JC Penney Analysis

ITNPBD2 - Assignment

Student: 2710017

13 December 2024

Table of contents

1 Summary

This report describes the analysis of JC Penney customer sales and review information that has been provided in five files extracted from JCP operational information systems. The analysis was conducted using Python and Jupyter Notebooks and followed the CRISP-DM process to collate and validate the provided data before then performing appropriate data analysis to identify any insights from this data. All Python code and output is included to support the observations made.

The primary conclusion from this analysis is that the quality of the data is extremely poor, there are many gaps and inconsistencies; to the extent that meaningful business analysis and conclusions are difficult to complete. So a set of data validation steps has been developed that can easily be repeated once reliable data is sourced and then further business analysis can be completed. The root causes of the data quality issues should be examined, looking at the extraction process and also the operational systems that sourced the data. Given that JCP is in the process of overhauling its customer and stock information systems then there is an opportunity to include changes that will improve future data extraction.

Any business actions need to be treated carefully given the data quality issues, but two areas that might be worth examining have been identified. Firstly, there may be an opportunity to rationalise the large product range and focus on a small number of products and with a smaller product range. Profit margin and cost data will be needed to do this. Secondly, the geographic spread of customers and sales revenue doe not reflect the spread of populations across US states; the reason for this could reveal opportunities to increase sales in several large US states.

2 Objectives & Approach

2.1 Objectives

This report describes the findings from an analysis of sales review data for the major US retail chain, JCPenney. It is the final assignment for the module ITNPBD2 - Representing and Manipulating Data. This analysis uses five provided datasets, with some supplementary data to augment.

The report is structured into the following parts:

- Summary
- Objectives & Approach
- Data Collation, Exploration & Preparation
- Data Analysis
- Conclusions
- References
- Appendix

2.2 Approach

This is intended primarily as a business report to identify and make recommendations for improvements, with appropriate supporting evidence. In addition, there is a need to meet the additional assignment objectives of:

- Demonstrate that a data science process (here CRISP-DM) has been followed;
- Demonstrate the use of Python for data manipulation and analysis.

Once initial analysis commenced it quickly became evident that two challenges needed to be addressed. The first is that trying to produce a single PDF report with commentary, Python code and runtime output would result in a very long and difficult to read artefact. The second challenge was that a brief examination of the data using Data Wrangler (Microsoft, 2023) quickly revealed that the data was of very poor quality and would make meaningful business conclusions possibly misleading.

As a result, the following approach has been taken:

- Focus on the data preparation part of the process and make recommendations on areas to improve
- Carry out some limited business analysis, but no significant geographic or demographic aspects
- Establish a set of processes that can be reused when better quality source data is available (ie several Jupyter Notebooks)

And in terms of assignment documents, have a two part PDF report:

- A largely text report with limited Python code and output
- A supporting appendix detailing all Python code and output for the dat preparation stage

2.3 CRISP-DM

The approach to the analysis follows parts two, three and four of the six stage "CRoss-Industry Standard Process for Data Mining' (CRISP-DM) process (Ncr et al., 1999), (Hotz, 2024). In summary, this process is:

1. Business Understanding: Define project objectives and requirements by collaborating with stakeholders
2. Data Understanding: Collect and explore data, analysing its characteristics and quality
3. Data Preparation: Clean, handle missing values, and transform variables to create a structured dataset
4. Modelling: Apply various techniques such as machine learning algorithms or statistical models to the prepared data
5. Evaluation: Rigorously assess models based on predefined criteria, including performance and reliability
6. Deployment: Integrate successful models into existing systems and monitor their effectiveness

2.4 Who Are JCPenney?

JCPenney (JCP) is a major North American department store chain (Wikipedia), operating as Penney OpCo LLC. JCP has 656 stores in 49 states plus Puerto Rico according to the JCP Store Locator.

In 2020, JCP filed for bankruptcy and were purchased by an asset management company, a large number of stores were also closed. Subsequently, in August 2023, JCP announced a major turnaround plan to replace its current website and inventory management systems, as well as make major upgrades to its retails stores (ModernRetail, 2023). Note that it is assumed that the data used here was extracted from the operational systems prior to any overhaul of information systems.

3 Data Collation, Exploration & Preparation

This section describes the sequence and findings of the process of collating the source data, validating and then preparing it for subsequent analysis. Once the preparation processes are all completed the resultant dataframes are persisted locally (using the magic command 'store') making the data available for further analysis. This section focuses on the results of the process and the Appendix details all the Pythin code used.

3.1 Provided Data Sources & Content

The provided data sources for this analysis of JC Penney consists of two JSON files and three CSV files:

- JSON: jcpenney_products, jcpenney_reviewers
- CSV: products, reviews, users

It was not immediately obvious what the relationships between the two types of data files were but the json and CSV files appear to be partial duplicates of each other; also the three CSV files hold slightly less information (eg sales price is missing from the csv files). The CSV files appear to be a first attempt to extract data from the json files (eg the json products file has a JSON field holding multiple user reviews and this looks like it was extracted to prepare the reviews.csv file).

Given the above, the approach used in this analysis was to go back to the 'original' JSON files and work from these but with appropriate sanity checks against the three CSV files to make sure no data was missed or inconsistent.

3.1.1 Load JSON Data Files

It is assumed that the data is a snapshot extract of sales information from JCP operational system and the bulk of this has been flattened and then used to create the jcpenney_products.json file and the jcpenney_reviewers.json file.

The tables below show the data items and key counts for each file.

File Summary for: jcpenney_products.json

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	7982	0	0	7982	object	7982	0	0	0
sku	7982	0	67	6044	object	7982	0	0	0
name_title	7982	0	0	6002	object	7982	0	0	0
description	7982	0	543	5620	object	7982	0	0	0
list_price	7982	0	2166	1037	object	7982	0	0	0
sale_price	7982	0	18	2063	object	7982	0	0	0
category	7982	0	636	1169	object	7982	0	0	0
category_tree	7982	0	636	1997	object	7982	0	0	0
average_product_rating	7982	0	0	153	float64	0	0	7982	0
product_url	7982	0	0	7982	object	7982	0	0	0
product_image_urls	7982	0	157	6519	object	7982	0	0	0
brand	7982	0	0	721	object	7982	0	0	0
total_number_reviews	7982	0	0	22	int64	0	7982	0	0
Reviews	7982	0	0	7982	object	0	0	0	7982
Bought With	7982	0	0	7982	object	0	0	0	7982

Figure 3.1: Products Structure

First 3 Rows

	uniq_id	sku	name_title	description	list_price	sale_price	category	category_tree
0	b6c0b6bea69c722939585baeac73c13d	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant	You'll return to our Alfred Dunner pull-on cap...	41.09	24.16	alfred dunner	jcpenney|women|alfred dunner
1	93e5272c51d8cce02597e3ce67b7ad0a	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant	You'll return to our Alfred Dunner pull-on cap...	41.09	24.16	alfred dunner	jcpenney|women|alfred dunner
2	013e320f2f2ec0cf5b3ff5418d688528	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant	You'll return to our Alfred Dunner pull-on cap...	41.09	24.16	view all	jcpenney|women|view all

Figure 3.2: Products Data

File Summary for: jcpenney_reviewers.json

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
Username	5000	0	0	4999	object	5000	0	0	0
DOB	5000	0	0	52	object	5000	0	0	0
State	5000	0	0	57	object	5000	0	0	0
Reviewed	5000	0	0	4030	object	0	0	0	5000

First 3 Rows

	Username	DOB	State	Reviewed
0	bkpn1412	31.07.1983	Oregon	[cea76118f6a9110a893de2b7654319c0]
1	gqjs4414	27.07.1998	Massachusetts	[fa04fe6c0dd5189f54fe600838da43d3]
2	eehe1434	08.08.1950	Idaho	[]

Figure 3.3: Reviewers Structure & Data

3.2 Working Data Structure - Validation & Augmentation

3.2.1 Data Structure Summary

The five data sources provided were examined and validated in more detail in order to understand the contents and data structure, and to complete any cleaning required. In addition, areas requiring augmentation were identified and additional data was sourced and combined with the original sources. The resulting working data structure consists of five dataframes, their preparation and content is described in the sections that follow, in the order of their preparation. In summary the five dataframes are:

Sales

- Details of all sales activity, 7,982 sales records
- Uniquely identified by the key 'uniq_id', in the Pandas dataframe: *sales_df*
- Sourced from the provided file: jcpenney_products.json
- A relatively small number of sales prices were missing

Customer Sales Reviews

- Details of all customer reviews, 39,063 in total
- Uniquely identified by combined 'uniq_id' + 'customer_id', in the Pandas dataframe: *customer_reviews_df*
- Sourced from the provided file: jcpenney_products.json
- Major issues with the quality of the reviews and and so of limited use in analysis. For example, 15,535 reviews appear to be duplicated across different customers. This could be a data export issue or even the introduction of fake reviews

Customer Details

- A reference list of 5,001 unique JCP customers who have submitted reviews of purchases
- Uniquely identified by the key 'customer_id', in the Pandas dataframe: *customers_df*

- Sourced from the provided file: jcpenney_reviewers.json
- Major issues with the quality of date of birth information. Appears to be artificially generated and so of limited use in analysis

Stock Details

- A reference list of 6,110 unique stock items
- Uniquely identified by the key 'sku', in the Pandas dataframe: *stock_df*
- Sourced from the provided file: jcpenney_products.json
- Derived from the 6,044 unique items in the file jcpenney_products.json
- Some issues with basic data differences for stock but these have been rationalised

States & Territories

- A reference list of the 57 US states and territories, with population and JCP store numbers per state
- Uniquely identified by the key 'state_ISO', in the Pandas dataframe: *states_df*
- Sourced from the file: JCP_Stores_State_Collated.csv
- Originated from the JCP store locater website and from the US Census Bureau

3.2.2 States & Territories

A reference list for all US states and territories. Contains 57 items (51 states and 6 territories). This is the ISO code for later validation of the provided customers data and with population data and JCP store numbers to assist later geographic analysis.

The data was sourced from:

- JCP's store locator, see website
- US Census Bureau, see website

Data Content

After review and validation the created dataframe, states_df, has 57 unique items. It consists of:

- territory_flag - Indicates whether a state or a territory
- state_ISO - ISO code of the state, territory
- state_name - Name of the state, territory
- population - Population at 2023
- stores_total - Total number of stores at November 2024

Collation & Validation

The additional data file, JCP_Stores_State_Collated.csv, was loaded and validated.

Summary of States – CSV

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
territory_flag	57	0	0	2	object	57	0	0	0
state_ISO	57	0	0	57	object	57	0	0	0
state_name	57	0	0	57	object	57	0	0	0
population	57	0	0	57	int64	0	57	0	0
stores_total	57	0	0	26	int64	0	57	0	0

First 3 rows

	territory_flag	state_ISO	state_name	population	stores_total
0	State	AL	Alabama	5108468	9
1	State	AK	Alaska	733406	1
2	State	AZ	Arizona	7431344	17

Figure 3.4: States Structure & Data

3.2.3 Customers

Details of customers that have completed a review of a purchase made. With 5,001 unique customer records. All customers have a date of birth, however examination of this showed that only 14 were really unique and appear to be artificially generated. Arguably this field should be dropped as it will not provide any meaningful results. However, it has been retained purely so that it can be used to demonstrate analysis techniques.

Data Content

After review and validation the created dataframe, customers_df, has 5,001 unique customers. It consists of:

- customer_id - A unique alphanumeric id
- DOB - Date of birth
- state_ISO - ISO code for the state or territory. A cross-reference to the states_df

Collation & Validation

The provided data file, jcpenney_reviewers.json, was examined. As these appears to be detailing customers that have completed a review, the term 'customer' was used instead of reviewer. The following actions were taken:

- Fields Rename: Columns renamed to be consistent with other dataframes
- Duplicates: One customer_id was used twice. To preserve information, it was decided to keep the duplicates and assign them a new unique customer_id
- Date of Birth: Surprisingly for 5,001 customers only 52 birth dates were found. Closer examination revealed that a day, month sequence was incremented across years, with only 14 unique dates ranging only from 26 July to 8 August.

- States: When validating against the states reference file to obtain ISO codes, 187 customers did not match due to the incorrect naming of the US Virgin Islands and US Minor Outlying Islands, so these were corrected. Only the ISO code was retained and the full state name dropped, in preference to it being looked up when required
- uniq_id_list - This list was dropped once the details had been cross-checked against the new customer_reviews and sales dataframes.

Summary for customers

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
customer_id	5000	0	0	4999	object	5000	0	0	0
DOB	5000	0	0	52	object	5000	0	0	0
state_name	5000	0	0	57	object	5000	0	0	0
uniq_id_list	5000	0	0	4030	object	0	0	0	5000

First 3 rows – Renamed Columns

	customer_id	DOB	state_name	uniq_id_list
0	bkpn1412	31.07.1983	Oregon	[cea76118f6a9110a893de2b7654319c0]
1	gqjs4414	27.07.1998	Massachusetts	[fa04fe6c0dd5189f54fe600838da43d3]
2	eehe1434	08.08.1950	Idaho	[]

Duplicated Customers:

	customer_id	DOB	state_name	uniq_id_list
731	dqft3311	28.07.1995	Tennessee	[5f280fb338485cfc30678998a42f0a55]
2619	dqft3311	03.08.1969	New Mexico	[571b86d307f94e9e8d7919b551c6bb52]

Figure 3.5: Customer Structure & Duplication

3.2.4 Stock Details

Details of all stock (product) data. Contains 6,110 unique lines of stock with each uniquely identified by the key 'sku'. For each stock line the details include description and its list price. There was significant inconsistency of basic details (name, description, list price, image url) for stock items in the provided products file. An attempt has been made to rationalise the data by retaining the most commonly used data items.

Data Content

After review and validation the created dataframe, stock_df, has 6,110 unique stock records. It consists of:

- sku - The unique identifier for the stock item
- stock_name - Short name for the stock item
- description - A long description of the stock
- list_price - The standard price for the stock
- stock_image_url - URL for the website image for the stock
- brand - The manufacturer's name for the stock item, eg 'Alfred Dunner'

Collation & Validation

The provided data file, jcpenney_products.json, was examined and all stock specific data extracted into the stock_df dataframe. The following actions were taken:

- Missing SKU ids: 67 were missing, so generated ids were added according in line with the most common format structure, to pp600nnnnnnn
- Drop Fields: Drop all fields that are sales specific: 'uniq_id', 'sale_price', 'category', 'category_tree', 'average_product_rating', 'product_url', 'total_number_reviews', 'Reviews', 'Bought With'
- Stock Name: A significant number of names differed for the same sku. The first name has been retained
- Description: A significant number of descriptions differed for the same sku or were missing. The first name has been retained. However, still 50 had no description so 'No Description Available' was added
- List Price: A significant number of items had different prices for the same stock. So the most common price for each item was used. Even so, 95 stock items do not have a list price
- Stock Image URL: For 170 stock items, the urls did not all match and so the most frequent one was retained. A sample request of images was successful (nb a VPN was requied as non-US locations were filtered).
- Brand: No missing or duplicated, so just copy one
- Rationalise Stock: Retain only a single unique sku record

Reasons for Splitting Sales & Stock Data

The provided file, jcpenney_products.json, appears to contain core stock information and sales specific information. For example a stock list price and a different sales price that varied depending on different sales categories. So stock data has been split out and cleaned in order to be able to more easily analyse stock vs sales data.

