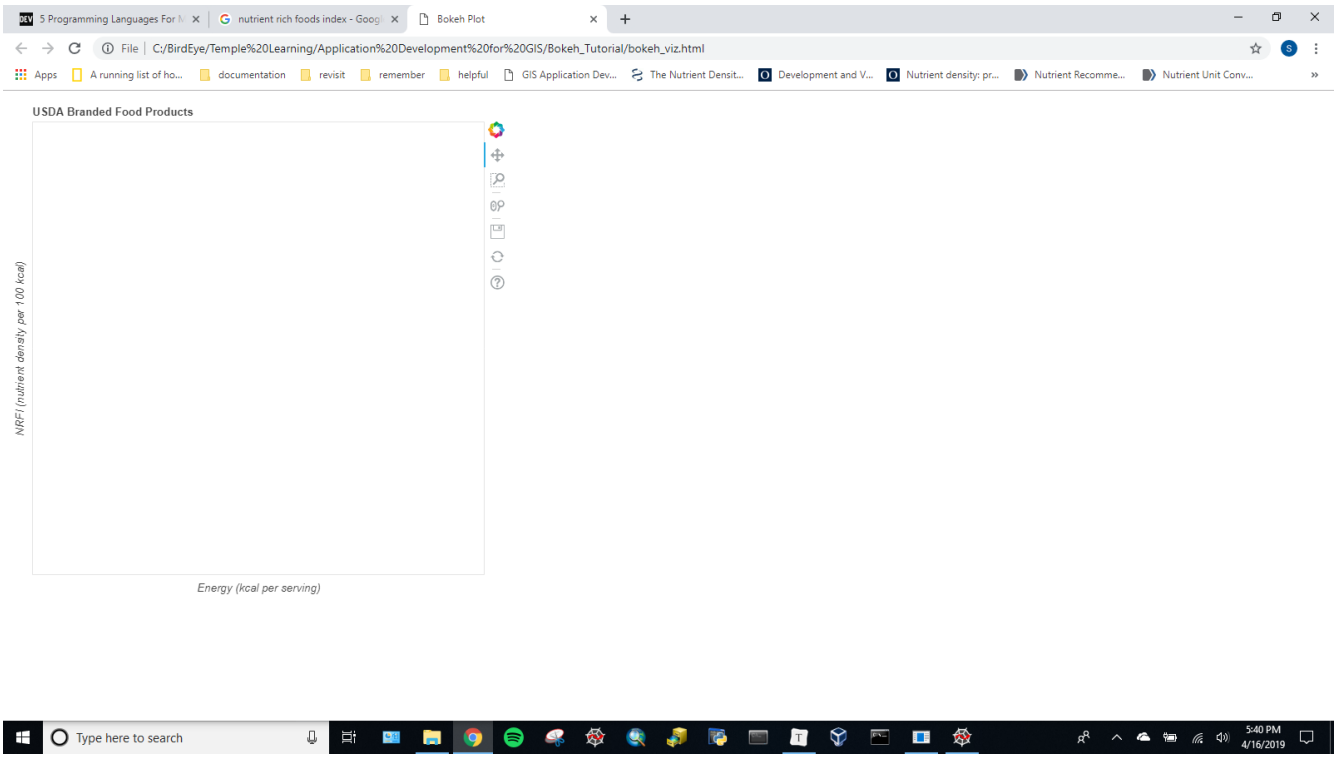
2. **Run** *viz.py*

```
from bokeh.plotting import figure, ColumnDataSource, curdoc, Row, show, output_file
from bokeh.models import HoverTool, CategoricalColorMapper, Select
import pandas as pd
import os

# change cwd to the location of this script
dir_script =  os.path.dirname(os.path.abspath("__file__"))
os.chdir(dir_script)

# initialize figure
viz = figure(x_axis_label = "Energy (kcal per serving)"\
             ,y_axis_label = "NRFI (nutrient density per 100 kcal)"\
             ,title="USDA Branded Food Products")

# import data
path = os.path.join(dir_script,"data.csv")
df = pd.read_csv(path, dtype=object)

# convert data types
float_cols = ["nrfi","serving_size", "household_serving_size", "energy_per_serving"]
df[float_cols] = df[float_cols].astype("float32")

# generate html file
output_file("output_file.html")

# show visualization in browser
show(viz)
```

