What are the prices of nutritionally-eligible food products that are and are not WIC-approved in a given State?

Project focus: differences in the pricing of the 100% whole wheat bread

```
- by whether it is WIC-approved or not WIC-approved
- by a type of a store
- by a source of price
```

Notebook Setup

In [ ]:
```
# pandas-related imports
import pandas as pd

# database interaction imports
from pyathenajdbc import connect

# visualization
import matplotlib as plt
%matplotlib inline

# show full cell content
pd.set_option('display.max_colwidth', -1)
```

In [ ]:
```
# connection to the database
conn = connect(s3_staging_dir = 's3://usda-iri-2019-queryresults/',
               region_name = 'us-gov-west-1',
               LogLevel = '0',
               workgroup = 'workgroup-iri_usda')
```

Get prices of 100% whole wheat bread in 2017

We are interested in the shelf price of 100% whole wheat bread (`dollarspaid` variable in the `trip_all` table).

```
In [ ]: qry = '''
        select dollarspaid, channelid, storename, purdate, deal, price_source, p
        roduct.upcdesc, quantity, category
        from iri_usda.trip_all trip
        join iri_usda.pd_master_all product
        on trip.upc = product.upc
        join iri_usda_2019_db.project_q2_cohort demo
        on demo.panid = trip.panid
        where trip.year = '2017' AND upcdesc like '%100% WHOLE WHEAT%' AND categ
        ory like '%BREAD%'
        '''

        prices = pd.read_sql(qry, conn)
```

```
In [ ]: prices.head()
```

WIC-approved and non-WIC approved

We will take a look at WIC-approved brands of 100% whole-wheat breads in Indiana.

```
In [ ]: # The WIC-approved product lists from 2019 are available for some states
        in the `wic_apl_STATE` table:
        qry = '''
        select category_description, subcategory_description, brand, food_descri
        ption, state
        from iri_usda.wic_apl_in
        '''

        wic_list = pd.read_sql(qry, conn)
```

```
In [ ]: wic_list.head()
```

We filter the `subcategory_description` by the `100% Whole Wheat` string.

```
In [ ]: wic_brands = wic_list[wic_list['subcategory_description'].str.contains(
        'Bread - 100% Whole Wheat')]
```

```
In [ ]: wic_brands.head()
```

Let's get all unique WIC-approved brand names.

```
In [ ]: wic_brands['brand'].unique()
```

Now let's check unique brands in our purchase data.

```
In [ ]: # Get first 30 rows of unique product descriptions from the purchase dat
        a
        prices['upcdesc'].unique()[:30]
```

Check if a specific brand is in the WIC-approved brands list in Indiana.

The standard practice when comparing strings is to lowercase them (to account for differences in uppercase or lowercase usage). We use here `.lower()` function.

```
In [ ]: wic_brands['brand'].str.lower().unique()
```

Now we can use `in` to check if something is in the list. It will return a `boolean` value: `True` or `False`. Let's check, for example, the brand that we know is in the list. It should return `True`.

```
In [ ]: INSERT BREAD in wic_brands['brand'].str.lower().unique()
```

If a bread is not in the list, we can use it as an example of a brand which is not on a WIC-approved list in Indiana.

Let's compare the prices for a non-WIC approved brand and a WIC-approved brand.

We can explore the summary statistics for both brands by using `describe` function (`dollarspaid` variable contains the price - see IRI Methodology report for how IRI assigns prices). Explore the dataset, find out the outliers (max or min), investigate the mean and percentiles.

```
In [ ]: #non-WIC-approved
        nonWIC_prices = prices[prices['upcdesc'].str.contains(INSERT NON-APPROVE
        D BREAD)]
        nonWIC_prices.describe()
```

```
In [ ]: # WIC-approved
        WIC_prices = prices[prices['upcdesc'].str.contains(INSERT APPROVED BREAD
        )]
        WIC_prices.describe()
```

Prices by a type of store

We can also compare prices by a type of store.

For example, if we know that if a specific brand is WIC-approved in a given state, we can filter by all purchases of this brand.

```
In [ ]: prices_filtered = prices[prices['upcdesc'].str.contains(INSERT WIC-APPRO
        VED BREAD)]
```

Let's see how many unique kinds of bread product this brand has.

```
In [ ]: prices_filtered['upcdesc'].unique()
```

As this brand has different types of bread product (16 oz, 20 oz, 40 oz), we will find the most popular type to focus on and compare prices across different types of store.

```
In [ ]: # Find the most popular bread product
        prices_filtered.groupby('upcdesc').size().reset_index().sort_values(0,as
        cending=False).head(1)
```

Now we will compare the price of this specific product across different types of stores.

```
In [ ]: popular_product = prices_filtered.groupby('upcdesc').size().reset_index
        ().sort_values(0,ascending=False).head(1)['upcdesc'].values
        popular_product_prices = prices_filtered[prices_filtered['upcdesc'].isin
        (popular_product)]
```

What is the mean price of this product in different types of stores?

```
In [ ]: store_desc = pd.DataFrame()
        store_desc['channelid'] = [1,2,3,4,5,6,7,8]
        store_desc['description'] = ['Grocery', 'Drug', 'Mass', 'Supercenter',
        'Convenience', 'Dollar', 'Club', 'All other']
```

```
In [ ]: store_desc.merge(popular_product_prices.groupby('channelid')['dollarspai
        d'].mean(),on='channelid').sort_values('dollarspaid',ascending=False)
```

Differences in prices by a source of price

The `price_source` variable indicates whether the prices was assigned through point-of-sale data, entered by the household, or assigned from the price dictionary. Please refer to the IRI report for more description of this variable.

We can compare differences in prices based on the source of the price of the product.

Let's check how many unique sources of the price we have for the same bread product as above.

```
In [ ]: popular_product_prices['price_source'].unique()
```

Let's get the descriptions of those codes and find the difference in prices based on the source of the price. We can look at the mean or median.

```
In [ ]: prices_desc = pd.DataFrame()
        prices_desc['price_source'] = [1,2,3]
        prices_desc['description'] = ['Panelist Input', 'Point Of Sale', 'Dictio
        nary']
```

```
In [ ]: # Getting the mean
        prices_desc.merge(popular_product_prices.groupby('price_source')['dollar
        spaid'].mean().reset_index(), on='price_source')
```

```
In [ ]: # Getting the median
        prices_desc.merge(popular_product_prices.groupby('price_source')['dollar
        spaid'].median().reset_index(), on='price_source')
```