It has been assumed that the field sku is the 'Stock Keeping Unit' see Wikipedia and should be a unique identifier. Therefore all instances of sku have been reviewed and collapsed down into a stock list dataframe, separate from sales activity.

```
duplicated skus: 3026
Summary For Stock Data - After Cleaning
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
sku	6110	0	0	6110	object	6110	0	0	0
stock_name	6110	0	0	5894	object	6110	0	0	0
description	6110	0	0	5596	object	6110	0	0	0
list_price	6110	0	0	990	float64	0	0	6110	0
stock_image_url	6110	0	0	5991	object	6110	0	0	0
brand	6110	0	0	721	object	6110	0	0	0

```
First 3 Rows
```

	sku	stock_name	description	list_price	stock_image_url	brand
0	0903a80	KitchenAid® Artisan® 5-qt. Stand Mixer KSM150PS	The mixer you've always dreamed of. Unique mix...	604.31	http://s7d9.scene7.com/is/image/JCPenney/09006...	Kitchen Aid
1	13cab12	JCPenney Home™ Saratoga Cut-to-Width Fringed B...	Saratoga cut--to-width blackout shade features...	27.80	http://s7d2.scene7.com/is/image/JCPenney/DP121...	JCP HOME
2	13cab4b	JCPenney Home™ Saratoga Cut-to-Width Unfringed...	The Saratoga cut-to-width roller shade effecti...	24.17	http://s7d2.scene7.com/is/image/JCPenney/DP080...	JCP HOME

Figure 3.6: Stock Structure & Data

3.2.5 Sales

Details of all sales activity. Contains 7,982 sales records with each uniquely identified by the key 'uniq_id'. The data for each sale includes the sales price, stock reference and sales channel information. Most of the data appeared complete and reasonably, although several hundred sales prices were in an invalid format or missing; the relatively small number should not skew later analysis.

Data Content

After review and validation the created dataframe, sales_df, has 7,982 records. It consists of:

- uniq_id - A unique identifier for the sales activity
- sku - A cross-reference for stock data in the stock_df
- sale_price - The price that the sales was
- category_tree - A string breaking down the structure of the sales channel
- category - The bottom level of the category tree
- sales_product_url - JCP website url for the product details as sold
- average_product_rating - An average of the customer review scores (1 to 5) for this sale
- total_number_reviews - The total number of customer reviews for this sale
- bought_with_list - other sales at the same time as this sale

Collation & Validation

The provided data file, jcpenney_products.json, was examined and all sales specific data extracted into the sales_df dataframe. The following actions are highlighted:

- Invalid & Missing Prices: 263 sales prices were in a range format (34.5-45.9) and these were converted taking the average. And 18 had no price and so were zeroed
- Categories Missing: 636 categories, category trees are missing. About 10% of the 7,982 sales
- Sales Product URL: All good, no missing or duplicated. However, a sample of requests to use the URL all failed
- Bought With: This was not reviewed and has been retained as source

```
Missing: Categories 636 Trees 636
Duplicated URLs: 0
Summary For Sales Data - After Cleaning
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	7982	0	0	7982	object	7982	0	0	0
sku	7982	0	0	6110	object	7982	0	0	0
sale_price	7982	0	0	1992	float64	0	0	7982	0
category	7982	0	0	1169	object	7982	0	0	0
category_tree	7982	0	0	1997	object	7982	0	0	0
average_product_rating	7982	0	0	153	float64	0	0	7982	0
sales_product_url	7982	0	0	7982	object	7982	0	0	0
total_number_reviews	7982	0	0	22	int64	0	7982	0	0
bought_with_list	7982	0	0	7982	object	0	0	0	7982

```
First 3 Rows
```

	uniq_id	sku	sale_price	category	category_tree	average_product_rating	
0	b6c0b6bea69c722939585baeac73c13d	pp5006380337	24.16	alfred dunner	jcpenney|women|alfred dunner	2.625	http://www.j
1	93e5272c51d8cce02597e3ce67b7ad0a	pp5006380337	24.16	alfred dunner	jcpenney|women|alfred dunner	3.000	http://www.j
2	013e320f2f2ec0cf5b3ff5418d688528	pp5006380337	24.16	view all	jcpenney|women|view all	2.625	http://www.j

Figure 3.7: Sales Structure & Data

3.2.6 Customer Sales Reviews

There are a total of 39,063 reviews but only 29,464 appear to be unique review comments. Further analysis found that 15,535 (40%) of reviews were used by several customers, worst case being several instances of 18 customers using the same comments. This could be because the sample data has been automatically generated or that customer ids are being created to generate false reviews. This data has *not* been dropped from the dataset, although later analysis of the reviews could be misleading.

Data Content

After review and validation the created dataframe, customer_reviews_df, has 39,063 customer reviews. It consists of:

- uniq_id - A cross-reference for sales data in the sales_df
- customer_id - A cross-reference for customer data in the customers_df
- review_text - Review comments made by the customer for the sale
- review_score - The score of 1 to 5 given by the customer

Collation & Validation

The provided data file, jcpenney_products.json, was examined and all review specific data extracted into the customer_reviews_df dataframe. This was carried out after first creating the stock_df and the sales_df. The following actions were taken:

- Reviews: From the sales details the list of customer reviews was decoded from its JSON format
- Ratings: The totals and averages across multiple customers were cross-checked to the sales data
- Customers: 17 reviews did not have a valid existing customer and a dummy customer was created (but flagged with state ISO of XX and DOB of NAT) so as not to loose the review data
- In 11 instances there were two reviews for the same customer and sale, but with different review comments; these were left as only a small number

```
For: e5bdf53f2374569526c9f4d55afdd88e not all customers match
Adding dummy customer for ID: dqft3311
Summary For Customer Reviews Data - All Reviews
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	39063	0	0	7982	object	39063	0	0	0
customer_id	39063	0	0	4993	object	39063	0	0	0
review_text	39063	0	0	29464	object	39063	0	0	0
review_score	39063	0	0	5	object	0	39063	0	0

```
First 3 Rows
```

	uniq_id	customer_id	review_text	review_score
0	b6c0b6bea69c722939585baeac73c13d	fsdv4141	You never have to worry about the fit...Alfred...	2
1	b6c0b6bea69c722939585baeac73c13d	krpz1113	Good quality fabric. Perfect fit. Washed very ...	4
2	b6c0b6bea69c722939585baeac73c13d	mbmg3241	I do not normally wear pants or capris that ha...	4

Figure 3.8: Reviews Data & Structure

```
Out of a total of 39063 reviews 15535 are duplicates.
Or approximately 40%
Several worst case situations with 18 customers using the same review comments.
```

Figure 3.9: Reviews Duplicates

CSV File Rejected

The reviews.csv file was examined and the scores were found to have a large number of zero values (11,265 out of 39,063) and a quick examination showed that many scores differ between the JSON and CSV source. This confirmed the decision to reject the CSV data and only use the JSON source.

4 Data Analysis

Once the data preparation stage was completed, the resulting five dataframes were used to complete data analysis. However, the data preparation stage raised several data quality issues which have potentially reduced the business value of the analysis. In particular, conclusions from geographic and demographic analysis may be incorrect given the underlying customer data issues; also using customer reviews is problematic given the different interpretations that appear to have been taken for the review score ranking direction. Nevertheless, some data analysis has been completed.

In summary, the themes analysed were:

- Customer Reviews
- Product Sales
- Geographic Impact

```
# Retrieve the completed working dataframes for analysis
%store -r sales_df stock_df customer_reviews_df customers_df states_df

# Libraries For file handling and dataframes
import os
import json
from IPython.display import display
import pandas as pd

# Libraries for maths, plots
import math
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('ggplot')

# Other Libraries
import math
import nltk
nltk.download('vader_lexicon')
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/stuartgow/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

```
True
```

4.1 Customer Reviews

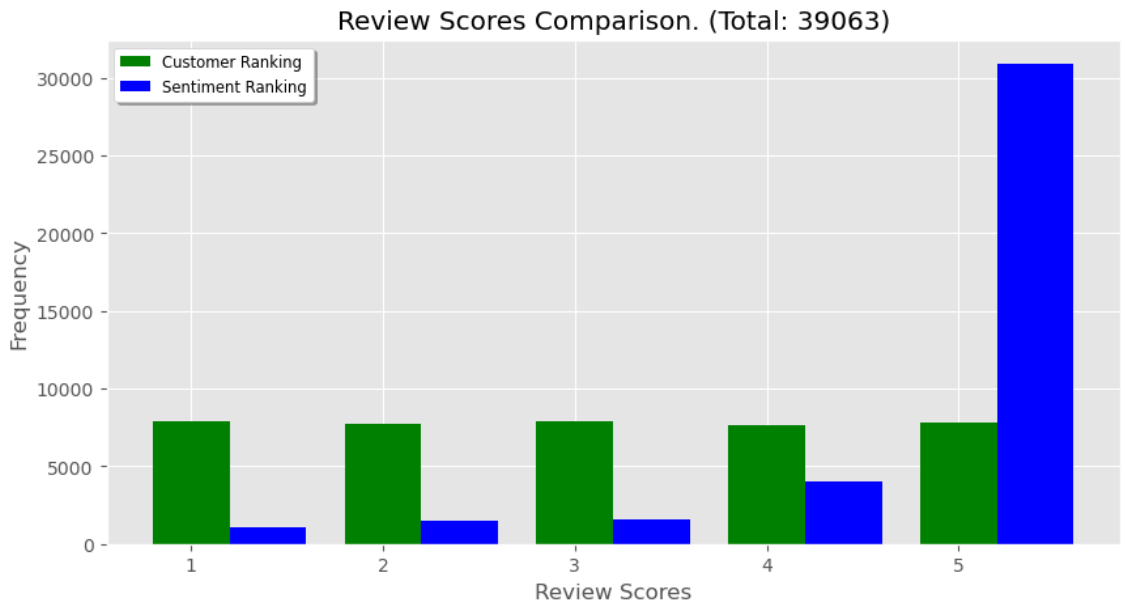
All customer reviews consist of descriptive text and a score, it appears that the score is a ranking from poor to good (1 to 5). Looking at the customer ranking scores there was a very even distribution of the scores which is unexpected. To try to understand the reviews better, a sentiment analysis was completed on the descriptive text for all reviews and then used to generate a 1 to 5 sentiment ranking.

The ranking produced from sentiment analysis were generally very positive which seems unusual; however, a manual examination of several reviews did support this as a very large number do have positive text. So this generated ranking was retained for comparisons.

The difference between customer rankings and sentiment rankings was then compared to get a sense of the accuracy of the customer rankings. The bar plot below shows that nearly 50% of the 39,000 reviews had significantly different rankings and that the rankings given by customers were generally much more conservative than the sentiment expressed in the the actual text. A manual spot-check of several customer reviews was carried out to better understand this issue. The reason for the big difference is that a very signifiant number of customers have interpreted the ranking in the opposite way, ie good to bad (1 to 5).

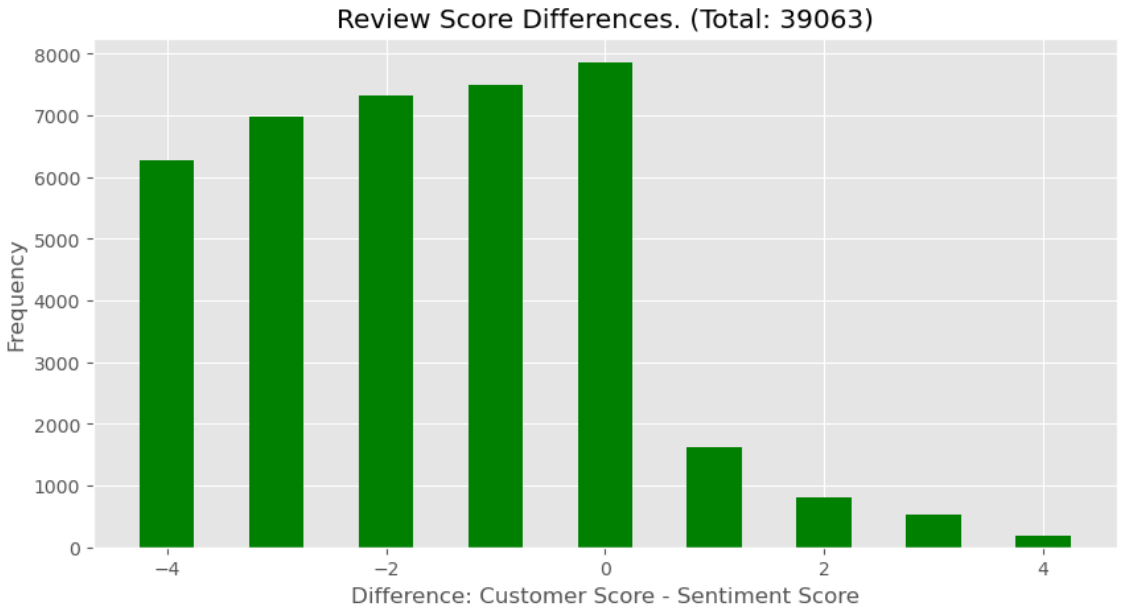
```
 # Sentiment analysis of reviews

#import nltk
#nltk.download()
from nltk.sentiment.vader import SentimentIntensityAnalyzer
analyser = SentimentIntensityAnalyzer()

#  Simple function to categorise sentiment to create a ranking score similar to customer
def sentiment_categorise(sentiment):
    # Uses the sentiment compound score defined as +ve >= 0.05, -ve <=-0.5, neutral in be
    # Set the 1 to 5 score
    if sentiment <= -0.5:
        return 1
    elif sentiment <= -0.05:
        return 2
    elif sentiment < 0.05:
        return 3
    elif sentiment < 0.5:
        return 4
    else:
        return 5

# Obtain the sentiment, compound score (-1 to 1) for each review
temp_customer_reviews = customer_reviews_df.copy()
sentiment_compound_scores = [analyser.polarity_scores(review_text)['compound'] for review

# Classify the sentiments to align to the review score ranking 1 to 5 and put into the te
temp_customer_reviews['sentiment_review_score'] = [sentiment_categorise(x) for x in senti
temp_customer_reviews['sentiment_compound_score'] = sentiment_compound_scores
temp_customer_reviews['review_score'] = temp_customer_reviews['review_score'].astype(int)
```

```
 # Compare the review rankings for customers compared to the sentiment analysis

rank_count_cust = temp_customer_reviews['review_score'].value_counts()
rank_count_sent = temp_customer_reviews['sentiment_review_score'].value_counts()
#display(rank_count_cust, rank_count_sent)

fig, ax = plt.subplots(figsize =(10, 5))
ax.set_title(f'Review Scores Comparison. (Total: {len(temp_customer_reviews['review_score
ax.set_xlabel('Review Scores')
ax.set_ylabel('Frequency')
bar_width = 0.4

columnsC = rank_count_cust.index
columnsS = [x + bar_width for x in rank_count_sent.index]

barsC = ax.bar(columnsC, rank_count_cust, color='green', width=bar_width, label='Customer
barsS = ax.bar(columnsS, rank_count_sent, color='blue', width=bar_width, label='Sentiment
ax.legend(fontsize='small', loc='upper left', shadow=True, facecolor='white')
plt.show()

# Tidy Up
del rank_count_cust, rank_count_sent, columnsC, columnsS
```




```
 # Compare the customer review rankings with the sentiment analysis
# Calculate the frequency of differences
score_differences = temp_customer_reviews['review_score'] - temp_customer_reviews['sentim
differences_frequency = score_differences.value_counts()
#display(differences_frequency)

# Plot the frequencies
fig, ax = plt.subplots(figsize =(10, 5))
ax.set_title(f'Review Score Differences. (Total: {len(score_differences)})')
ax.set_xlabel('Difference: Customer Score - Sentiment Score')
```

```
ax.set_ylabel('Frequency')
bars = ax.bar(differences_frequency.index, differences_frequency, color='green', width=0.
plt.show()

# Tidy Up
del analyser, sentiment_compound_scores, score_differences, differences_frequency
# temp_customer_reviews,
```