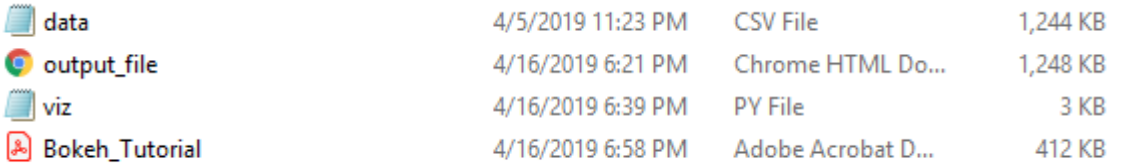
3. **Notice** the html file called *output_file.html* in the Bokeh_Tutorial folder

- *what's happening??*
    - import packages
    - set current working directory
    - **figure()**
        - initializes the plot
            - assigned to variable "viz"
        - parameters x axis label, y axis label, and title set the axis labels and title of the figure
    - **pd.read_csv()**
        - imports data as a dataframe
            - assigned to variable "df"
    - convert data types
    - **output_file()**
        - outputs an html file of the figure in the Bokeh_Tutorial folder
            - called "output_file.html"
    - **show()**
        - opens a browser to display the figure

Output of show():



Output of output_file():



4. **Set Data Source**

    1. in viz.py, uncomment the code shown below
    2. run viz.py

```
# set data source
data_source = ColumnDataSource(df)
```

- *what's happening??*
    - **ColumnDataSource()**
        - assigned to variable "data_source"
        - initializes data source
        - does not change the appearance of the figure

5. **Add glyph**
    1. in viz.py, uncomment the code shown below
    2. run viz.py

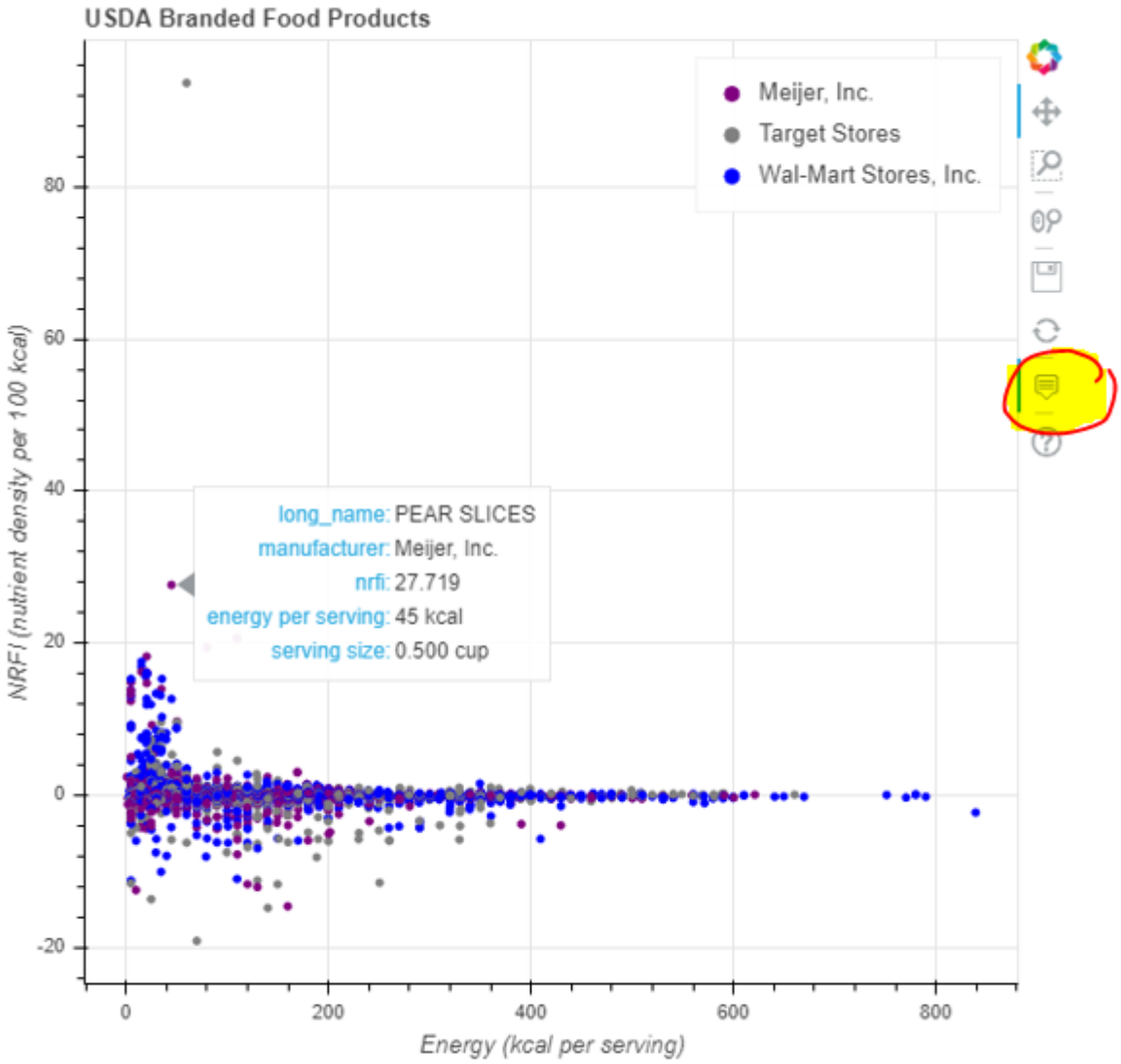
```
# color by manufacturer
mapper = CategoricalColorMapper(factors = ["Wal-Mart Stores, Inc.","Meijer, Inc."\
                                           ,"Target Stores"]\
                                ,palette=["blue","purple","gray"])

# add glyph
viz.circle("energy_per_serving","nrfi", source=data_source,\
           legend="manufacturer", color=dict(field='manufacturer',transform=mapper))
```

