**Cheese GLORIOUIS cheese**

# Readme for Data Vis Project

# About the author

Heidi Beezub

First Year Grad Student

Mercyhurst University

DATA 550 Data Visualization

Project 3 Weekly Dairy Product Prices (AKA the Cheese dataset)

12/12/17

This is a data cleaning and data visualization project.

#Why should you use this code or be interested in it?

If you want some ways to clean data (especially dates) you will be interested in some of the codes I used to work with this data. I had to do quite a bit of manipulation to break out the dates.

Beyond that who would not want to know about 500-pound wheels of cheese? (seriously). This is a great multiple data set that includes not only the 500-pound but 40 pound block of cheese pricing as well as data sets on butter, dry whey and non-fat dry milk prices. These data sets are totally underused (probably due to the date issue -which I will explain).

#This repository contains:

1. The data set: Datamart-Export\_DY\_WK100-500 Pound Barrel Cheddar Cheese Prices, Sales, and Moisture Content\_20170829\_122601.csv. available on Kaggle, search for ‘Weekly Dairy Product Prices’ to find the most current copy of the data set as well as the other ‘milk’ datasets I mentioned.

2. My Jupiter notebook.

3. This Readme

4. My ‘ugly’ files: My final file “\_5” is ‘clean & pretty.’ My working files include the commented out lines of code that didn’t work. It is important to keep track of what doesn’t work. Over several days/weeks of working with a data set you’ll forget what you’ve tried.

## Table of Contents

- [Background](#background)

- [Install](#install)

- [Usage](#usage)

- [Related Efforts](#related-efforts)

- [Project Explanation](#Project-explanation)

- [License](#license)

## Background

1. The dataset consists of an csv (excel) file. There are 1411 rows of data and 7 columns.

The columns:

* 1. Week Ending Date: This is the actual date of the price listed
  2. Report Date: This is the date the data was entered into the database.
  3. Date: full disclosure: I am assuming that this is the ‘age’ of the cheese. Unfortunately, I could not find a specific reference even on the USDA website and the source data/documents for database. There are 5 dates which range from the same as the week ending date to 1, 2, 3 & 4 weeks out (0,7,14,21,28 days). This was part of the data cleaning. These dates are listed as the day and month (no year) so in order to make sense of the data, these needed to be actual dates.
  4. Weighted Price : basically the price per pound
  5. Weighted Price adjusted to 38% moisture: again, the price (adjusted for moisture content)
  6. Sales: Number of units sold. Due to the large numbers in the sales for the 500 & 40 pound block cheese data sets, this is the number of pounds (not the number of 500 pound barrels of cheese). And no, a 500 pound (or 40 pound) wheel of chees doesn’t weigh exactly 500 (or 40) pounds,
  7. Moisture Content: percent of moisture content.

The rows:

1. The rows are groups of 5. There are 5 of the same ‘week ending’ with corresponding ‘report date.’ In each set there is a date (again the age) that corresponds to the same week ending date & then 7, 14, 21 & 28 days prior to the week ending date.
2. Problems with the data set:
   1. The biggest issue is the ‘Date’ column being only a day and month (for example) ’22-Jul.’ This will not read as a date. Since the price for each of these rows is different the graph produces a strange ‘fuzzy’ line if you try to graph the data.
   2. The 2016 dataset has some issues with the ‘age’ dates not being standard. Instead of being in 7 day increments there are multiple values. I can only guess that the data was entered incorrectly.
   3. There are 15 rows of data from 2013 that have zero (probably null) for the weighted price, sales & moisture content.
3. Data cleaning: My main goal was to get the ‘Date’ field with a valid date format so I could separate each line of data. This was not all easy.
   1. I could have simply modified these in the downloaded csv (excel file) in literally 3 seconds (I know my excel). However, that would not have given me the experience in cleaning the data, in addition what would I do when I had a database that I couldn’t download?
   2. I tried various methods to read the ‘Date’ using various strftime formatting. How I finally was able to get the date formatted is VERY inelegant. If you are reading this, it means you are going to work with this great data set, I hope you have a more elegant solution to the problem.
      1. First, I added a year (1999) which isn’t in the data so I would have a date format. I saved this as a new column in my data frame.
      2. Next, I extracted the year from the week ending date. (I couldn’t get this to add directly into the date column which is why I needed step i). This too I saved as a new column in the data frame.
      3. I had to break apart the day month into separate columns.
      4. Finally, I took all the pieces (year, month & day) and put them all back together in the column I created for the 1999 dates.
      5. I deleted the ‘day’ & month columns but kept the year column as this would be useful.
   3. I found a very helpful code ‘.dtypes’ (used dataframname.dtypes). This returns the variable type for each column. This helped me know that I needed to convert some object data into dates.
   4. Once I had valid dates in the Week ending & my new “Date” field, I was able to subtract to obtain the age.
   5. I also had to account for the age dates that were now negative. Since I used the week ending year each January the December week dates would show as the current year instead of the prior year. For example January 5, 2017 week ending dates the December prior week dates now used 2017 instead of 2016. I simply added 365 back to the number to get rid of the negatives. Again, I hope someone else has a more elegant solution to get the ‘age’ of the cheese.
4. Now that I had useable dates I could break my data into some smaller chunks & start graphing.

