Report_Draft

December 2, 2024

1 Summary

- ?? high-level summary of conclusions
- ?? report objectives etc??

2 Objectives & Approach

This report describes the findings from an analysis of sales review data for a US retail chain, JCPenney. It is the final assignment in the module: ITNPBD2, Representing and Manipulating Data and uses several datsets provided, with some suplimentary data to augment this.

The report is structured into four sections: - Summary - Data Collation, Exploration & Preparation - Visualation & Initial Analysis - Conclusions

2.0.1 Approach

This has primarily been written as a business report, with conclusions and supporting details. However, it also provides details of the analysis approach taken and listings of the Python code used. In order, not to disrupt the flow of the reporting, as much as is possible, the Python code has been put towards the end of report sections **??? or in appendices ??**.

I was not totally clear about the relative importance of the assignment objectives and so have attempted to meet the objectives stated below, in order of priority: - Provide a business report to senior managers who are not interested in the detailed approach or source data - Demonstrate that a data science process (here CRISP-DM) has been followed - Demonstrate the use of Python for data manipulation and analysis

The original source and timing for the provided datasets is not known and so the following assumptions have been made - Source is from JCP's operational stock and sales systems - The data was extracted in Q3 2023

!! your code must be working, correct, and well commented and shows an appreciation of style, efficiency and reliability. All choices for methods and structures are concisely justified and alternatives are given well thought considerations.

2.0.2 CRISP-DM

The approach to the analysis follows the first four parts of the six stage "CRoss-Industry Standard Process for Data Mining' (CRISP-DM) process. ***[!! See citations url and (Ncr et al., 1999) !!]***. In summary, this process is:

1. Business Understanding: Define project objectives and requirements by collaborating with stakeholders
2. Data Understanding: Collect and explore data, analyzing its characteristics and quality
3. Data Preparation: Clean, handle missing values, and transform variables to create a structured dataset
4. Modeling: 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

2.0.3 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. Later, in August 2023, JCP announced a major turnaround plan to replace its current website and inventory management systems, as well as make major upgardes to its retails stores. See this article for more details: Modern Retail Sep 2023.

3 Data Collation, Exploration & Preparation

[1]:
```
# 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 ....
## cccccc

# Variables used throughout the notebook
DATA_DIRECTORY = 'JCPenney_Data_Original'  # Designated data folder within the␣
 ↪current working directory
AUGMENTED_DATA = 'Data_Additional'         # Additional data sources

# A simple utility function 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 identified for uniquness
    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))
```

3.0.1 TO DO

- 1. Data Exploration - Explore the data and show you understand its structure and relations
- 2. Data Validation - Check the quality of the data. Is it complete? Are there obvious errors?
- x. Data Preparation - Addresss the issues identified, supliment/augment the data, restructure
  - Select Data - which to use and which to exclude .. with reasoning
  - Clean Data - 'correct, impute, remove' …
  - Construct Data - derive new attributes as needed/helpful
  - Integrate Data - new datasets, augment, other sources
  - Format Data - reformat as needed, eg string to numeric, dates, categorical
- ?? data structure, size
- ?? data quality, missing etc

Data Completeness

- Missing Data - Identify and resolve by removal or inferring. Imputation and visualisation, examining descriptive stats
- Noisy - Random errors, eg from faulty sensors, data transmission
- Duplicates - Identify and eliminate duplicates, redundancy. NB: Can occur after data integration
- Inconsistency - Data items don't align, eg DOB and age

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 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 has looks to have been extracted to prepare the reviews.csv file).

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

3.1.1 Data Sources Overview - JSON

It is assumed that the data is a snapshot extract of sales information from JCP databases and the bulk of this has been flattened and used to create the jcpenney_products.json file with the jcpenney_reviewers.json file providing details of individual customers.