- *what's happening??*
    - **viz.circle()**
        - adds circle glyph to the figure (viz)
            - parameters 1 & 2
                - fields from df to be used for x and y axes
            - source parameter
                - specifies the data for the glyph (defined in step 4)
                - multiple glyphs can be added to a plot with different data sources
            - legend parameter
                - specifies field to be used in the legend
            - color parameter
                - defines the color of the glyph
                - syntax: dict(field='*field-to-be-color-mapped*', transform=*how-to-map-field*)
    - **CategoricalColorMapper()**
        - assigned to variable "mapper"
        - factor parameter
            - values to be color-mapped
            - must be values in the field of viz.circle() 's color parameter
        - palette parameter
            - defines colors for factors
            - order matters

6. **Define and Add HoverTool**
    1. in viz.py, uncomment the code shown below
    2. run viz.py
    3. notice a new tool in the sidebar
    4. hover over a glyph

```
# define hover tool
hover = HoverTool(tooltips=[("long_name","@long_name")\
                            ,("manufacturer","@manufacturer")\
                            ,("nrfi", "@nrfi")\
                            ,("energy per serving","@energy_per_serving kcal")\
                            ,("serving size","@household_serving_size
@household_serving_size_uom")])

# add hover tool
viz.add_tools(hover)
```

- *what's happening??*
    - **HoverTool()**
        - defines HoverTool
            - tooltips parameter
                - specifies labels and data for HoverTool
                    - syntax: tooltips = [(*"label-to-be-displayed"*, *"@field-name-for-data-to-be-displayed optional-additional-text"*)]
                        - @ symbol means show data from field
                        - $ symbol can be used to show data from cursor position ($x or $y)
    - **viz.add_tools(hover)**
        - adds HoverTool to visualization
        - without this step, the hover tool would be defined, but would not appear on the visualization


USDA Branded Food Products
Meijer, Inc.
Target Stores
Wal-Mart Stores, Inc.
long_name: PEAR SLICES
manufacturer: Meijer, Inc.
nrfi: 27.719
energy per serving: 45 kcal
serving size: 0.500 cup
NRFI (nutrient density per 100 kcal)
80
60
40
20
0
-20
0
200
400
600
800
Energy (kcal per serving)

