

---

# EXTRACTION, TRANSFORMATION, AND LOAD TECHNICAL REPORT

---

BACON PRICING DATABASE CONSTRUCTION AND MAINTENANCE



BEAUTIFUL BACON GROUP  
Amit Patel, Austin Wen, Preston Hinkel

## Contents

Introduction .....	2
Scope.....	2
Project Members .....	2
Definitions, Acronyms and Abbreviations .....	3
ETL Methodology .....	3
Data Integrity .....	6
Data Refresh Frequency.....	6
Data Security.....	6
Data Loading and Availability.....	6
Data Quality .....	8

## Introduction

The purpose of this Extraction, Transformation, and Load (ETL) Technical Report is to help Beta Alpha Kappa National Fraternity office (the Client) to track daily bacon pricing. Bacon represents over 80 percent of the Client's annual food expenditure; therefore, the Client engaged Beautiful Bacon Group to create an in-house process to monitor and to store bacon related data. First, the Client would like to obtain a daily bacon pricing feed from a specified grocer to measure the immediate cost of consumption. Second, the Client would also like to track bacon's consumer price index (CPI) to better understand the effect of inflation. Finally, the Client would want to receive lean hogs commodity pricing data for future hedging strategy.

## Scope

The scope of this report covers the following:

1. Smithfield Thick Cut Naturally Hickory Smoked Bacon 16oz pricing data from Kroger.
2. Bacon, sliced, per lb. in U.S. city average price from US Bureau of Labor Statistics.
3. Lean Hogs futures from Investing.com.

Beautiful Bacon Group should create an automated process to obtain all above data from their respective sources. The obtained data should be stored in a database for the Client to retrieve easily.

## Project Members

Our team Beautiful Bacon Group consists of three data scientists: Amit Patel, Austin Wen, and Preston Hinkel. Mr. Amit Patel focused on Kroger pricing data acquisition and Github repository maintenance. Mr. Austin Wen focused on obtaining the US Bureau of Labor Statistics bacon CPI data. Mr. Preston Hinkel focused on retrieving Lean Hogs futures pricing from Investing.com.

## Definitions, Acronyms and Abbreviations

This report is technical in nature. Below is a list of definitions, acronyms and abbreviations for the Client to better understand the report.

1. Python – An interpreted general-purpose programming language that Beautiful Bacon Group used as the main engine for this ETL processes.
2. Python Module – A Python file contains additional useful statements and definitions in addition to the default. The modules used in this project are: Pandas, JSON, Requests, DateTime, SQLAlchemy, Splinter, and BeautifulSoup.
3. PostgreSQL – An open-source relational database management system for data storage.
4. CPI – Consumer Price Index. A measure for changes in the weighted price level of a given product or basket.

## ETL Methodology

Kroger Smithfield Thick Cut Naturally Hickory Smoked Bacon 16oz pricing data

### Extraction

Kroger is the Client's preferred retail grocery store to procure Smithfield Thick Cut Naturally Hickory Smoked Bacon 16oz. Given the Client's interest in tracking the daily price of the product, Beautiful Bacon Group created a Python function that access Kroger.com website directly, and then reads in all available HTML data. The initial HTML data contains more than the pricing data we are interested in. The Python function can pinpoint and extract the pricing data from the HTML raw data.

### Transformation

Once the price is located, we used Python DateTime Module to give the pricing data a date timestamp. And then convert the data into a Python Pandas DataFrame for storage.

### Load

Once the data is in a Python Pandas DataFrame format, we deployed PostgreSQL, an open-source relational database management system, to store the data into the database `bacon_db` under Table `kroger_price`.

### Extraction

The United States Bureau of Labor Statistic (BLS) publishes sliced bacon per pound consumer price index data on a monthly basis. The data history goes back to year 1980, and it is freely available via their website <https://data.bls.gov/timeseries/APU0000704111>. The website also provides an Application Programming Interface (API) method for users to download data in JSON format. We created a Python function (`download_bls_api`) that connects to BLS's API and downloads bacon CPI data. The function defaults to pull every data point since the beginning of the previous year. However, it can be adjusted to accommodate wider time period if needed.

### Transformation

The initial bacon CPI data download is in JSON format. The raw data comes with excess information such as download status and empty footnotes. We created another Python function (`transform_bls_data`) that processes and distills the raw data to two columns, a year/month column and a CPI value column. The clean result is then converted into a Python Pandas DataFrame for easy display and storage.

### Load

Like the previous Kroger pricing data ETL, we store the clean CPI data into a PostgreSQL database `bacon_db`. The function we created is called `store_cpi_data_to_db`. The function establishes connection to the PostgreSQL server, and then stores the DataFrame into the database. It has a build-in mechanism to add only the newly available CPI data to avoid duplicates in the database.