4.2 Product Sales Analysis

The distribution of products sold was examined and a surprisingly large spread of sales revenue across the product range was identified. The vast majority of sales revenue is generated from products in the £20 to $4,500 price range, whereas the high-end and low-end products do not generate as significant a sales revenue.

Product Sales Revenue

The entire product range totals 6,110 items and generated a sales revenue of approximately $4,000k. Within this, 100 products generated $1,600k, or 40%, of the sales revenue. And over 2,000 products generated less than $150k sales revenue.

Product Pricing

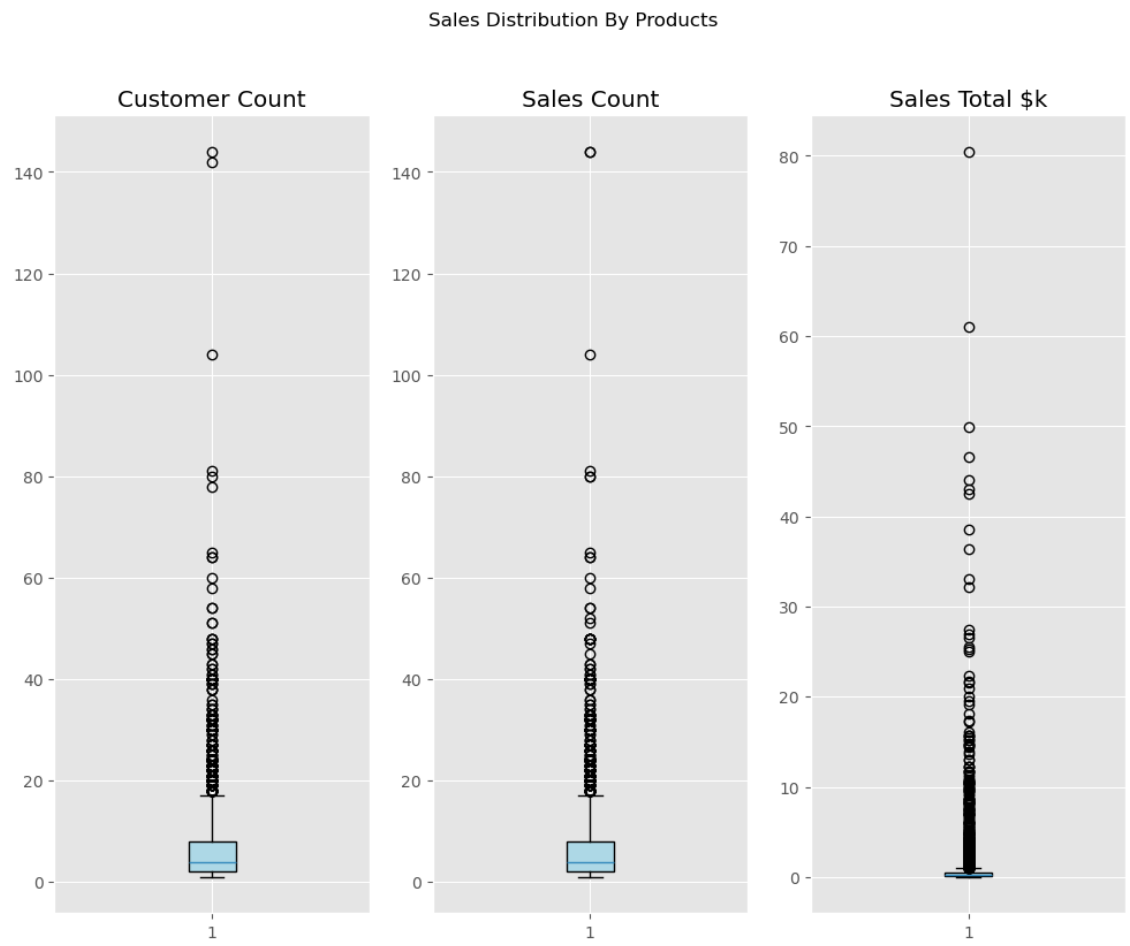
Approximately $2,000k, 50% of sales revenue, is generated by the 200 highest priced products (all over $500 each); the majority of which is from products in the $1,500 to $4,500 price range. The majority of products (over 3,500) are in the $20 to $100 price range and generate approximately $1,250k sales revenue. The very lowest priced products, 1,200 items under $20, generate very little sales revenue.

```
# Examine product sales

import pandas as pd
import matplotlib.pyplot as plt
```

```
plt.style.use('ggplot')
import seaborn as sns
import scipy.stats

# Create customer sales
cust_sales = pd.merge(customer_reviews_df, customers_df, on='customer_id')
cust_sales = pd.merge(cust_sales, sales_df, on='uniq_id')
cust_sales_stock = pd.merge(cust_sales, stock_df, on='sku', how='left')

# Exclude without a list price
no_sale_price = len(cust_sales_stock[cust_sales_stock['sale_price'] <= 0])
print(f'Sales with no sale price: {no_sale_price}')
cust_sales_stock = cust_sales_stock[cust_sales_stock['sale_price'] > 0]

# Aggregate totals for each product the sales information
stock_summary = cust_sales_stock[['sku', 'stock_name']].drop_duplicates()

groups = cust_sales_stock.groupby('sku')
stock_summary['sales_count'] = groups['sale_price'].transform('count')
stock_summary['sales_total$k'] = groups['sale_price'].transform('sum')
stock_summary['sales_total$k'] = (stock_summary['sales_total$k'] / 1000).round(4)
stock_summary['sales_price'] = groups['sale_price'].transform('mean')
stock_summary['uniq_cust_count'] = groups['customer_id'].transform('nunique')

# Box plot of the distribution for products
fig, (ax0, ax1, ax2) = plt.subplots(1, 3, figsize =(12, 9))
fig.suptitle('Sales Distribution By Products')

ax0.set_title('Customer Count')
ax1.set_title('Sales Count')
ax2.set_title('Sales Total $k')

box0 = ax0.boxplot(stock_summary['uniq_cust_count'], patch_artist=True)
for patch in box0['boxes']:
    patch.set(facecolor='lightblue')
box1 = ax1.boxplot(stock_summary['sales_count'], patch_artist=True)
for patch in box1['boxes']:
    patch.set(facecolor='lightblue')
box2 = ax2.boxplot(stock_summary['sales_total$k'], patch_artist=True)
for patch in box2['boxes']:
    patch.set(facecolor='lightblue')
plt.show()

# Examine top and bottom 5 products by sales value
top5 = stock_summary.nlargest(5, 'sales_total$k')
bottom5 = stock_summary.nsmallest(5, 'sales_total$k')

print('Top Five Products by Sales Value')
display(top5[['stock_name', 'sales_price', 'uniq_cust_count', 'sales_count', 'sales_total
print('Bottom Five Products by Sales Value')
display(bottom5[['stock_name', 'sales_price', 'uniq_cust_count', 'sales_count', 'sales_to
```

```
# Sales from top x
x = 100
largestX = stock_summary.nlargest(x, 'sales_total$k')
largestX_value = (largestX['sales_total$k'].sum()).round(2)
total_sales = (stock_summary['sales_total$k'].sum()).round(2)
percentage = ((largestX_value / total_sales) * 100).round(2)
total_products = len(stock_df)
print(f'Total sales value $k: {total_sales}, from top {x}: {largestX_value}, approx {perc
print(f'Total number of products: {total_products}')

# Visualise the Sales Value ranges
value_bins = [0, 0.05, 0.1, 0.2, 0.5, 1, 5, 500]
value_bins_labels = ['< 0.05', '0.05 to 0.1', '0.1 to 0.2', '0.2 to 0.5', '0.5 to 1', '1
stock_summary['sales_groups'] = pd.cut(stock_summary['sales_total$k'], bins=value_bins,
                        labels=value_bins_labels, right=False)

fig, ax = plt.subplots(figsize =(10, 5))
sns.histplot(stock_summary['sales_groups'], bins=value_bins, kde=False, color = 'green')
plt.xticks([0, 1, 2, 3, 4, 5, 6])
plt.title('Distribution of Sales by Products')
plt.xlabel('Sales Value - $k')
plt.ylabel('Number of Products')
plt.show()
```

```
Sales with no sale price: 95
```



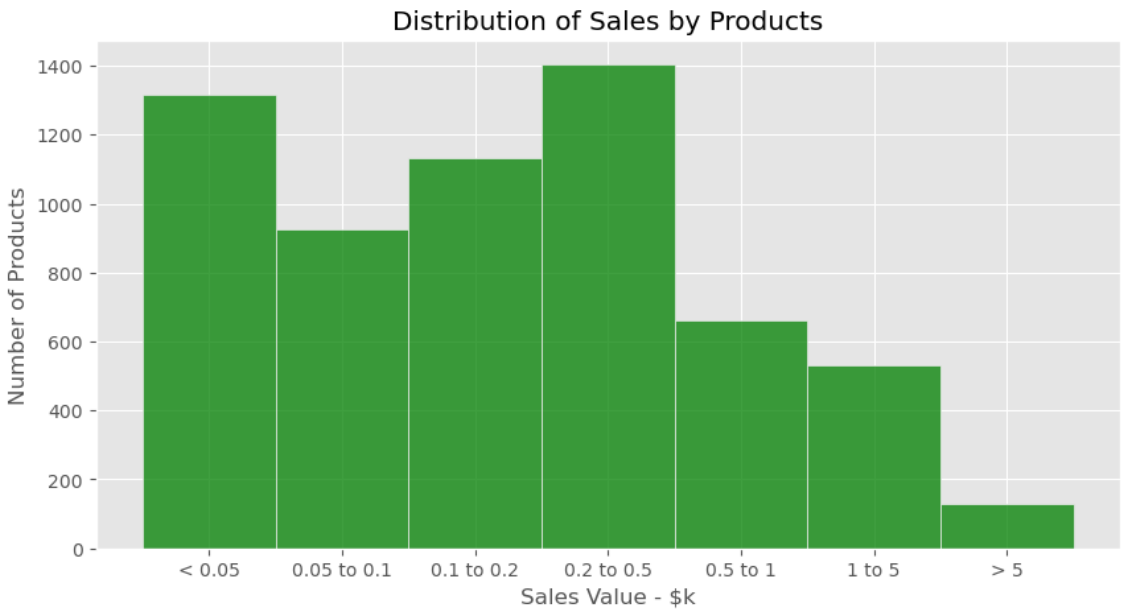
Top Five Products by Sales Value

	stock_name	sales_price	uniq_cust_count	sales_
36354	GE Profile™ 30” Built-In Double Wall Oven Self...	4023.510	20	20
19259	LG ENERGY STAR® 30 cu. ft. Super Capacity 3-Do...	4356.100	14	14
35750	Samsung ENERGY STAR® 26 cu. ft. 3-Door French ...	2270.705	22	22
36007	GE® 4.8 DOE Cu. ft. Capacity RightHeight™ Desi...	1412.880	33	33
10789	Samsung ENERGY STAR® 23 cu. ft. 4-Door French ...	3141.220	14	14

Bottom Five Products by Sales Value

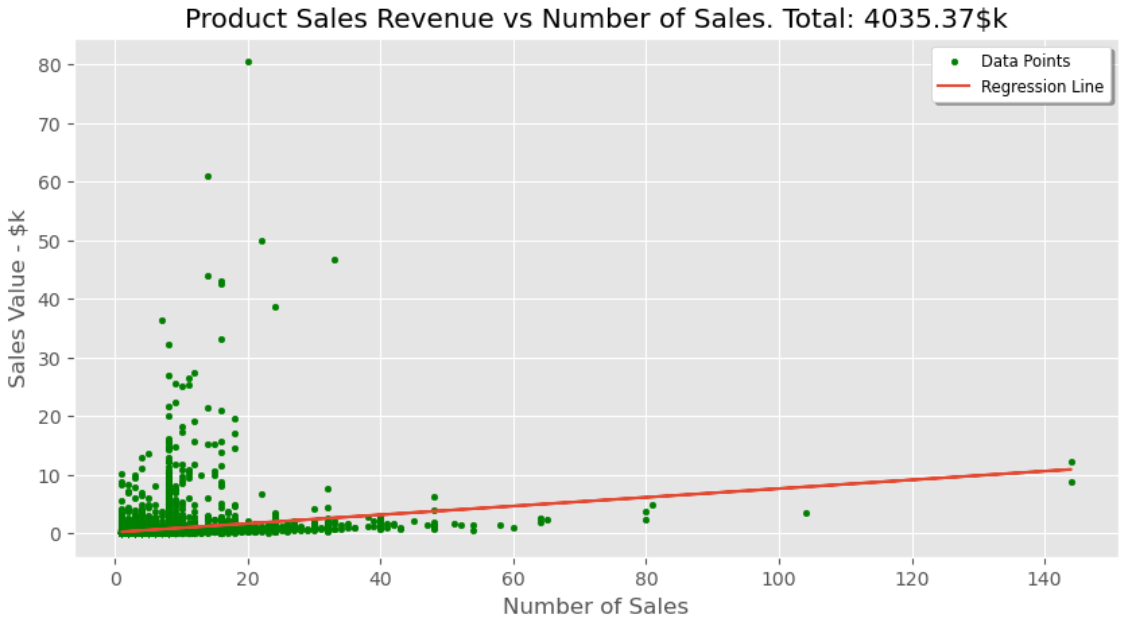
	stock_name	sales_price	uniq_cust_count	sales_count
6848	Ambrielle® Bonded Hipster Panties	3.61	1	1
15587	Marie Meili Mila Hipster Panties	4.82	1	1
24296	Disney Collection Fauna Mini Plush	4.83	1	1
38558	St. Eve Striped Hipster Panties	4.82	1	1
13482	HS by Happy Socks™ Mens Striped Crew Socks	4.97	1	1