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

[2]:
```python
# 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))
```

```python
# Tidy up
del file_name, file_path
```

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

                       Float  List
uniq_id                    0     0
sku                        0     0
name_title                 0     0
description                0     0
list_price                 0     0
sale_price                 0     0
category                   0     0
category_tree              0     0
average_product_rating  7982     0
product_url                0     0
product_image_urls         0     0
brand                      0     0
total_number_reviews       0     0
Reviews                    0  7982
Bought With                0  7982

First 3 Rows

                            uniq_id            sku  \
0  b6c0b6bea69c722939585baeac73c13d  pp5006380337
1  93e5272c51d8cce02597e3ce67b7ad0a  pp5006380337
2  013e320f2f2ec0cf5b3ff5418d688528  pp5006380337

                                name_title  \
0  Alfred Dunner® Essential Pull On Capri Pant
```

```
1  Alfred Dunner® Essential Pull On Capri Pant
2  Alfred Dunner® Essential Pull On Capri Pant

                                        description list_price sale_price  \
0  You'll return to our Alfred Dunner pull-on cap…      41.09      24.16
1  You'll return to our Alfred Dunner pull-on cap…      41.09      24.16
2  You'll return to our Alfred Dunner pull-on cap…      41.09      24.16

        category                  category_tree  average_product_rating  \
0  alfred dunner  jcpenney|women|alfred dunner                    2.625
1  alfred dunner  jcpenney|women|alfred dunner                    3.000
2       view all       jcpenney|women|view all                    2.625

                                        product_url  \
0  http://www.jcpenney.com/alfred-dunner-essentia…
1  http://www.jcpenney.com/alfred-dunner-essentia…
2  http://www.jcpenney.com/alfred-dunner-essentia…

                                 product_image_urls          brand  \
0  http://s7d9.scene7.com/is/image/JCPenney/DP122…  Alfred Dunner
1  http://s7d9.scene7.com/is/image/JCPenney/DP122…  Alfred Dunner
2  http://s7d9.scene7.com/is/image/JCPenney/DP122…  Alfred Dunner

   total_number_reviews                                        Reviews  \
0                     8  [{'User': 'fsdv4141', 'Review': 'You never hav…
1                     8  [{'User': 'tpcu2211', 'Review': 'You never hav…
2                     8  [{'User': 'pcfg3234', 'Review': 'You never hav…

                                        Bought With
0  [898e42fe937a33e8ce5e900ca7a4d924, 8c02c262567…
1  [bc9ab3406dcaa84a123b9da862e6367d, 18eb69e8fc2…
2  [3ce70f519a9cfdd85cdbdecd358e5347, b0295c96d2b…
```

[3]:
```python
# 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}')
```

```python
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                                  []
```

3.1.2 Data Sources Overview - CSV

TODO: Decode and compare to JSON

3.2 Collation, Augmentation & Data Structure

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 steps taken, Python code used and outputs are described in the sub-sections below and summarised below.

Sales Activity - ?? key to customer review, customers, stock - ??Sales Activity - From the 7982 rows in the file jcpenney_products.json ?????

Customer Reviews - ?? key to sales activity, customer - ??Customer Sales Reviews - XXXX unique items in the file jcpenney_products.json ?????

Customer Details - A reference list of 5,000 unique JCP customers who have submitted reviews of purchases made - Sourced from the file jcpenney_reviewers.json - In the Pandas dataframe: customers_df - Unique key is customer_id

Stock Details - A reference list of 1,154 unique stock details - Derived from the 6,044 unique items in the file jcpenney_products.json - In the Pandas dataframe - Unique key is sku

States & Territories - A reference list of the 57 US states and territories, with population and JCP store numbers per state - Sourced from JCP and the US Census Bureau. Loaded from the file: JCP_Stores_State_Collated.csv - In the Pandas dataframe: states_df - Unique key is state_ISO

3.2.2 Stock Details

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' (https://en.wikipedia.org/wiki/Stock_keeping_unit) and should be a unique identifier. Therefore all instances of sku have been reviewed and collapsed down into a stock list dataframe, seperate from sales activity. The 6,044 product rows have produced 1,154 stock lines.

Data Content After review and vailidation the created dataframe, stock_df, has 1,154 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 sigificant 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, 182 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 - Brand: No missing or duplicated, so just copy one - Rationalise Stock: Retain only a single unique sku reocrd

[4]:
```
# Establish a reference list of all product / stock details
# And also the initial draft of the sales datafreame