## Lean Hog Commodity Data

### Extraction

The Lean Hog Commodity data is accessible via the Investing.com website (URL: <https://www.investing.com/commodities/lean-hogs-historical-data>). The available data consists of a historical price and volume table that extends back one month by default. We use splinter's Browser and bs4's BeautifulSoup to scrape the html data from this website. We then loop through the available tables in the html object and select the table that contains the phrase "Open" as our desired table uniquely contains that phrase. We have extracted our desired table.

### Transformation

First, we do some initial cleaning on the extracted table. The table is currently in a string format that allows us to easily replace data. We format the full string prior to splitting the string into a list. We clean the list and remove any of the blank items from the list; each item in the list is going to become a prospective row. We then loop through the list and split each item, now we have a list of lists. We clean the list of lists to remove additional blank lists that may occur and we strip and leading or trailing blank-space as well.

We then create a DataFrame from the list of lists; the first list becomes the headers for the DataFrame and the rest of the lists become rows in the DataFrame. Next we clean the DataFrame by removing unnecessary text (K, PCT), cleaning up the 'Date' column, converting the 'Date' column into DataFrame format, and converting the non- DataFrame columns to float64 format.

### Load

We establish the connection to the Postgres database set up for this project: bacon\_db. We then attempt to pull the "lean\_hog\_commodity" table in case the data already exists in the database. If the table already exists in the database we append the new data from the most recent extraction and transformation. We then clean up the table by removing rows that contain duplicate Dates with the a bias to keep the duplicate Date that was extracted most

recently (ex: If Row A contains the Date “2020-06-05” and Row B contains the same Date, we would keep the Row that was pulled more recently). Now that the DataFrame contains the new and old data as well as unique ‘Date’ rows we can add the DataFrame to the Postgres database. If the table already exists in the database we replace the table with the new DataFrame, otherwise we create the table fresh with our DataFrame.

## Data Integrity

During the testing period, all three data sources were stable, and the date parsing mechanism functioned as expected. However, given all three data sources are maintained by third parties without entering explicit data contracts, it is possible the future updates might cause interruption to the ETL process. The Client is encouraged to actively monitor the ETL process. The print output messages from the Python code will provide valuable information whether each ETL is successful.

## Data Refresh Frequency

The Kroger and Lean Hogs Commodity pricing data should be refreshed daily at or after 5 pm ET, and the Bacon CPI data requires monthly refresh on the last day of the month. The Kroger pricing data is the only time sensitive source because the website does not provide historical data. On the other hand, the Lean Hogs Commodity pricing and Bacon CPI historical data are available via their respective sources in case a scheduled refresh is missed.

## Data Security

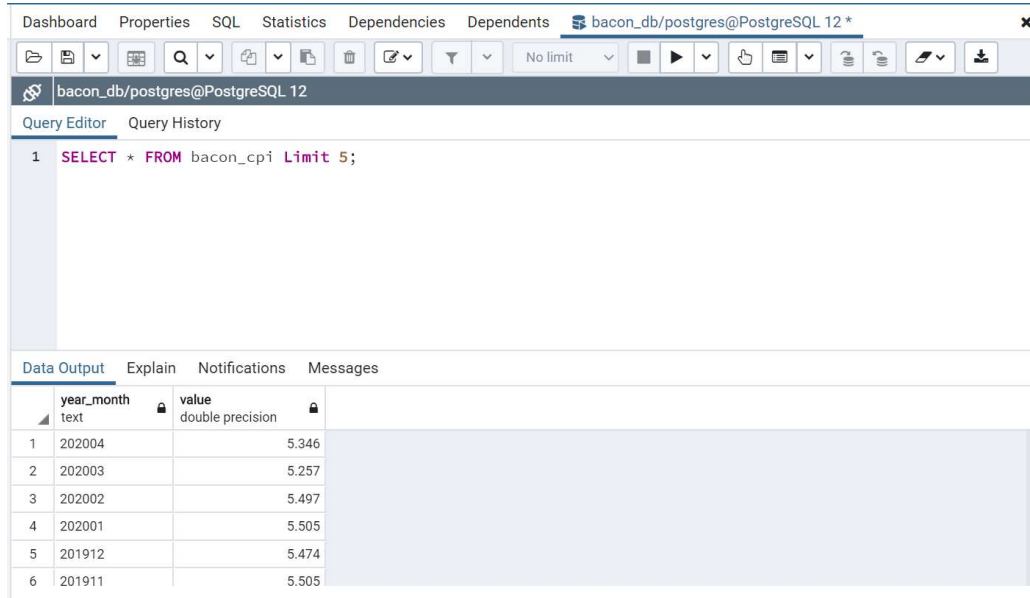
Along with the scripts that we have provided as part of our work product we have included a config.py file that will contain the API key as well as the access information for the Postgres database. It is important to ensure protection of this config.py file. While the data is all accessible via the websites that the Beautiful Bacon Group utilized, it is still best practices to protect the ETL data in the database.