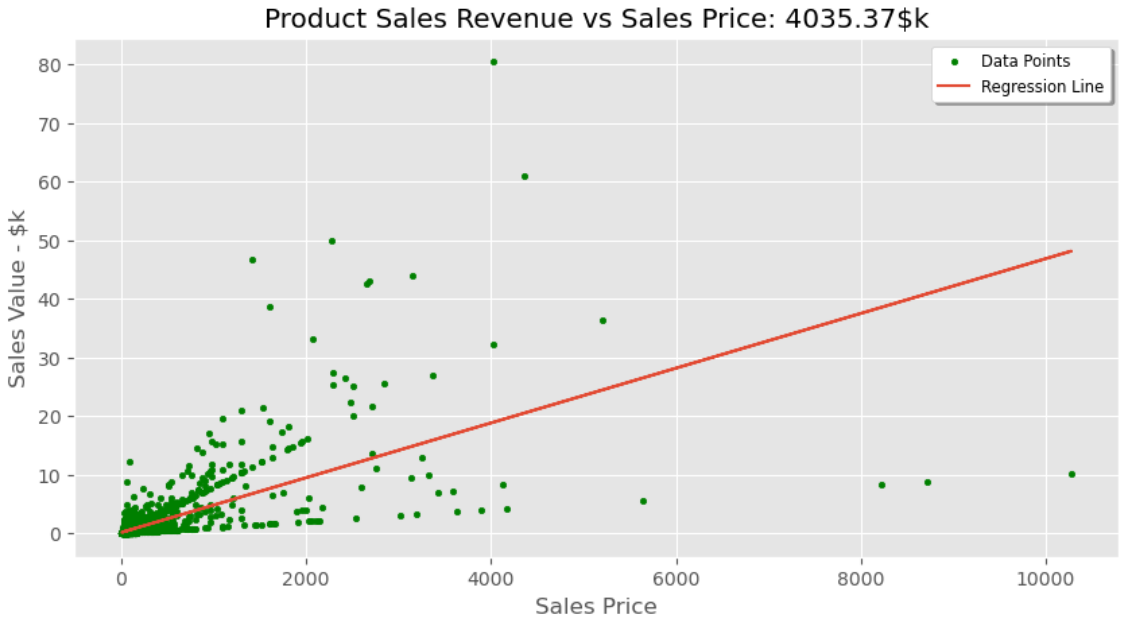
```
Total sales value $k: 4035.37, from top 100: 1635.13, approx 40.52%
Total number of products: 6110
```

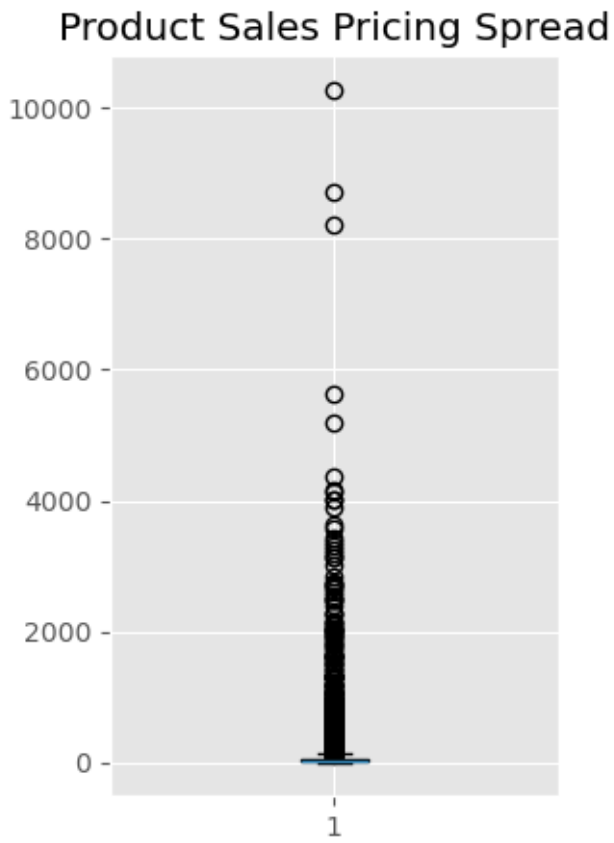


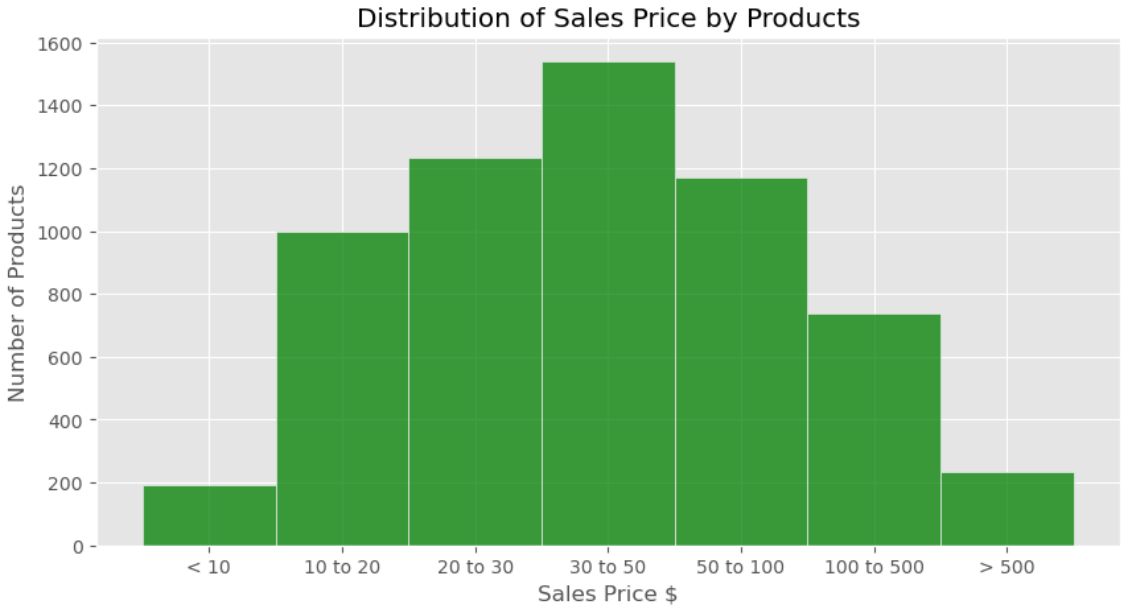

```
# Examine Product Pricing

# Identify where high sales value comes from?
# Is it sales volume or pricing

# Compare sales revenue with number of sales
x = stock_summary['sales_count']
y = stock_summary['sales_total$k']

slope, intercept, r, p, stderr = scipy.stats.linregress(x, y)
fig, ax = plt.subplots(figsize =(10, 5))

ax.set_title(f'Product Sales Revenue vs Number of Sales. Total: {total_sales}$k')
ax.set_xlabel('Number of Sales')
ax.set_ylabel('Sales Value - $k')
ax.scatter(x, y, label= 'Data Points', color = 'g', s = 10)
ax.plot(x, intercept + slope * x, label='Regression Line')
ax.legend(fontsize='small', loc='upper right', shadow=True, facecolor='white')
plt.show()

# Compare sales revenue with sales price
x = stock_summary['sales_price']
y = stock_summary['sales_total$k']

slope, intercept, r, p, stderr = scipy.stats.linregress(x, y)
fig, ax = plt.subplots(figsize =(10, 5))

ax.set_title(f'Product Sales Revenue vs Sales Price: {total_sales}$k')
ax.set_xlabel('Sales Price')
ax.set_ylabel('Sales Value - $k')
ax.scatter(x, y, label= 'Data Points', color = 'g', s = 10)
ax.plot(x, intercept + slope * x, label='Regression Line')
ax.legend(fontsize='small', loc='upper right', shadow=True, facecolor='white')
```

```
plt.show()

# Look at the spread of pricing
fig,ax0 = plt.subplots(figsize =(3, 5))
ax0.set_title('Product Sales Pricing Spread')
box0 = ax0.boxplot(stock_summary['sales_price'], patch_artist=True)
for patch in box0['boxes']:
    patch.set(facecolor='lightblue')
plt.show()

# Look at the number of products across price bands
value_bins = [0, 10, 20, 30, 50, 100, 500, 500000]
value_bins_labels = ['< 10', '10 to 20', '20 to 30', '30 to 50', '50 to 100', '100 to 500
stock_summary['sales_price_bands'] = pd.cut(stock_summary['sales_price'], bins=value_bins
                      labels=value_bins_labels, right=False)

fig, ax = plt.subplots(figsize =(10, 5))
sns.histplot(stock_summary['sales_price_bands'], bins=value_bins, kde=False, color = 'gre
plt.xticks([0, 1, 2, 3, 4, 5, 6])
plt.title('Distribution of Sales Price by Products')
plt.xlabel('Sales Price $')
plt.ylabel('Number of Products')
plt.show()


# Revenue by price band
bands_summary = stock_summary[['sales_price_bands']].drop_duplicates()
groups = stock_summary.groupby('sales_price_bands')
bands_summary['sales_total$k'] = groups['sales_total$k'].transform(sum)

fig, ax = plt.subplots(figsize =(10, 5))
ax.set_title(f'Sales Revenue by Sales Price Band')
ax.set_xlabel('Price Bands - $')
ax.set_ylabel('Sales Revenue - $k')
bars = ax.bar(bands_summary['sales_price_bands'], bands_summary['sales_total$k'], color='
plt.show()
```

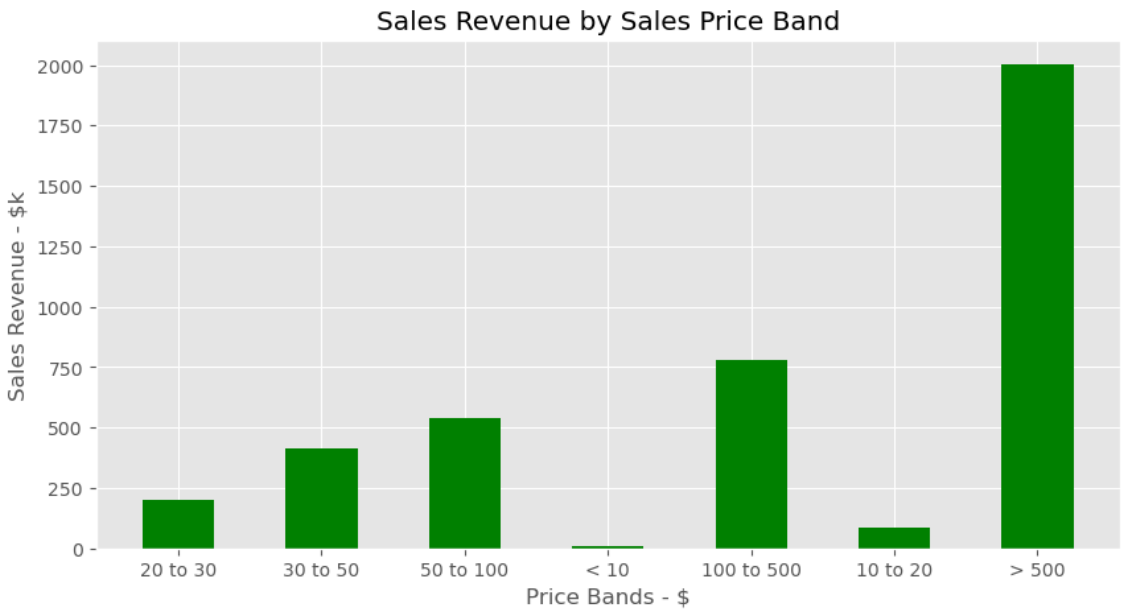

Product Sales Revenue vs Number of Sales. Total: 4035.37$k
Data Points
Regression Line
Sales Value - $k
Number of Sales
0
10
20
30
40
50
60
70
80
0
20
40
60
80
100
120
140


Product Sales Revenue vs Sales Price: 4035.37$k
Data Points
Regression Line
Sales Value - $k
Sales Price
0
10
20
30
40
50
60
70
80
0
2000
4000
6000
8000
10000






```
/var/folders/_3/08zshp4n3z1_glk6l5vgyqz40000gn/T/ipykernel_87432/3627445658.py:61: Future
  groups = stock_summary.groupby('sales_price_bands')
/var/folders/_3/08zshp4n3z1_glk6l5vgyqz40000gn/T/ipykernel_87432/3627445658.py:62: Future
  bands_summary['sales_total$k'] = groups['sales_total$k'].transform(sum)
```



4.3 Geographic Impact

The geographic distribution of sales across the US was examined. The results were very surprising and could either be due to very poor quality data and data collection or very unusual sales patterns.

Customer Distribution

The number of customers across states is very evenly spread, the interquartile range is approximately 80 to 95; this is surprising given the population range across all states is 300 to nearly 39 million.