7. **Define Select Widget and Layout**

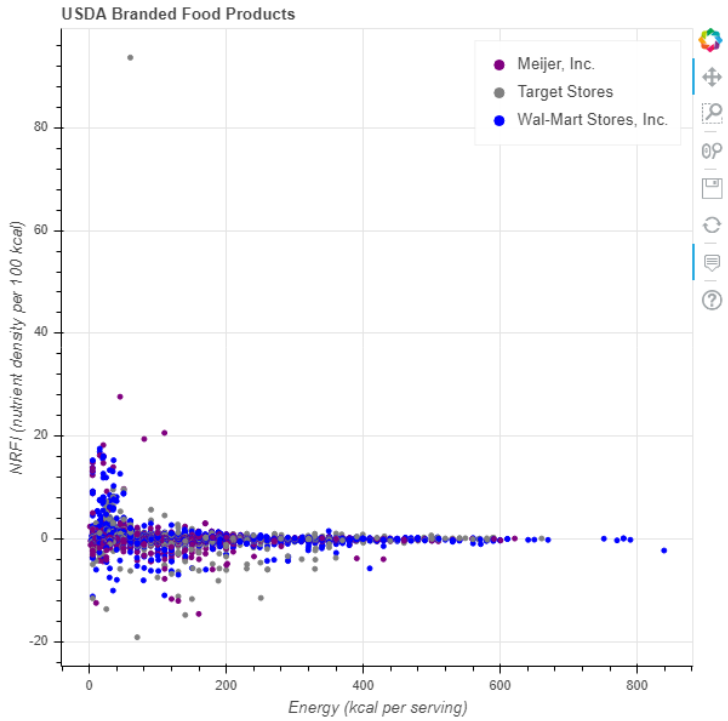
    1. in viz.py, uncomment the code shown below
    2. *!! at the bottom of viz.py, change show(viz) to show(layout)*
    3. run viz.py
    4. change the selection in the select tool

```
# define Select widget
manufacturers = list(pd.unique(df.manufacturer))
manufacturers.append("All")
select_widget = Select(title="Manufacturer", options=manufacturers, value="All")

# define layout
layout = Row(select_widget,viz)
```

- *what's happening??*
    - **manufacturers**
        1. **list(pd.unique(df.manufacturer))**
            - create a list of unique values in the manufacturer column in the df dataframe
            - these will be the options for the Select widget
        2. **manufacturers.append("All")**
            - add the string "All" to the list of unique manufacturers
            - this option will display data from all three manufacturers
    - **select_widget = Select(title="Manufacturer", options=manufacturers, value="All")**
        - assigned to variable "select_widget"
        - defines the Select widget
            - syntax: Select(title=*"title-for-select-tool"*, options=*select-tool-options*, value=*"default-option-for-select-tool"*)
    - **Row()**
        - assigned to the variable "layout"
        - defines layout of application (widgets and figures should go)
            - select_widget will be on the left
            - the figure (viz) will be on the right
        - can use **Column()** instead to arrange objects vertically

Manufacturer

All



8. **Define the Callback**
    1. in viz.py, uncomment the code shown below
    2. run viz.py
    3. change the selection in the select tool

```
# define callback for select_widget
def update():
    if select_widget.value != "All":
        df_select = df[df.manufacturer == select_widget.value]
        source.data.update(df_select)
    else:
        source.data.update(df)

# call update() function when select_widget value changes
select_widget.on_change('value', lambda attr,old,new : update())

# update layout
curdoc().add_root(layout
```

- *what's happening??*
    - **def update()**
        - defines the function that is called when a user changes the selection in the select tool
            1. *if* the value attribute of select_widget is not equal to "All" (in other words, if only data from a specific manufacturer should be displayed) *then...*
                - df.manufacturer == select_widget.value
                    - create a boolean series that is True when the manufacturer field of df is equal to the value of select_widget
                - df_select = df[df.manufacturer == select_widget.value]
                    - use the boolean series to to index df
                    - save the indexed dataframe as "df_select"
                    - df_select contains only the rows from df where the manufacturer matches the value of select_widget
                - source.data.update(df_select)
                    - update the data attribute of source
            2. *else*
                - source.data.update(df)
                    - update the data attribute of source
    - **select_widget.on_change('value', lambda attr,old,new : update())**
        - this line of code makes the visualization interactive!! without it, the data would not update
        - **.on_change()** method specifies what should happen when the value attribute of select_widget changes
            - first parameter ('value')
                - the attribute of select_widget which will be changing
            - second parameter (lambda attr,old,new : update())
                - what should happen when the value attribute of select_widget changes
                    - lambda function with three parameters (attr, old, new) which calls the update() function
    - **curdoc().add_root(layout)**
        - updates the current document to match the new layout (with changes from select_widget)