# 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)
```

```python
# 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_product_rating',
                        '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':␣
 ↪'description',
                                    '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
new_stock_df = stock_df.head(0).copy()
for sku, group in sku_groups_dup:
    new_sku = stock_df.head(0).copy()
    new_sku.at[0, 'sku'] = sku
    # stock name - Just retain the first recod
    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 duplicated 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')
```

```python
# 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_empty
del sku_count, missing_sku, row, new_id, most_frequent
del new_sku, new_stock_df, columns_not_required, sku, sku_duplicated,␣
 ↪sku_groups_dup, group
```

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

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

                        Float  List
uniq_id                     0     0
sku                         0     0
name_title                  0     0
description                 0     0
list_price                  0     0
sale_price                  0     0
category                    0     0
category_tree               0     0
average_product_rating   7982     0
product_url                 0     0
product_image_urls          0     0
brand                       0     0
total_number_reviews        0     0
Reviews                     0  7982
Bought With                 0  7982

First 3 Rows

                                uniq_id          sku  \
0  b6c0b6bea69c722939585baeac73c13d  pp5006380337
1  93e5272c51d8cce02597e3ce67b7ad0a  pp5006380337
2  013e320f2f2ec0cf5b3ff5418d688528  pp5006380337

                                    name_title  \
0  Alfred Dunner® Essential Pull On Capri Pant
1  Alfred Dunner® Essential Pull On Capri Pant
2  Alfred Dunner® Essential Pull On Capri Pant

                                         description list_price sale_price  \
0  You'll return to our Alfred Dunner pull-on cap…      41.09      24.16
1  You'll return to our Alfred Dunner pull-on cap…      41.09      24.16
2  You'll return to our Alfred Dunner pull-on cap…      41.09      24.16

        category                 category_tree  average_product_rating  \
0  alfred dunner  jcpenney|women|alfred dunner                   2.625
1  alfred dunner  jcpenney|women|alfred dunner                   3.000
2       view all       jcpenney|women|view all                   2.625

                                         product_url  \
0  http://www.jcpenney.com/alfred-dunner-essentia…
1  http://www.jcpenney.com/alfred-dunner-essentia…
2  http://www.jcpenney.com/alfred-dunner-essentia…

                                  product_image_urls          brand  \
0  http://s7d9.scene7.com/is/image/JCPenney/DP122…  Alfred Dunner
1  http://s7d9.scene7.com/is/image/JCPenney/DP122…  Alfred Dunner
2  http://s7d9.scene7.com/is/image/JCPenney/DP122…  Alfred Dunner

   total_number_reviews                                            Reviews  \
0                     8  [{'User': 'fsdv4141', 'Review': 'You never hav…
1                     8  [{'User': 'tpcu2211', 'Review': 'You never hav…
2                     8  [{'User': 'pcfg3234', 'Review': 'You never hav…

                                         Bought With
0  [898e42fe937a33e8ce5e900ca7a4d924, 8c02c262567…
1  [bc9ab3406dcaa84a123b9da862e6367d, 18eb69e8fc2…
2  [3ce70f519a9cfdd85cdbdecd358e5347, b0295c96d2b…
```