Sales Distribution

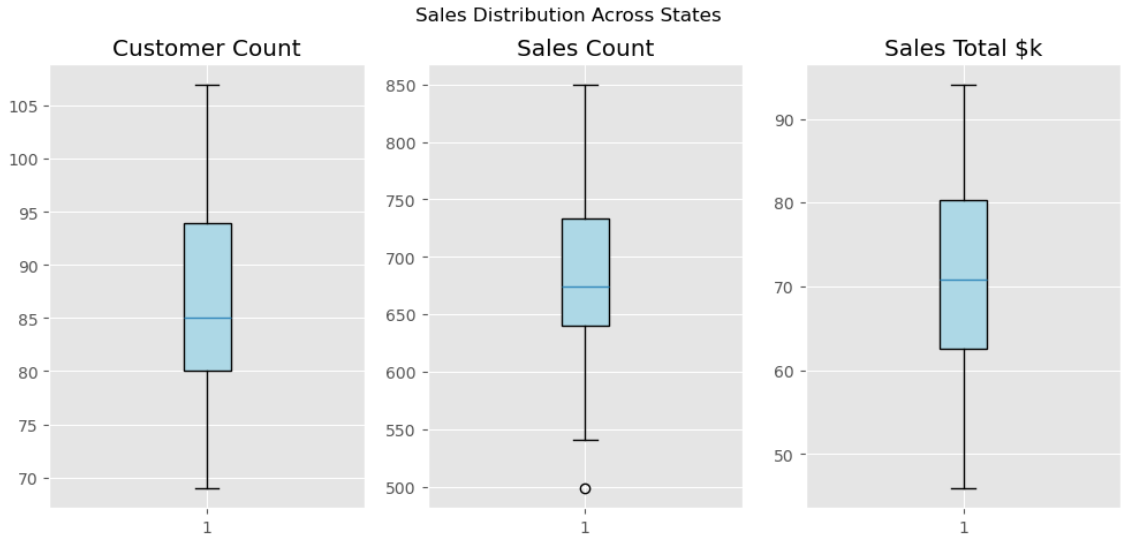
The sales data was analysed in a variety of ways (box plot, scatter plot with regression line, top and bottom 5, a choropleth map) but all supported the following conclusions:

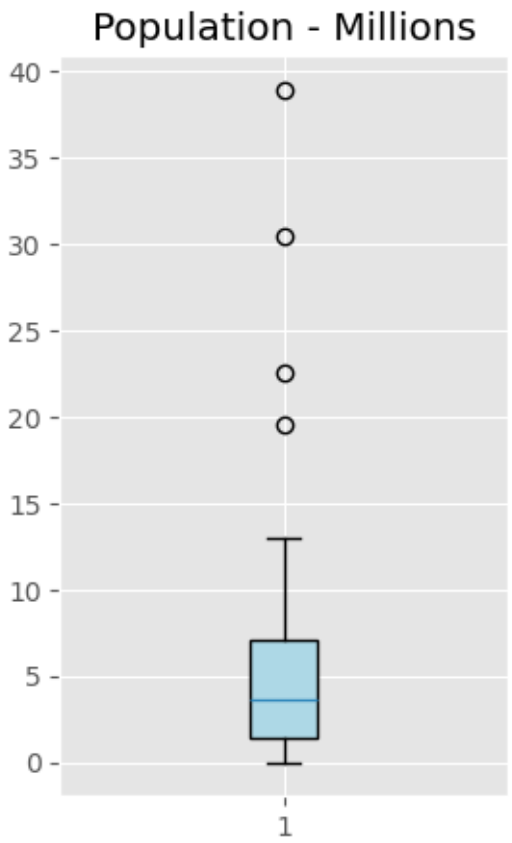
- The spread of the total sales count and the sales $ value is very small (when the population distribution is considered)
- For example the 4th highest sales are in the US Minor Outlying Islands which has a population of 300 but spending as much as Alabama which has a population of over 5 million
- And, the largest state, California, with a population of nearly 39 million has the lowest sales total

This is in contrast to the number of physical store per state which do correlate well with population size. So it does seem likely that there is a problem with the quality of the sales data collected. Before making any business recommendations, the data collection pipeline and sources should be investigated; or worst case the operational information systems that provided the source data.

```
 # Examine key customer and sales data by state

import pandas as pd
import matplotlib.pyplot as plt
plt.style.use('ggplot')

# Create customer sales
cust_sales = pd.merge(customer_reviews_df, customers_df, on='customer_id')
cust_sales = pd.merge(cust_sales, sales_df, on='uniq_id')

# Aggregate customer sales counts and total value
cust_summary = cust_sales[['customer_id', 'state_ISO']].drop_duplicates()
groups = cust_sales.groupby('customer_id')
cust_summary['sales_count'] = groups['uniq_id'].transform('count')
cust_summary['sales_total$k'] = groups['sale_price'].transform('sum')
cust_summary['sales_total$k'] = (cust_summary['sales_total$k'] / 1000).round(2)

# Aggregate the customer counts and sales totals by state
states_working = states_df.copy()
states_working['customer_count'] = customers_df.groupby('state_ISO')['state_ISO'].transfo
states_working.reset_index()
states_working['sales_count'] = cust_summary.groupby('state_ISO')['sales_count'].sum().re
states_working['sales_total$k'] = cust_summary.groupby('state_ISO')['sales_total$k'].sum(

# Some analysis of these figures geographical
# Too many states for a meaningful bar plot for each states ...

# Distribution of sales across states, using box plots
fig, (ax0, ax1, ax2) = plt.subplots(1, 3, figsize =(12, 5))
fig.suptitle('Sales Distribution Across States')

ax0.set_title('Customer Count')
ax1.set_title('Sales Count')
ax2.set_title('Sales Total $k')

box0 = ax0.boxplot(states_working['customer_count'], patch_artist=True)
for patch in box0['boxes']:
    patch.set(facecolor='lightblue')
box1 = ax1.boxplot(states_working['sales_count'], patch_artist=True)
for patch in box1['boxes']:
    patch.set(facecolor='lightblue')
box2 = ax2.boxplot(states_working['sales_total$k'], patch_artist=True)
for patch in box2['boxes']:
    patch.set(facecolor='lightblue')
plt.show()

# As comparison, the population distribution
fig,ax0 = plt.subplots(figsize =(3, 5))
ax0.set_title('Population - Millions')
box0 = ax0.boxplot(states_working['population']/1000000, patch_artist=True)
for patch in box0['boxes']:
```

```
    patch.set(facecolor='lightblue')
plt.show()

# Examine top and bottom 5 states by sales value
top5 = states_working.nlargest(5, 'sales_total$k')
bottom5 = states_working.nsmallest(5, 'sales_total$k')

print('Top Five States by Sales Value')
display(top5[['state_name', 'population', 'customer_count', 'sales_count', 'sales_total$k
print('Bottom Five States by Sales Value')
display(bottom5[['state_name', 'population', 'customer_count', 'sales_count', 'sales_tota
```





Top Five States by Sales Value

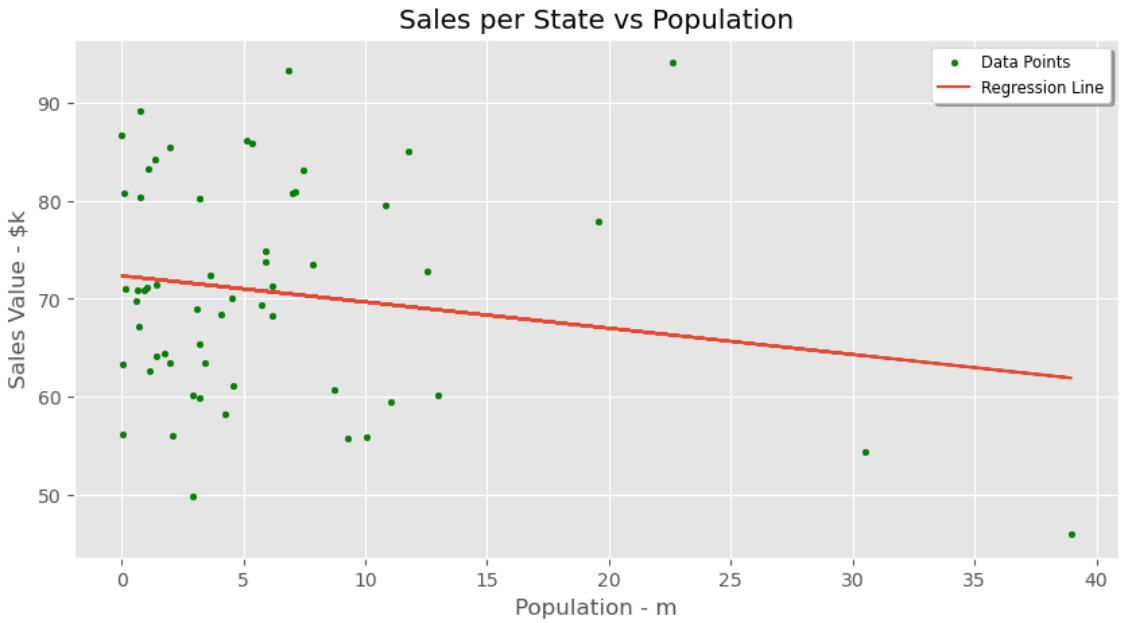
	state_name	population	customer_count	sales_count	sales_total$k
9	Florida	22610726	90	838	94.10
14	Indiana	6862199	83	715	93.25
34	North Dakota	783926	79	802	89.19
53	US Minor Outlying Islands	300	79	716	86.74
0	Alabama	5108468	96	774	86.14

Bottom Five States by Sales Value

	state_name	population	customer_count	sales_count	sales_total$k
4	California	38965193	79	546	45.95
16	Kansas	2940546	79	541	49.78
43	Texas	30503301	76	705	54.34
30	New Jersey	9290841	102	499	55.79
22	Michigan	10037261	69	630	55.90

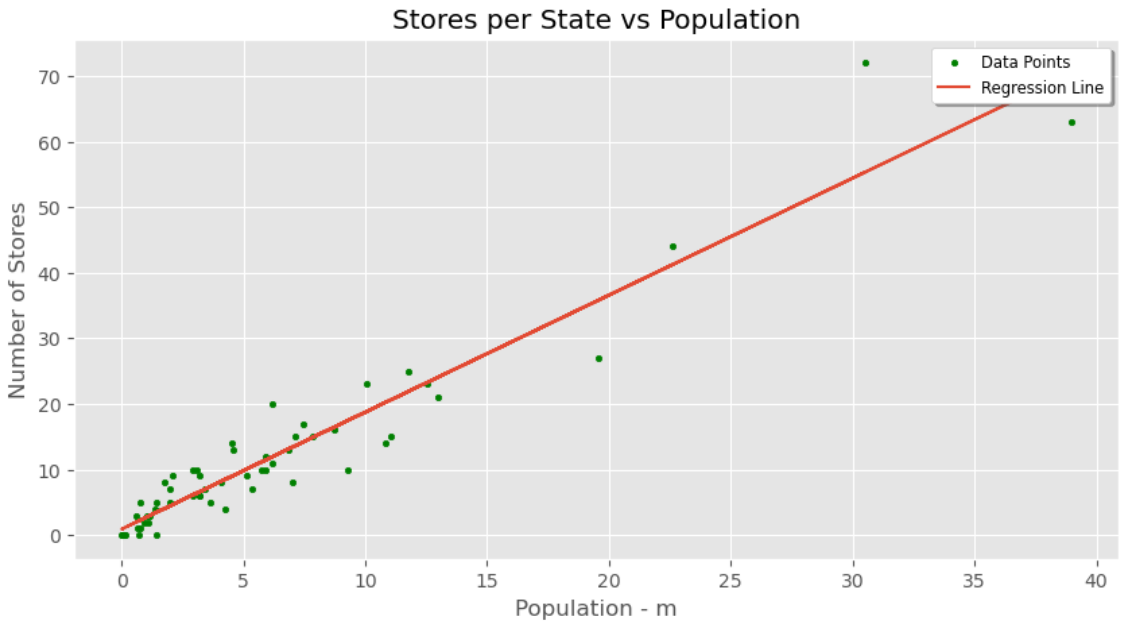
```
# Compare the sales value per state against the population
# Scatter plot and regression line

import scipy.stats
```

```
x = states_working['population'] / 1000000
y = states_working['sales_total$k']

# Calculate the regression line
slope, intercept, r, p, stderr = scipy.stats.linregress(x, y)

# Scatter plot with fir line
fig, ax = plt.subplots(figsize =(10, 5))

ax.set_title('Sales per State vs Population')
ax.set_xlabel('Population - m')
ax.set_ylabel('Sales Value - $k')

ax.scatter(x, y, label= 'Data Points', color = 'g', s = 10)
ax.plot(x, intercept + slope * x, label='Regression Line')
ax.legend(fontsize='small', loc='upper right', shadow=True, facecolor='white')

plt.show()
```




```
 # Sanity check of physical stores per state against the population
# Scatter plot and regression line

import scipy.stats

x = states_working['population'] / 1000000
y = states_working['stores_total']

# Calculate the regression line
slope, intercept, r, p, stderr = scipy.stats.linregress(x, y)

# Scatter plot with fir line
fig, ax = plt.subplots(figsize =(10, 5))
```

```
ax.set_title('Stores per State vs Population')
ax.set_xlabel('Population - m')
ax.set_ylabel('Number of Stores')

ax.scatter(x, y, label= 'Data Points', color = 'g', s = 10)
ax.plot(x, intercept + slope * x, label='Regression Line')
ax.legend(fontsize='small', loc='upper right', shadow=True, facecolor='white')

plt.show()
```




```
# Map plot of sales totals per state
import plotly.express as px

fig = px.choropleth(states_working, locations='state_ISO', locationmode='USA-states', col
        color_continuous_scale='Blues', scope='usa', title='Sales Value $k by State')

fig.show()
```

```
Unable to display output for mime type(s): application/vnd.plotly.v1+json
```

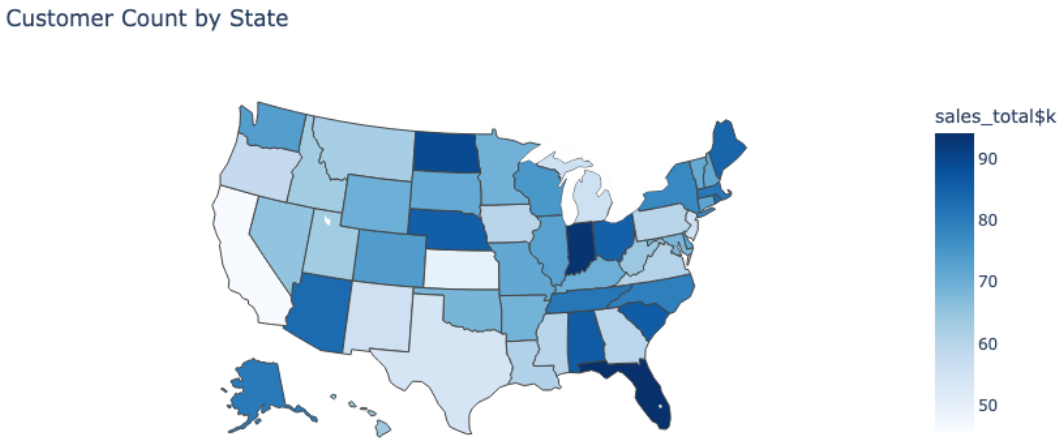


Figure 4.1: Map

5 Conclusions

5.1 Data Quality - Operational Improvements

During the data validation and preparation stage it quickly became evident that the overall quality of the provided data is poor. The CSV format data was rejected in preference to the JSON format sources because the CSV files had more missing data and were not consistent with the JSON data. It was assumed that the CSV files are the result of a first attempt to extract data from the JSON files.