The goals for this repository are:

Clean the data set & break out the age dates so I could graph ‘single’ lines & not a line based on the 5 values in the week ending date.

## Install

This project uses:

Python & the following python packages

pandas

seaborn

matplotlib

datetime

This module depends upon a knowledge of Pandas and matplotlib.

## Usage

Build on what I’ve done! Even though my cleaning isn’t elegant, it gets the job done. I’d love to see more done with this cheese data.

## Related Efforts

Up until a month ago this data had no kernels (Kinda bummed I didn’t get to this sooner). The visualization that is there has a “fuzzy” graph as it is plotting all 5 of the pricing parameters.

## Project Explanation

Explanation of Jupiter notebook code:

Ok I went into some of the description of what I did to clean the data. Now we’ll take a look line by line:

Line 1: Import python package Pandas (a Python package for working with data), seaborn (a Python graphing library) and Python matplotlib (a Python plotting library), datetime (used to format dates). The Seaborn package generates warnings that we don't want to see, so there is a "warning" code to suppress them.

Line 2: Imports the excel data as a csv file. It is saved into the variable ‘data.’ For working on my computer, the code is:

data = pd.read\_csv('Datamart-Export\_DY\_WK100-500 Pound Barrel Cheddar Cheese Prices, Sales, and Moisture Content\_20170829\_122601.csv').

But once uploaded to Kaggle it needs to be:

data = pd.read\_csv('../input/ Datamart-Export\_DY\_WK100-500 Pound Barrel Cheddar Cheese Prices, Sales, and Moisture Content\_20170829\_122601.csv')

Line 3: The next few line are ‘standard’ ways to heck the data & take a quick look at it. First the ‘.shape’; the number of rows & columns.

Line 4: ".head" command shows the first few rows of the data. The 10 specifies the number of rows. If the () is blank, the default is 5 rows.

Line 5: Verify that we are working with a data frame. We ‘know’ we have a data frame, type(data) verifies that the data was read in as a data frame .

line 6: this is a new code I found ‘.dtypes.’ Returns the type of each data type in the dataframe. This tells me the 3 ‘date” fields aren’t dates & that the sales field in not an integer (or float).

Line 7: See if we have any missing data. We don’t have any missing data. We are asking if there are any null values. If we had any null values we would get ‘True” but since everything is ‘False’ we don’t have any null values. However, we do know there are a handful of zero data (which are probably null data).

Line 8: Use data.describe to see an overview of the data means, median, max, min, etc. for each column. This gives us information on the columns with data, but the columns with word descriptors (the 3 date columns & the sales column) are not included. This is an interesting feature that gives us some statistical information about our data. We see the that there is zero listed as a minimum (from the zeros in the data).

Line 9: Here is where I begin working to get useable dates. (Again the ‘ugly’ files show many of the attempts I made that didn’t work). First I took the ‘Date’ column & saved as a new ‘mo\_day’ column and added 1999 to the end (for a full date format). Data head then shows us the data.