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

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

```
                               uniq_id          sku  \
2269  9fa199671d88a2a3cddd06a0dac02763      0903a80
3984  4875e80ad4e5d0d8970850046a4c8b8c  PP100000902
7884  6dcebbf40f3195554080edced28d401b      0903a80

                                         name_title  \
2269  KitchenAid® Artisan® 5-qt. Stand Mixer KSM150PS
3984   Alyx® Gauze Print Tank Top or Millennium Pants
7884  KitchenAid® Artisan® 5-qt. Stand Mixer KSM150PS

                                          description list_price sale_price  \
2269  The mixer you've always dreamed of. Unique mix…     604.31     423.01
3984                                          Missing    Missing    24.1633
7884  The mixer you've always dreamed of. Unique mix…    Missing     604.31

                category                          category_tree  \
2269    small appliances  jcpenney|for-the-home|small appliances
3984  outfits you'll love      jcpenney|women|outfits you'll love
7884             wedding                        jcpenney|wedding

      average_product_rating  \
2269                   2.750
3984                   3.000
7884                   3.125

                                          product_url  \
2269  http://www.jcpenney.com/kitchenaid-artisan-5-q…
3984  http://www.jcpenney.com/alyx-gauze-print-tank-…
7884  http://www.jcpenney.com/kitchenaid-artisan-5-q…

                                   product_image_urls        brand  \
2269  http://s7d9.scene7.com/is/image/JCPenney/09006…  Kitchen Aid
3984  http://s7d9.scene7.com/is/image/JCPenney/DP032…         Alyx
7884  http://s7d2.scene7.com/is/image/JCPenney/DP021…  Kitchen Aid

      total_number_reviews                                         Reviews  \
2269                     8  [{'User': 'vlfw2311', 'Review': 'I dont know w…
3984                     8  [{'User': 'tlim1231', 'Review': 'I was worried…
7884                     8  [{'User': 'lzci4334', 'Review': 'I dont know w…

                                          Bought With
2269  [0f09d5de035bbb347c17f55222d9efa4, dae30fb78a6…
3984  [53cf4a9eb003e2b5e9c63722d1011951, 5b7416f4e6a…
7884  [eb8e7f2068b80379afbae5135b280c7b, 44725052ce6…
```

```
Empty DataFrame
Columns: [uniq_id, sku, name_title, description, list_price, sale_price,␣
↪category, category_tree, average_product_rating, product_url,␣
↪product_image_urls, brand, total_number_reviews, Reviews, Bought With]
Index: []
```

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

                 Count  Missing  Empty  Unique     Type  String  Int  Float  \
sku                1154        0      0    1154   object    1154    0      0
stock_name         1154        0      0    1135   object    1154    0      0
description        1154        0      0    1081   object    1154    0      0
list_price         1154        0      0     322  float64       0    0   1154
stock_image_url    1154        0      0    1141   object    1154    0      0
brand              1154        0      0     228   object    1154    0      0

                 List
sku                0
stock_name         0
description        0
list_price         0
stock_image_url    0
brand              0
```

```
First 3 Rows

       sku                                        stock_name  \
0  0903a80    KitchenAid® Artisan® 5-qt. Stand Mixer KSM150PS
1  13cab12  JCPenney Home  Saratoga Cut-to-Width Fringed B…
2  13e154b         Glamorise® Full-Figure Body Briefer - 6201

                                      description  list_price  \
0  The mixer you've always dreamed of. Unique mix…      604.31
1  Saratoga cut--to-width blackout shade features…       27.80
2  Glamorise's best-selling full-figure body brie…       81.97

                                  stock_image_url        brand
0  http://s7d9.scene7.com/is/image/JCPenney/09006…  Kitchen Aid
1  http://s7d2.scene7.com/is/image/JCPenney/DP121…     JCP HOME
2  http://s7d9.scene7.com/is/image/JCPenney/09006…    Glamorise
```

3.2.3 Sales Activity

Data Content After review and vailidation 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 to stock_df for stock data - 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 - product_url - JCP website url for the product details - average_product_rating - Average of the various listed - total_number_reviews - Count of the review listed - reviews_list - List of user reviews associated 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 were taken: - Invalid & Missiong 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 ***Lookup list price in stock?*** - Categories Missing: 636 categories, category tress are missing. Aboit 10% of the 7,982 sales - Proudcut URL: All good, no missing or duplicated -

Reviews: All format, completion is good for: average_product_rating, total_number_reviews - List of Reviews: ***already coded this … see the end of this seperate out into a seperate reviews dataframe?? decode the individual json and put into a list check users, check totals and score average match the other fields*** - Bought With: ***check uniq-ids match and are in the main list what information does it provide***

```
[14]: # Establish a list of all product sales

      # Use the initial sales file creating during the stock file creation
      # Drop columns that were retained in the stock_df
      columns_not_required = ['name_title', 'description', 'list_price',␣
       ↪'product_image_urls', 'brand']
      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_tree',
                                          'average_product_rating':␣
       ↪'average_product_rating',
                                          'product_url': 'product_url',
                                          'total_number_reviews':␣
       ↪'total_number_reviews',
                                          'Reviews': 'reviews_list', 'Bought With':␣
       ↪'bought_with_list'})