More detailed examination of the JSON data revealed frequent gaps and inconsistencies. For example:

- Customer Reviews: Out of 39,000 reviews, 16,000 (40%) were duplicated across multiple customers. This could be a result of an error during extraction form operational systems, hopefully it is not a reflection of the production data in operational systems or even an attempt to create misleading reviews. Also score rankings were inconsistent, potentially because customers do not know if the review score rank of 1 to 5 means poor to good or is it good to poor. Sentiment analysis of the review text supported this observation.
- Customers: The data of birth for customers appears to be artificially generated and of limited use for demographic analysis. This could be deliberate obfuscation and hopefully not a reflection of the customer data held in source operational systems.
- Sales & Stock: Approximately 2% to 3% of data is inconsistent or incomplete, for example stock without sales prices or descriptions.

Due to the above, more time was spent on creating and documenting processes (ie Jupyter Notebooks) to make the validation and preparation stage easily repeatable when better quality data is provided. The reasons for the many data quality issues need to be understood so that the data extraction can be repeated more successfully. If the root cause is extraction then this can be easily addressed and appropriate validation checks put in place. If the causes are due to issues in the source operational systems then these need to be addressed as part of JCP's planned overhaul of its customer and stock systems.

With better quality data, the data collation can be repeated and then additional data analysis steps developed

5.2 Business Observations

As stated earlier, the data analysis completed is limited due to the poor quality of data available. All conclusions must be heavily caveated and probably not actioned on until the exercise can be repeated.

5.2.1 Product & Price Range

JCP has a very large product range, with over 6,000 stock items and with a wide range of prices from $3.61 up to $4,023.51. In these broad ranges, 40% of the total $4m sales revenue is generated by only 100 products. In terms of price, 50% of the sales revenue comes from products over $500 each, conversely 1,200 products are priced below $20 and generate less than $150k revenue. The very high price but low volume products do not make a big contribution and similarly the very high volume products do not make a big contribution to the overall sales revenue. The sweet spot appears to be products in the $20 to $4,500 price range.

The product range should be reviewed to confirm if rationalistion could improve overall sales profit by ceasing the sale of the very low price, higher volume, products and also very high price low volume products. Obviously the profit margin of individual products needs to be known for that analysis to be completed.

5.2.2 Geographic Impact

The distribution of the number of customers across all US States was extremely even, ranging from 80 to 95 customers per state, in comparison the total population across states (and territories) ranges from 300 to 39 million people. This could be yet another data quality issue or a very unusual geographic spread. Similarly the spread of sales revenue is unusual, for example the US Minor Outlying Islands has a population of 300 but spends as much as Alabama with a population of 5 million. And the largest state, California with a population of 39 million has the lowest sales revenue.

The first action is to review the quality of the sales data to see if these ranges are accurate and if not then what is the root cause of the data collection issues. If the sales data is accurate then a very detailed examination of sales patterns needs to be completed to understand why bigger revenue is not obtained from states such as California.

References

Microsoft (2023) https://devblogs.microsoft.com/python/data-wrangler-release/

Ncr, P.C. et al. (1999) 'CRISP-DM 1.0'

Hotz (2024) https://www.datascience-pm.com/crisp-dm-2/

Modern Retail (2023) https://www.modernretail.co/operations/jcpenney-is-the-latest-department-store-to-announce-a-major-turnaround-plan/

Appendix - Detailed Steps

Detailed Steps for Data Collation, Exploration & Preparation

This appendix details the sequence and outputs of the process of collating the source data, validating and then preparing it for subsequent analysis. The main report describes the results of this process but without all the detailed Python code, in this appendix all the detail is included.

```
# General setup and imports used throughout the Jupyter Notebook
#
# Libraries For file handling and dataframes
import os
import json
from IPython.display import display
import pandas as pd

# Libraries fo plots
import matplotlib.pyplot as plt
plt.style.use('ggplot')

# Libraries for maths etc
import math
import nltk
nltk.download('vader_lexicon')

# Libraries for URL requests
import requests
from PIL import Image
from io import BytesIO

# Variables used throughout the notebook
DATA_DIRECTORY = 'JCPenney_Data_Original'  # Designated data folder within the current wo
AUGMENTED_DATA = 'Data_Additional'         # Additional data sources

# Simple utility functions to obtain and summarise key elements of a provided dataframe
#
def print_file_summary(data_frame):
    # Create a temporary df and ensure no lists remain, so that unique items can be ident
    temp_df = data_frame.copy()
    temp_df = temp_df.map(lambda cell: str(cell) if isinstance(cell, list) else cell)

    # Calculate some
    summary_of_df = pd.DataFrame({'Count': data_frame.count(),
                                  'Missing': data_frame.isnull().sum(), 'Empty': 0,
```

```
                                    'Unique': temp_df.nunique(),
                                    'Type': data_frame.dtypes,
                                    'String': 0, 'Int': 0, 'Float': 0, 'List': 0
                                    })
    summary_of_df['Empty'] = (data_frame == '').sum()
    summary_of_df['String'] = data_frame.map(lambda cell: isinstance(cell, str)).sum()
    summary_of_df['Int'] = data_frame.map(lambda cell: isinstance(cell, int)).sum()
    summary_of_df['Float'] = data_frame.map(lambda cell: isinstance(cell, float)).sum()
    summary_of_df['List'] = data_frame.map(lambda cell: isinstance(cell, list)).sum()

    display(summary_of_df)

def print_full_summary(title, data_frame):
    # Print the summary and head for the given dataframe
    # Used frequently in the notebook so created a function to reduce repetition
    print(title)
    print_file_summary(data_frame)
    print(f'First 3 Rows')
    display(data_frame.head(3))

# Simple function to request a URL and return success
def check_url(with_url):
    try:
        response = requests.get(with_url)
        response.raise_for_status()
        print(response.status_code)
        #img = Image.open(BytesIO(response.content))
        #img.show()
        return True
    except requests.exceptions.RequestException as e:
        print(f"An error occurred: {e}")
        return False
```

```
[nltk_data] Downloading package vader_lexicon to
[nltk_data]     /Users/stuartgow/nltk_data...
[nltk_data]   Package vader_lexicon is already up-to-date!
```

Provided Data Sources & Content

Load JSON Data Files

```
# Load the JSON product file and examine the format and content
# NB: Use pandas json load to directly create a dataframe

# Products file source
file_name = 'jcpenney_products.json'
file_path = os.path.join(os.getcwd(), DATA_DIRECTORY, file_name)
if not os.path.isfile(file_path):
```

```
    raise Exception(f'File not found: {file_path}')

# File load into a Pandas dataframe, retained and not amended
source_jcp_products_df = pd.read_json(file_path, lines=True)

# Initial look at the file and data fields
print(f'File Summary for: {file_name}')
print_file_summary(source_jcp_products_df)
print(f'First 3 Rows')
display(source_jcp_products_df.head(3))


# Tidy up
del file_name, file_path
```

```
File Summary for: jcpenney_products.json
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	7982	0	0	7982	object	7982	0	0	0
sku	7982	0	67	6044	object	7982	0	0	0
name_title	7982	0	0	6002	object	7982	0	0	0
description	7982	0	543	5620	object	7982	0	0	0
list_price	7982	0	2166	1037	object	7982	0	0	0
sale_price	7982	0	18	2063	object	7982	0	0	0
category	7982	0	636	1169	object	7982	0	0	0
category_tree	7982	0	636	1997	object	7982	0	0	0
average_product_rating	7982	0	0	153	float64	0	0	7982	0
product_url	7982	0	0	7982	object	7982	0	0	0
product_image_urls	7982	0	157	6519	object	7982	0	0	0
brand	7982	0	0	721	object	7982	0	0	0
total_number_reviews	7982	0	0	22	int64	0	7982	0	0
Reviews	7982	0	0	7982	object	0	0	0	7982
Bought With	7982	0	0	7982	object	0	0	0	7982

```
First 3 Rows
```

	uniq_id	sku	name_title
0	b6c0b6bea69c722939585baeac73c13d	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant
1	93e5272c51d8cce02597e3ce67b7ad0a	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant
2	013e320f2f2ec0cf5b3ff5418d688528	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant

```
# Load the JSON reviewers file and examine the format and content
# NB: Use pandas json load to directly create a dataframe

# Reviewers file source
file_name = 'jcpenney_reviewers.json'
```

```
file_path = os.path.join(os.getcwd(), DATA_DIRECTORY, file_name)
if not os.path.isfile(file_path):
    raise Exception(f'File not found: {file_path}')

# File load into a Pandas dataframe, retained and not amended
source_jcp_reviewers_df = pd.read_json(file_path, lines=True)

# Initial look at the file and data fields
print(f'File Summary for: {file_name}')
print_file_summary(source_jcp_reviewers_df)
print(f'First 3 Rows')
display(source_jcp_reviewers_df.head(3))

# Tidy up
del file_name, file_path
```

```
File Summary for: jcpenney_reviewers.json
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
Username	5000	0	0	4999	object	5000	0	0	0
DOB	5000	0	0	52	object	5000	0	0	0
State	5000	0	0	57	object	5000	0	0	0
Reviewed	5000	0	0	4030	object	0	0	0	5000

```
First 3 Rows
```

	Username	DOB	State	Reviewed
0	bkpn1412	31.07.1983	Oregon	[cea76118f6a9110a893de2b7654319c0]
1	gqjs4414	27.07.1998	Massachusetts	[fa04fe6c0dd5189f54fe600838da43d3]
2	eehe1434	08.08.1950	Idaho	[]

Working Data Structure - Validation & Augmentation

States & Territories

```
# Establish a reference list of states/territories with additional data to augment

# Load the states .csv file, exit if do not exist or are invalid
file_path = os.path.join(os.getcwd(), AUGMENTED_DATA, 'JCP_Stores_State_Collated.csv')
if not os.path.isfile(file_path):
    raise Exception(f"File not found: {file_path}")
states_df = pd.read_csv(file_path)

# Initial look at the file and data fields
```

```
print(f'Summary of States - CSV')
print_file_summary(states_df)
print(f'First 3 rows')
display(states_df.head(3))

# Rename column names & set the index on ISO
states_df = states_df.rename(columns={'State or Territory': 'territory_flag',
                                      'State_ISO': 'state_ISO', 'State_Name': 'stat
                                      'Population_2023': 'population',
                                      'Store_Count': 'stores_total'})
#states_df.set_index(keys='state_ISO', inplace=True)

# Convert population to int
states_df['population'] = states_df['population'].str.replace(',', '').astype(int)

# Final look at the file and data fields
print(f'Summary of States - CSV')
print_file_summary(states_df)
print(f'First 3 rows')
display(states_df.head(3))

# Tidy up
del file_path

# Provide a simple unique state lookup of ISO for a given name
def get_state(state_name):
    matched_state = states_df.loc[states_df['state_name'] == state_name]
    if len(matched_state) == 1:
        return matched_state.iloc[0]
    else:
        return None
```

```
Summary of States - CSV
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
State or Territory	57	0	0	2	object	57	0	0	0
State_ISO	57	0	0	57	object	57	0	0	0
State_Name	57	0	0	57	object	57	0	0	0
Population_2023	57	0	0	57	object	57	0	0	0
Store_Count	57	0	0	26	int64	0	57	0	0

```
First 3 rows
```

	State or Territory	State_ISO	State_Name	Population_2023	Store_Count
0	State	AL	Alabama	5,108,468	9
1	State	AK	Alaska	733,406	1
2	State	AZ	Arizona	7,431,344	17

```
Summary of States - CSV
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
territory_flag	57	0	0	2	object	57	0	0	0
state_ISO	57	0	0	57	object	57	0	0	0
state_name	57	0	0	57	object	57	0	0	0
population	57	0	0	57	int64	0	57	0	0
stores_total	57	0	0	26	int64	0	57	0	0

```
First 3 rows
```

	territory_flag	state_ISO	state_name	population	stores_total
0	State	AL	Alabama	5108468	9
1	State	AK	Alaska	733406	1
2	State	AZ	Arizona	7431344	17

Customers

```
# Establish a customer details dataframe

# Create a new dataframe for all customer reviews
customers_df = source_jcp_reviewers_df.copy()

# Rename customer column names and validate content for each
customers_df = customers_df.rename(columns={'Username': 'customer_id',
                                            'State': 'state_name',
                                            'Reviewed': 'uniq_id_list'})

# Print the file and data fields
print(f'Summary for customers')
print_file_summary(customers_df)
print(f'First 3 rows - Renamed Columns')
display(customers_df.head(3))

# Identify duplicate customers
duplicates_flag = customers_df.duplicated(subset=['customer_id'], keep=False)
duplicated = customers_df[duplicates_flag]
print(f'Duplicated Customers:')
display(duplicated)

# Replace duplicates with new customer_id 'DUPnnnxxxxxx' to preserve
# Use itertuples as faster for larger datasets
dup_count = 0
for row in duplicated.itertuples():
    dup_count += 1
    new_id = 'DUP' + str(dup_count).zfill(3) + row.customer_id
```

```
    customers_df.at[row.Index, 'customer_id'] = new_id

# Double check no duplicates remain
duplicates_flag = customers_df.duplicated(subset=['customer_id'], keep=False)
duplicated = customers_df[duplicates_flag]
print(f'Double-Check No Remaining Duplicated Customers:')
display(duplicated)

# DOB convert to date format and examine the dates used
customers_df['DOB'] = pd.to_datetime(customers_df['DOB'], dayfirst=True, errors='coerce')
dates_grouped = customers_df.groupby('DOB').size().reset_index(name='counts')
date_range = pd.DataFrame({'Oldest': [dates_grouped['DOB'].min()],
                           'Youngest': [dates_grouped['DOB'].max()],
                           'Unique Dates': [len(dates_grouped)]})
display(date_range)

# Remove year to examine the day / month distribution & plot this
converted_dates = customers_df['DOB'].dt.strftime('%m-%d').reset_index().groupby('DOB').s
fig, ax = plt.subplots(figsize =(10, 5))
ax.set_title(f'DOB month/day distribution. (Total: {len(converted_dates)})')
ax.set_xlabel('Date - MM-DD')
ax.set_ylabel('Count')
bars = ax.bar(converted_dates.index, converted_dates, color='green', width=0.5)
plt.show()

# States validation - lookup ISO codes, add to customer data and check for invalid matche
customers_df['state_ISO'] = customers_df['state_name'].apply(lambda x: get_state(x)['stat
unmatched_states = customers_df[customers_df['state_ISO'].isnull()]
print(f'Unmatched States:')
display(unmatched_states[['customer_id', 'state_name']])

# Names mismatch for US Virgin Islands and US Minor Outlying Islands
customers_df.replace('U.S. Virgin Islands', 'US Virgin Islands', inplace=True)
customers_df.replace('Minor Outlying Islands', 'US Minor Outlying Islands', inplace=True)

# Repeat the checks & drop state_name if all ISO populated
customers_df['state_ISO'] = customers_df['state_name'].apply(lambda x: get_state(x)['stat
unmatched_states = customers_df[customers_df['state_ISO'].isnull()]
print(f'Unmatched States:')
display(unmatched_states[['customer_id', 'state_name']])

# Drop the state name, rely on the ISO code and states lookup
if len(unmatched_states) != 0:
    raise Exception(f'Cannot match: {len(unmatched_states)} states')
customers_df = customers_df.drop('state_name', axis=1)

# Visual check on state details - PRint not included in report due to size
states = customers_df.groupby('state_ISO').size().reset_index(name='counts')
print(f'Customers by State:')
#display(states)
```

```
# Tidy up
del duplicates_flag, duplicated, dup_count, new_id, row
del dates_grouped, converted_dates, date_range
del unmatched_states, states
```