## Data Loading and Availability

All Python codes for this ETL project are saved in a Jupyter Notebook Beautiful\_Bacon\_Group\_ETL.ipynb. The notebook handles an end-to-end process from extracting data from the sources to loading the clean data to Postgres database. The Client can retrieve all data via Postgres database bacon\_db, which

consists of three tables: Kroger\_price, bacon\_cpi, and lean\_hog\_commodity. Two example queries are listed below.

Example 1: Select data from bacon\_cpi table.



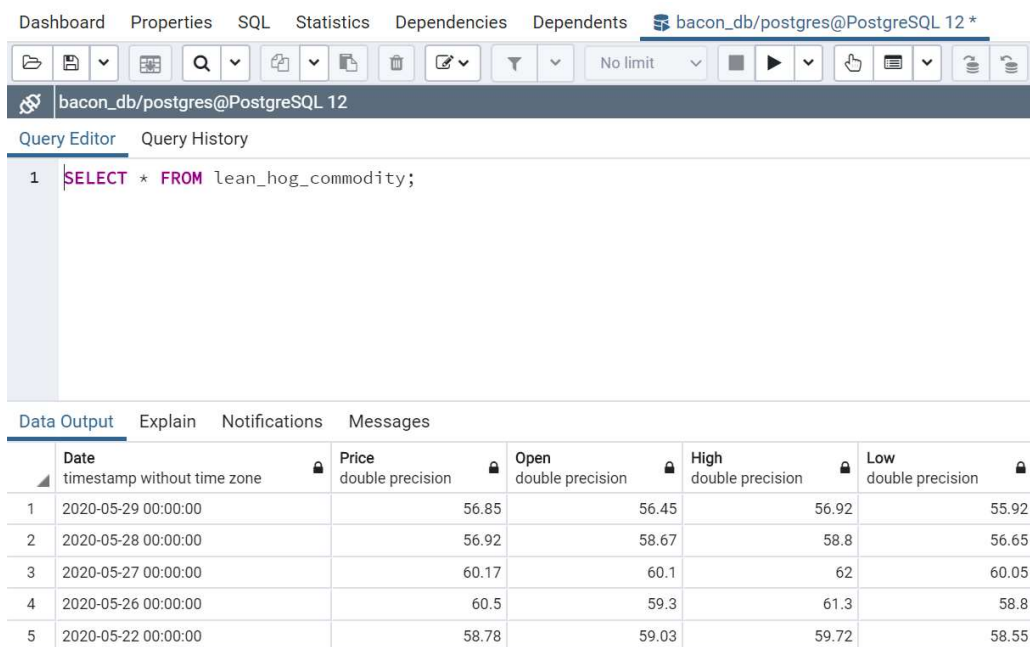
The screenshot shows a database query editor interface. The top navigation bar includes 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', and 'Dependents'. The current tab is 'bacon\_db/postgres@PostgreSQL 12 \*'. Below the navigation bar is a toolbar with various icons. The main area is the 'Query Editor', which contains the following SQL query:

```
1 SELECT * FROM bacon_cpi Limit 5;
```

Below the query editor, there are tabs for 'Data Output', 'Explain', 'Notifications', and 'Messages'. The 'Data Output' tab is selected, showing a table with the following data:

	year_month text	value double precision
1	202004	5.346
2	202003	5.257
3	202002	5.497
4	202001	5.505
5	201912	5.474
6	201911	5.505

Example 2: Select data from lean\_hog\_commodity table.



The screenshot shows a database query editor interface. The top navigation bar includes 'Dashboard', 'Properties', 'SQL', 'Statistics', 'Dependencies', and 'Dependents'. The current tab is 'bacon\_db/postgres@PostgreSQL 12 \*'. Below the navigation bar is a toolbar with various icons. The main area is the 'Query Editor', which contains the following SQL query:

```
1 SELECT * FROM lean_hog_commodity;
```

Below the query editor, there are tabs for 'Data Output', 'Explain', 'Notifications', and 'Messages'. The 'Data Output' tab is selected, showing a table with the following data:

	Date timestamp without time zone	Price double precision	Open double precision	High double precision	Low double precision
1	2020-05-29 00:00:00	56.85	56.45	56.92	55.92
2	2020-05-28 00:00:00	56.92	58.67	58.8	56.65
3	2020-05-27 00:00:00	60.17	60.1	62	60.05
4	2020-05-26 00:00:00	60.5	59.3	61.3	58.8
5	2020-05-22 00:00:00	58.78	59.03	59.72	58.55



## Data Quality

The success criterion for this project would require that each of the data pulls are able to continue successfully and that the data remain in a clean and consistent format. We met with the Beta Alpha Kappa National Fraternity office stakeholders and ran through the necessary steps to procure and access the data and they signed off both on the data quality as well as the ease of use.

The Client will also ensure that the tables do not grow faster than the data production should allow: the lean hog commodity data and Kroger data should grow at a maximum rate of one row per day and the bacon cpi data should grow at a maximum rate of one row per month.

Those are the rates at which the new data is produced, if the client witnesses any larger growth or data error they should contact the Beautiful Bacon group for a solution.