Line 10: extracts the year from the Week ending date.

Line 11: Splits the date into the month and year form the date format we forced into ‘mo\_day’.

Line 12: Look at ‘.dtypes’ again to see what fields we have been able to modify.

Line 13: put the year, month & day back together again as a new column ‘Date\_year.’ Other than changing the ‘type” of data we haven’t changed any of the original data.

Line 14 converts our new ‘Date\_year’ into a date format.

Line 15: Again use ‘.dtypes’ to check our field types.

Line 16: converts sales to an integer, we must remove the commas (,) from the fields.

Line 17: Now that we have a date format in week ending and the mo\_day column we can get the difference to find the age of the cheese. This actually took quite a bit of searching for the right code. Since I was subtracting 2 dates it wanted to make the result a date filed as well (‘0 days’ or ‘7 days’, etc). But I needed this number to be an integer.

Line 18 Once again, use ‘.dtypes’ to check our field types.

Line 19: We want to look at our data. (Specifically, January). Since we had to force a year based on the week ending date into the “prior week(s)’ data for the age, the December dates in January should be for the prior year. We see where this results in a negative value for the age.

Line 20: To correct these we’ll add back in 365. The code to modify the dates and arrive at the age looks really simple (now), but it took me over 3 hours to just figure out how to get the age. The lesson: Don’t despair when working with different methods to clean the data.

Line 21: We can drop the month and day columns we created; we have the date format we needed.

Line 22: Index the data by the year.

Line 23: separate the date into smaller year ‘chunks’ so we can take a look at it.

Line 24: separate the 2016 data, I look at both the head and tail. If one of the lines is not commented out with a ‘#’ it only shows the last command (head or tail).

Line 25: The 2015 data.

Line 26: The 2014 data

Line 27: We have the data broken into a few smaller chunks so we can take a look at the data. This first graph shows why we needed to separate out the ‘age.’ The multiple prices tied to the Week Ending Date don’t give us a concise graph.

Line 28: The 2017 data based on sales; showing all 5 age lines.

Line 29: The 2017 data based on moisture content; showing all 5 age lines.

Line 30: The 2017 data based on price; showing all 5 age lines. If we compare these 3 preliminary graphs. We notice that sales increase when the price is lowest.

Line 31: The 2016 graph shows some problems with the age data. We’re going to skip over this 2016 data for now. More data cleaning for another project.

Line 32: The 2015 data based on sales; showing all 5 age lines. This data has quite a few swings, so we’ll increase our figure size to get a better look.

Line 33: The 2015 data based on price; showing all 5 age lines. Unlike the 2017 data, we don’t see an increase in demand at a lower price.

Line 34: Let’s take the same look at the 2014 data. The 2014 sales.

Line 35: The 2014 data based on price; showing all 5 age lines. Again, we aren’t seeing a correlation between price & sales volume.

the 36: 2017 price data, we don’t see an increase in demand at a lower price. The above graphs also show that the pricing (regardless of the age) follows the same trajectory.

Line 37: We will take a different direction and look at the data by the age. We will look at the 14 day old cheese data. Index by the age and pull out all the 14 age. Again, this code looks pretty simple but it entailed many attempts to figure out how to get this.

Line 38: we’ll check the shape of this data.

Line 39: This graph compares all the 14 day prices in our data set. We see how those “zero” prices affect our graph.

Line 40: When we compare the sales for the 14 day cheddar we see a relatively steady demand over time.

Line 41: Lets see if the 28-day cheddar shows us the same information. We’ll create a 28 day dataset.

Line 42: This graph compares all the 28 day prices in our data set. Agin we see the drop with the “zero” prices.

Line 43: This shows us what we would expect the same relatively steady demand over time for 28 day cheddar.

Line 44: We can also compare a limited number of years. Here is a data set for 14 day cheddar for 2013 to 2015.

Line 45: With fewer data points (only 3 years). This graph is much less congested and easier to understand.

Line 46: This data is best viewed as time series (line) graphs. The Sales data can be plotted as a bar graph. Again, we show a relatively steady demand.

## License

none