Summary for customers

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
customer_id	5000	0	0	4999	object	5000	0	0	0
DOB	5000	0	0	52	object	5000	0	0	0
state_name	5000	0	0	57	object	5000	0	0	0
uniq_id_list	5000	0	0	4030	object	0	0	0	5000

First 3 rows - Renamed Columns

	customer_id	DOB	state_name	uniq_id_list
0	bkpn1412	31.07.1983	Oregon	[cea76118f6a9110a893de2b7654319c0]
1	gqjs4414	27.07.1998	Massachusetts	[fa04fe6c0dd5189f54fe600838da43d3]
2	eehe1434	08.08.1950	Idaho	[]

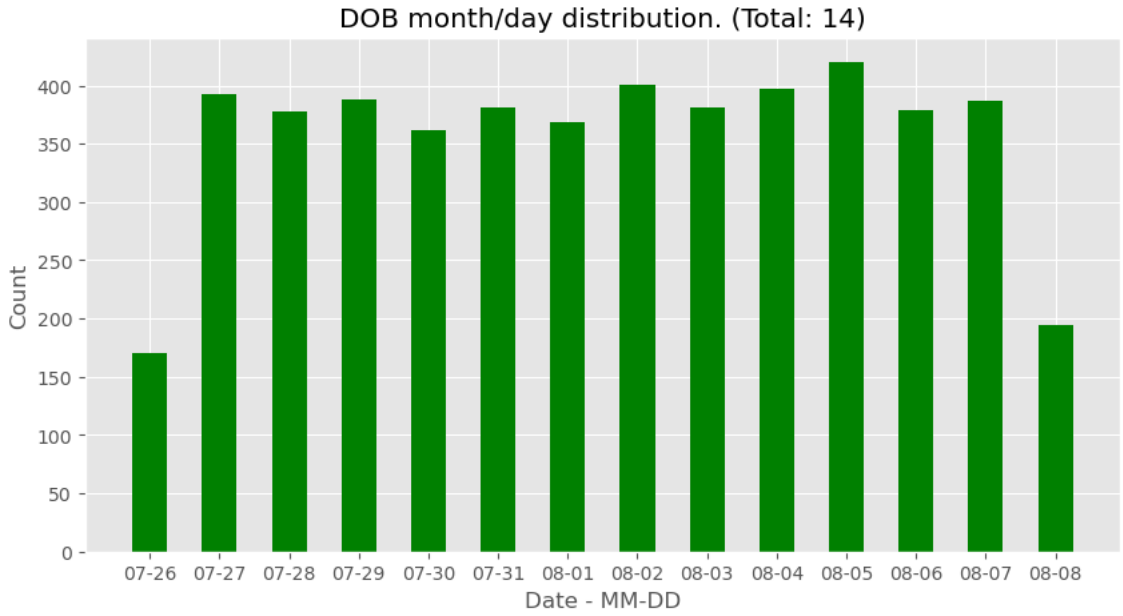
Duplicated Customers:

	customer_id	DOB	state_name	uniq_id_list
731	dqft3311	28.07.1995	Tennessee	[5f280fb338485cfc30678998a42f0a55]
2619	dqft3311	03.08.1969	New Mexico	[571b86d307f94e9e8d7919b551c6bb52]

Double-Check No Remaining Duplicated Customers:

customer_id	DOB	state_name	uniq_id_list

	Oldest	Youngest	Unique Dates
0	1950-08-08	2001-07-26	52



Unmatched States:

	customer_id	state_name
29	wjfh4432	Minor Outlying Islands
104	ulkz1412	Minor Outlying Islands
106	bsqg4331	Minor Outlying Islands
203	bbiv3413	Minor Outlying Islands
215	surt1311	U.S. Virgin Islands
...	...	...
4872	ypcn2342	U.S. Virgin Islands
4940	lric2324	U.S. Virgin Islands
4960	okun1224	Minor Outlying Islands
4970	kjgm1311	U.S. Virgin Islands
4976	gjed1211	U.S. Virgin Islands

Unmatched States:

customer_id	state_name

Customers by State:

Stock Details

```
# Establish a reference list of all product / stock details
# And also the initial draft of the sales dataframe for further preparation

# Create an initial new dataframe for all stock details
```

```
stock_df = source_jcp_products_df.copy()

# Print the file and data fields
print_full_summary('Summary For Stock/Sales/Product Data - Initial Look', stock_df)

# Flag all missing fields for easier checking and replacement
missing_flag = 'Missing'
stock_df.replace('', missing_flag, inplace=True)
stock_df.fillna(missing_flag, inplace=True)

# Count missing and check formats of SKU
sku_formats = {'pp500nnnnnn': r'^pp500\d{6}',
               '1xxxxxx': r'^1\w{6}',
               'enxnnnnnnnnnn': r'^en\D\d{10}',
               missing_flag: r'Missing'}
counts = {}
filtered = stock_df.copy()
for sku_format, regex_pattern in sku_formats.items():
    matched = stock_df[stock_df['sku'].str.contains(regex_pattern, na=False)]
    counts[sku_format] = len(matched)
    filtered = filtered[~filtered['sku'].isin(matched['sku'])]
print(f'Counts for SKU missing and format types + formats not matching')
display(counts)
display(filtered)

# Generate ids for missing SKU
# Use itertuples as faster for larger datasets
sku_count = 6000000000
missing_sku = stock_df[stock_df['sku'] == missing_flag]
for row in missing_sku.itertuples():
    sku_count += 1
    new_id = 'pp' + str(sku_count)
    stock_df.at[row.Index, 'sku'] = new_id
# Double-check all updated
missing_sku = stock_df[stock_df['sku'] == missing_flag]
display(missing_sku)

# Create an initial new dataframe for all sales details ready for later manipulation
sales_df = stock_df.copy()

# Drop non stock columns
columns_not_required = ['uniq_id', 'sale_price', 'category', 'category_tree', 'average_pr
                        'product_url',
                        'total_number_reviews', 'Reviews', 'Bought With']
stock_df.drop(columns=columns_not_required, inplace=True, errors='ignore')

# Rename the retained columns
# (nb all listed for documentation purposes)
stock_df = stock_df.rename(columns={'name_title': 'stock_name', 'description': 'descripti
                                    'list_price': 'list_price',
                                    'product_image_urls': 'stock_image_url',
```

```
                                    'brand': 'brand'})


# Remove duplicated sku ids rows
# Checking consistency of the other columns
sku_duplicated = stock_df.groupby('sku').filter(lambda sku: len(sku) > 1)
print(f'duplicated skus: {len(sku_duplicated) }')
#sku_groups_dup = sku_duplicated.groupby('sku')

# Iterate through all grouped sku ids and validate, select the individual column values
# And drop duplicated rows
sku_groups = stock_df.groupby('sku')
new_stock_df = stock_df.head(0).copy()

for sku, group in sku_groups:
    new_sku = stock_df.head(0).copy()
    new_sku.at[0, 'sku'] = sku
    # stock name - Just retain the first record
    new_sku.at[0, 'stock_name'] = group['stock_name'].iloc[0]
    # description - Keep the first non-blank
    non_empty = group[group['description'] != missing_flag]
    if(len(non_empty) != 0):
        new_sku.at[0, 'description'] = non_empty['description'].iloc[0]
    else:
        new_sku.at[0, 'description'] = 'No Description Available'
    # list_price
    non_empty = group[group['list_price'] != missing_flag]
    if(len(non_empty) != 0):
        most_frequent = non_empty['list_price'].value_counts().idxmax()
        new_sku.at[0, 'list_price'] = most_frequent
    else:
        new_sku.at[0, 'list_price'] = 0
    # stock_image_url
    non_empty = group[group['stock_image_url'] != missing_flag]
    if(len(non_empty) != 0):
        most_frequent = non_empty['stock_image_url'].value_counts().idxmax()
        new_sku.at[0, 'stock_image_url'] = most_frequent
    else:
        new_sku.at[0, 'stock_image_url'] = 'No URL Available'
    # brand
    non_empty = group[group['brand'] != missing_flag]
    if(len(non_empty) != 0):
        new_sku.at[0, 'brand'] = non_empty['brand'].iloc[0]
    else:
        new_sku.at[0, 'brand'] = 'No Details Available'

    # Add the single row to the temporary stock dataframe
    new_stock_df = pd.concat([new_stock_df, new_sku], ignore_index=True)

# Finally, copy the reduced stock list and set all fields to appropriate types
stock_df = new_stock_df.copy()
```

```
stock_df['list_price'] = pd.to_numeric(stock_df['list_price'], errors='coerce')

# Print the file and data fields
print_full_summary('Summary For Stock Data - After Cleaning', stock_df)

# Tidy up
del counts, filtered, matched, regex_pattern, sku_format, sku_formats, missing_flag, non_
del sku_count, missing_sku, row, new_id, most_frequent
del new_sku, new_stock_df, columns_not_required, sku, sku_duplicated, group, sku_groups
```

```
Summary For Stock/Sales/Product Data - Initial Look
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	7982	0	0	7982	object	7982	0	0	0
sku	7982	0	67	6044	object	7982	0	0	0
name_title	7982	0	0	6002	object	7982	0	0	0
description	7982	0	543	5620	object	7982	0	0	0
list_price	7982	0	2166	1037	object	7982	0	0	0
sale_price	7982	0	18	2063	object	7982	0	0	0
category	7982	0	636	1169	object	7982	0	0	0
category_tree	7982	0	636	1997	object	7982	0	0	0
average_product_rating	7982	0	0	153	float64	0	0	7982	0
product_url	7982	0	0	7982	object	7982	0	0	0
product_image_urls	7982	0	157	6519	object	7982	0	0	0
brand	7982	0	0	721	object	7982	0	0	0
total_number_reviews	7982	0	0	22	int64	0	7982	0	0
Reviews	7982	0	0	7982	object	0	0	0	7982
Bought With	7982	0	0	7982	object	0	0	0	7982

```
First 3 Rows
```

	uniq_id	sku	name_title
0	b6c0b6bea69c722939585baeac73c13d	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant
1	93e5272c51d8cce02597e3ce67b7ad0a	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant
2	013e320f2f2ec0cf5b3ff5418d688528	pp5006380337	Alfred Dunner® Essential Pull On Capri Pant

```
Counts for SKU missing and format types + formats not matching

{'pp500nnnnnn': 7505, '1xxxxxx': 394, 'enxnnnnnnnnnn': 13, 'Missing': 67}
```

	uniq_id	sku	name_title
2269	9fa199671d88a2a3cddd06a0dac02763	0903a80	KitchenAid® Artisan® 5-qt. Stand Mixer KS
3984	4875e80ad4e5d0d8970850046a4c8b8c	PP100000902	Alyx® Gauze Print Tank Top or Millennium
7884	6dcebbf40f3195554080edced28d401b	0903a80	KitchenAid® Artisan® 5-qt. Stand Mixer KS

uniq_id	sku	name_title	description	list_price	sale_price	category	category_tree	average_

```
duplicated skus: 3026
Summary For Stock Data - After Cleaning
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
sku	6110	0	0	6110	object	6110	0	0	0
stock_name	6110	0	0	5894	object	6110	0	0	0
description	6110	0	0	5596	object	6110	0	0	0
list_price	6110	0	0	990	float64	0	0	6110	0
stock_image_url	6110	0	0	5991	object	6110	0	0	0
brand	6110	0	0	721	object	6110	0	0	0

```
First 3 Rows
```

	sku	stock_name	description
0	0903a80	KitchenAid® Artisan® 5-qt. Stand Mixer KSM150PS	The mixer you've always dreamed
1	13cab12	JCPenney Home™ Saratoga Cut-to-Width Fringed B...	Saratoga cut--to-width blackout sh
2	13cab4b	JCPenney Home™ Saratoga Cut-to-Width Unfringed...	The Saratoga cut-to-width roller sl

```
# Check valid URLs

# Get a few random product URLs to test
stock_random = stock_df.sample(n=5)
for row in stock_random.itertuples():
    url = row.stock_image_url
    if check_url(url):
        print(f'URL: {url} is good.')
        break
    else:
        print(f'URL: {url} failed.')
```

```
200
URL: http://s7d9.scene7.com/is/image/JCPenney/DP0115201617053791M.TIF?hei=380&amp;wid=380
```