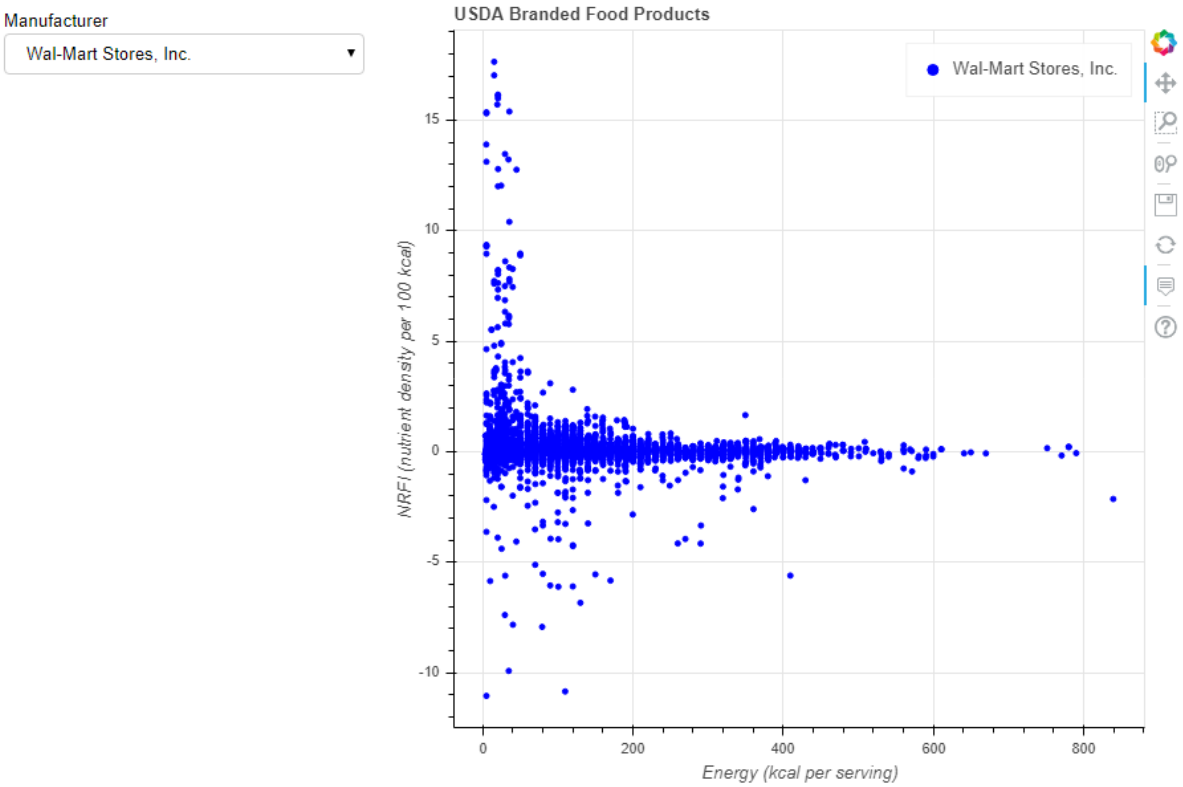
9. **Connect to Bokeh Server**

    1. save viz.py
    2. open the Anaconda Prompt
    3. navigate to the Bokeh_Tutorial directory (where the script is stored)
    4. run the line of code shown below
    5. change the selection in the select tool

```
bokeh serve --show viz.py
```

Congratulations!

You've created an interactive visualization using Bokeh server!



About The Data

The data in this tutorial comes from the USDA Branded Food Products Database.

- read more about USDA BFPD:
    - https://ndb.nal.usda.gov/ndb/docs/BFPDB_Doc.pdf

The nutrient rich foods index (NRFI) was calculated for each food product in the BFPD.

- read more about NRFI:
    - https://academic.oup.com/jn/article/139/8/1549/4670510
    - https://academic.oup.com/ajcn/article/99/5/1223S/4577490
- NIH data used to calculate NRFI:
    - https://ods.od.nih.gov/Health_Information/Dietary_Reference_Intakes.aspx
    - https://www.dsld.nlm.nih.gov/dsld/unitconversion.jsp

Further Reading on Bokeh

- *Key concepts:* https://bokeh.pydata.org/en/latest/docs/user_guide/concepts.html
- *More on Bokeh server:* https://bokeh.pydata.org/en/latest/docs/user_guide/server.html

- *Bokeh application gallery:* https://bokeh.pydata.org/en/latest/docs/gallery.html