Sales

```
# Establish a list of all product sales

# Use the initial sales file creating during the stock file creation,
# and drop columns that were retained in the stock_df
columns_not_required = ['name_title', 'description', 'list_price', 'product_image_urls',
sales_df.drop(columns=columns_not_required, inplace=True, errors='ignore')
```

```
 # Print the file and data fields
print_full_summary('Summary For Product Data - Initial Look', sales_df)

# Rename the retained columns (nb all listed for documentation purposes)
sales_df = sales_df.rename(columns={'uniq_id': 'uniq_id',
                                    'sku': 'sku',
                                    'sale_price': 'sale_price',
                                    'category': 'category', 'category_tree': 'category_tr
                                    'average_product_rating': 'average_product_rating',
                                    'product_url': 'sales_product_url',
                                    'total_number_reviews': 'total_number_reviews',
                                    'Reviews': 'reviews_list', 'Bought With': 'bought_wit

# Flag all missing fields for easier checking and replacement
missing_flag = 'Missing'
sales_df.replace('', missing_flag, inplace=True)
sales_df.fillna(missing_flag, inplace=True)

# uniq_id - nb visual inspection shows no duplicate and no missing
# sku - validated during production of the sales file

# sale_price - several values missing and formatted as range numbers
# Clean up and convert to float
def convert_price(price):
    try:
        # Trap the values with a range
        if '-' in price:
            low, high = map(float, price.split('-'))
            averaged = (low + high) / 2
            return averaged
        if price == missing_flag:
            return 0.0
        return float(price)
    except:
        return 0.0
sales_df['sale_price'] = sales_df['sale_price'].apply(convert_price)

# category and category tree
missing_cat = len(sales_df[sales_df['category'] == missing_flag])
missing_cat_tree = len(sales_df[sales_df['category_tree'] == missing_flag])
print(f'Missing: Categories {missing_cat} Trees {missing_cat_tree}')

# product URL check
duplicates_count = sales_df.duplicated(subset=['sales_product_url'], keep=False).sum()
print(f'Duplicated URLs: {duplicates_count}')

# Create an initial new dataframe for customer reviews ready for later manipulation
working_customer_reviews_df = sales_df.copy()
columns_not_required = ['sku', 'sale_price', 'category', 'category_tree', 'sales_product_
working_customer_reviews_df.drop(columns=columns_not_required, inplace=True, errors='igno
```

```
# From this sales df, drop reviews details, but keep averages
# (nb the average and totals will be cross-checked as part of creating the reviews df)
columns_not_required = ['reviews_list']
sales_df.drop(columns=columns_not_required, inplace=True, errors='ignore')

# Print the file and data fields
print_full_summary('Summary For Sales Data - After Cleaning', sales_df)

# Tidy up
del columns_not_required
del missing_flag, missing_cat_tree, missing_cat, duplicates_count
```

```
Summary For Product Data - Initial Look
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	7982	0	0	7982	object	7982	0	0	0
sku	7982	0	0	6110	object	7982	0	0	0
sale_price	7982	0	0	2063	object	7982	0	0	0
category	7982	0	0	1169	object	7982	0	0	0
category_tree	7982	0	0	1997	object	7982	0	0	0
average_product_rating	7982	0	0	153	float64	0	0	7982	0
product_url	7982	0	0	7982	object	7982	0	0	0
total_number_reviews	7982	0	0	22	int64	0	7982	0	0
Reviews	7982	0	0	7982	object	0	0	0	7982
Bought With	7982	0	0	7982	object	0	0	0	7982

```
First 3 Rows
```

	uniq_id	sku	sale_price	category	category_tree
0	b6c0b6bea69c722939585baeac73c13d	pp5006380337	24.16	alfred dunner	jcpenney|women|alf
1	93e5272c51d8cce02597e3ce67b7ad0a	pp5006380337	24.16	alfred dunner	jcpenney|women|alf
2	013e320f2f2ec0cf5b3ff5418d688528	pp5006380337	24.16	view all	jcpenney|women|vi

```
Missing: Categories 636 Trees 636
Duplicated URLs: 0
Summary For Sales Data - After Cleaning
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	7982	0	0	7982	object	7982	0	0	0
sku	7982	0	0	6110	object	7982	0	0	0
sale_price	7982	0	0	1992	float64	0	0	7982	0
category	7982	0	0	1169	object	7982	0	0	0
category_tree	7982	0	0	1997	object	7982	0	0	0
average_product_rating	7982	0	0	153	float64	0	0	7982	0
sales_product_url	7982	0	0	7982	object	7982	0	0	0

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
total_number_reviews	7982	0	0	22	int64	0	7982	0	0
bought_with_list	7982	0	0	7982	object	0	0	0	7982

First 3 Rows

	uniq_id	sku	sale_price	category	category_tree
0	b6c0b6bea69c722939585baeac73c13d	pp5006380337	24.16	alfred dunner	jcpenney|women|alf
1	93e5272c51d8cce02597e3ce67b7ad0a	pp5006380337	24.16	alfred dunner	jcpenney|women|alf
2	013e320f2f2ec0cf5b3ff5418d688528	pp5006380337	24.16	view all	jcpenney|women|vi

```
# Check valid URLs

# Get a few random product URLs to test
product_random = sales_df.sample(n=10)
for row in product_random.itertuples():
    url = row.sales_product_url
    if check_url(url):
        print(f'URL: {url} is good.')
        break
    else:
        print(f'URL: {url} failed.')
```

200
URL: http://www.jcpenney.com/liz-claiborne-leanne-shoulder-bag/prod.jump?ppId=pp500579067

Customer Sales Reviews

```
# Establish the dataframe with all customer reviews
# (nb initial working df prepared earlier during validation of the sales df)

customer_reviews_df = pd.DataFrame(columns=['uniq_id', 'customer_id', 'review_text', 'rev

 # Print the file and data fields
print_full_summary('Summary For Customer Reviews Data - Reviews Not Decoded', working_cus

# Extract all reviews held in JSON format in the reviews list
# Include the uniq_id to link each review back to the original sales details

# Iterate through all rows of the original products data, to extract and decode the serie
# TODO: Ittertuples is fast but this takes 4 seconds to run, replace with a more efficien
for row in working_customer_reviews_df.itertuples(index=False):
    # Create reviews from decoded JSON & add the uniq_id for this row
    reviews_json = row.reviews_list
    reviews_dict_string = json.dumps(reviews_json)
    reviews_df = pd.DataFrame(json.loads(reviews_dict_string))
```

```
    reviews_df.columns = ['customer_id', 'review_text', 'review_score']
    reviews_df.insert(0, 'uniq_id', row.uniq_id)
    reviews_df = reviews_df

    # Cross-check the customer_id for each review
    # Several not found so need to iterate through each and create a dummy customer recor
    if not reviews_df.customer_id.isin(customers_df.customer_id).all():
        print(f'For: {row.uniq_id} not all customers match')
        for customer in reviews_df.itertuples(index=False):
            if customer.customer_id not in customers_df.customer_id.values:
                print(f'Adding dummy customer for ID: {customer.customer_id}')
                new_customer = pd.DataFrame([{'customer_id': customer.customer_id,
                                              'DOB': pd.NaT,
                                              'state_ISO': 'XX',
                                              'uniq_id_list':[]}])
                customers_df = pd.concat([customers_df, new_customer], ignore_index=True)

    # Check the average and totals originally in the source file
    if not math.isclose(row.average_product_rating, reviews_df.review_score.mean()):
        print (f'For: {row.uniq_id} ratings mismatch, original: {row.average_product_rati
    if not math.isclose(row.total_number_reviews, len(reviews_df)):
        print (f'For: {row.uniq_id} counts mismatch, original: {row.total_number_reviews}

    # Add the review to the customers review df
    customer_reviews_df = pd.concat([customer_reviews_df, reviews_df])

 # Print the completed file and data fields
print_full_summary('Summary For Customer Reviews Data - All Reviews', customer_reviews_df

# Tidy up
del row, reviews_json, reviews_dict_string, reviews_df, working_customer_reviews_df, new_
```

```
Summary For Customer Reviews Data - Reviews Not Decoded
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	7982	0	0	7982	object	7982	0	0	0
average_product_rating	7982	0	0	153	float64	0	0	7982	0
total_number_reviews	7982	0	0	22	int64	0	7982	0	0
reviews_list	7982	0	0	7982	object	0	0	0	7982

```
First 3 Rows
```

	uniq_id	average_product_rating	total_number_reviews	reviews_list
0	b6c0b6bea69c722939585baeac73c13d	2.625	8	[{'User': 'fs
1	93e5272c51d8cce02597e3ce67b7ad0a	3.000	8	[{'User': 'tp
2	013e320f2f2ec0cf5b3ff5418d688528	2.625	8	[{'User': 'pc

```
For: e5bdf53f2374569526c9f4d55afdd88e not all customers match
Adding dummy customer for ID: dqft3311
Summary For Customer Reviews Data - All Reviews
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
uniq_id	39063	0	0	7982	object	39063	0	0	0
customer_id	39063	0	0	4993	object	39063	0	0	0
review_text	39063	0	0	29464	object	39063	0	0	0
review_score	39063	0	0	5	object	0	39063	0	0

```
First 3 Rows
```

	uniq_id	customer_id	review_text
0	b6c0b6bea69c722939585baeac73c13d	fsdv4141	You never have to worry about the fit...Alfred...
1	b6c0b6bea69c722939585baeac73c13d	krpz1113	Good quality fabric. Perfect fit. Washed very ...
2	b6c0b6bea69c722939585baeac73c13d	mbmg3241	I do not normally wear pants or capris that ha...

```
# Further cross-validation of reviews and customers data

# Tidy up customers_df to cross-check and drop the uniq_id_list

# Check that all sales listed in the customers df match customer reviews
# !! This takes 24 seconds, so very slow, but needs to iterate nested to handle individua
for customer in customers_df.itertuples(index=False):
    sales_list = customer.uniq_id_list
    for uniq_id in sales_list:
        # Check if the sale exists in the sales df
        if uniq_id in sales_df.uniq_id.values:
            # Check that a unique review exists in the customer reviews df for the sale +
            matched_reviews = customer_reviews_df.loc[
                (customer_reviews_df['customer_id'] == customer.customer_id) &
                (customer_reviews_df['uniq_id'] == uniq_id)]
            if len(matched_reviews) != 1:
                print(f'For customer: {customer.customer_id} + uniq_id {uniq_id} reviews
        else:
            print(f'For customer: {customer.customer_id}. A uniq_id {uniq_id} does not ex

# Drop the uniq_id_list from the customers df as all covered in the customer reviews file
customers_df.drop(columns='uniq_id_list', inplace=True, errors='ignore')

# Tidy up
del customer, sales_list, uniq_id, matched_reviews
```

```
For customer: fwbl1442 + uniq_id fe4541f4c1dde497edda95fa46e9e98d reviews = 2
For customer: DUP001dqft3311 + uniq_id 5f280fb338485cfc30678998a42f0a55 reviews = 0
For customer: ffxf2322 + uniq_id b28c5fe83b8b20b05c2451e79cea85f1 reviews = 2
For customer: vwuj3242 + uniq_id e7bea081cac88a6bdcb1d447a4253bab reviews = 2
```

```
For customer: ntvh2341 + uniq_id fedc1fca14619493cd14436a9817c4f2 reviews = 2
For customer: slos2412 + uniq_id 63251a30df90f586fb769ddf2aa5ed54 reviews = 2
For customer: mbdt1413 + uniq_id 77661aaf8abd87167e310721616c6f6a reviews = 2
For customer: pawj4231 + uniq_id a60d13f2f6313bd961546c40c6a3ca96 reviews = 2
For customer: jeph4124 + uniq_id 2dcd61eaea3a7ded2049f305391ae2b8 reviews = 2
For customer: DUP002dqft3311 + uniq_id 571b86d307f94e9e8d7919b551c6bb52 reviews = 0
For customer: ndkl1344 + uniq_id 387d1795d7221b01252a2d8eff30ba87 reviews = 2
For customer: fnmd4431 + uniq_id 07647adc11b605d1a50ccc163eb96c54 reviews = 2
For customer: wnmx2211 + uniq_id 6f7a799e8e5bd4c959379217a776eb86 reviews = 2
```

```
# Look at how many reviews are duplicates and how many customers are linked to these

duplicates_by_customer = customer_reviews_df.groupby('review_text')['customer_id'].size()
reviews_duplicated = duplicates_by_customer.groupby('cust_count').count().reset_index()

count_reviews_single = reviews_duplicated[reviews_duplicated['cust_count'] == 1]
count_reviews_duplicated = len(customer_reviews_df) - count_reviews_single['review_text']
max_duplicates = reviews_duplicated['cust_count'].max()

print(f'Out of a total of {len(customer_reviews_df)} reviews {count_reviews_duplicated} a
print(f'Or approximately {((count_reviews_duplicated/len(customer_reviews_df)) * 100):.0f
print(f'Several worst case situations with {max_duplicates} customers using the same revi

# Tidy up
del duplicates_by_customer
del reviews_duplicated
del count_reviews_single
del count_reviews_duplicated
del max_duplicates
```

```
Out of a total of 39063 reviews 15535 are duplicates.
Or approximately 40%
Several worst case situations with 18 customers using the same review comments.
```

```
# Load the CSV reviews file to cross-check against the data extracted from the JSON sourc

# Load the reviews .csv file, exit if do not exist or are invalid
file_path = os.path.join(os.getcwd(), DATA_DIRECTORY, 'reviews.csv')
if not os.path.isfile(file_path):
    raise Exception(f"File not found: {file_path}")
source_reviewsCSV_df = pd.read_csv(file_path)

# Initial look at the file and data fields
print(f'Summary for customer reviews - CSV')
print_file_summary(source_reviewsCSV_df)
print(f'First 3 rows')
display(source_reviewsCSV_df.head(3))

# Scores look very different
count_zero_scores = source_reviewsCSV_df[source_reviewsCSV_df['Score'] == 0]['Score'].cou
```

```
count_zero_scoresJSON = customer_reviews_df[customer_reviews_df['review_score'] == 0]['re

print(f'Compare JSON sourced review vs CSV file source')
print(f'Count:  {len(customer_reviews_df)} vs {len(source_reviewsCSV_df)}')
print(f'Scores with zero: {count_zero_scoresJSON:.0f} vs {count_zero_scores:.0f}')
print(f'Mean:  {customer_reviews_df['review_score'].mean():.1f} vs {source_reviewsCSV_df[

# Tidy Up
del count_zero_scores, count_zero_scoresJSON, source_reviewsCSV_df
```

```
Summary for customer reviews - CSV
```

	Count	Missing	Empty	Unique	Type	String	Int	Float	List
Uniq_id	39063	0	0	7982	object	39063	0	0	0
Username	39063	0	0	4993	object	39063	0	0	0
Score	39063	0	0	6	int64	0	39063	0	0
Review	39063	0	0	29463	object	39063	0	0	0

```
First 3 rows
```

	Uniq_id	Username	Score	Review
0	b6c0b6bea69c722939585baeac73c13d	fsdv4141	2	You never have to worry about the fit...Alf
1	b6c0b6bea69c722939585baeac73c13d	krpz1113	1	Good quality fabric. Perfect fit. Washed ve
2	b6c0b6bea69c722939585baeac73c13d	mbmg3241	2	I do not normally wear pants or capris that

```
Compare JSON sourced review vs CSV file source
Count:  39063 vs 39063
Scores with zero: 0 vs 11265
Mean:  3.0 vs 1.5
```

Tidy Up and Save

```
# Store the completed working dataframes for analysis in separate workbooks
%store sales_df stock_df customer_reviews_df customers_df states_df
```

```
Stored 'sales_df' (DataFrame)
Stored 'stock_df' (DataFrame)
Stored 'customer_reviews_df' (DataFrame)
Stored 'customers_df' (DataFrame)
Stored 'states_df' (DataFrame)
```