      # 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 - visual inspection shows no duplicate and no missing

      # sale_price several missing and many 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: Categrories {missing_cat} Trees {missing_cat_tree}')

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

'''
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'
'''

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

# Tidy up
del columns_not_required
```

```
Summary For Product Data - Initial Look

                      Count  Missing  Empty  Unique     Type  String   Int  \
uniq_id                7982        0      0    7982   object    7982     0
sku                    7982        0      0    6110   object    7982     0
sale_price             7982        0      0       1  float64       0     0
category               7982        0      0    1169   object    7982     0
category_tree          7982        0      0    1997   object    7982     0
average_product_rating 7982        0      0     153  float64       0     0
product_url            7982        0      0    7982   object    7982     0
total_number_reviews   7982        0      0      22    int64       0  7982
reviews_list           7982        0      0    7982   object       0     0
bought_with_list       7982        0      0    7982   object       0     0

                      Float  List
uniq_id                   0     0
sku                       0     0
```

```
sale_price              7982     0
category                   0     0
category_tree              0     0
average_product_rating  7982     0
product_url                0     0
total_number_reviews       0     0
reviews_list               0  7982
bought_with_list           0  7982

First 3 Rows
                            uniq_id           sku  sale_price       category  \
0  b6c0b6bea69c722939585baeac73c13d  pp5006380337         0.0  alfred dunner
1  93e5272c51d8cce02597e3ce67b7ad0a  pp5006380337         0.0  alfred dunner
2  013e320f2f2ec0cf5b3ff5418d688528  pp5006380337         0.0       view all

                 category_tree  average_product_rating  \
0  jcpenney|women|alfred dunner                   2.625
1  jcpenney|women|alfred dunner                   3.000
2       jcpenney|women|view all                   2.625

                                         product_url  total_number_reviews  \
0  http://www.jcpenney.com/alfred-dunner-essentia…                     8
1  http://www.jcpenney.com/alfred-dunner-essentia…                     8
2  http://www.jcpenney.com/alfred-dunner-essentia…                     8

                                        reviews_list  \
0  [{'User': 'fsdv4141', 'Review': 'You never hav…
1  [{'User': 'tpcu2211', 'Review': 'You never hav…
2  [{'User': 'pcfg3234', 'Review': 'You never hav…

                                    bought_with_list
0  [898e42fe937a33e8ce5e900ca7a4d924, 8c02c262567…
1  [bc9ab3406dcaa84a123b9da862e6367d, 18eb69e8fc2…
2  [3ce70f519a9cfdd85cdbdecd358e5347, b0295c96d2b…

Missing: Categrories 636 Trees 636
Duplicated URLs: 0
Summary For Sales Data - After Cleaning
                        Count  Missing  Empty  Unique     Type  String   Int  \
uniq_id                  7982        0      0    7982   object    7982     0
sku                      7982        0      0    6110   object    7982     0
sale_price               7982        0      0       1  float64       0     0
category                 7982        0      0    1169   object    7982     0
category_tree            7982        0      0    1997   object    7982     0
average_product_rating   7982        0      0     153  float64       0     0
product_url              7982        0      0    7982   object    7982     0
total_number_reviews     7982        0      0      22    int64       0  7982
reviews_list             7982        0      0    7982   object       0     0
```

```
bought_with_list         7982        0      0    7982   object       0     0

                       Float  List
uniq_id                    0     0
sku                        0     0
sale_price              7982     0
category                   0     0
category_tree              0     0
average_product_rating  7982     0
product_url                0     0
total_number_reviews       0     0
reviews_list               0  7982
bought_with_list           0  7982
```

```
First 3 Rows
```

```
                            uniq_id            sku  sale_price       category  \
0  b6c0b6bea69c722939585baeac73c13d  pp5006380337         0.0  alfred dunner
1  93e5272c51d8cce02597e3ce67b7ad0a  pp5006380337         0.0  alfred dunner
2  013e320f2f2ec0cf5b3ff5418d688528  pp5006380337         0.0       view all

                 category_tree  average_product_rating  \
0  jcpenney|women|alfred dunner                   2.625
1  jcpenney|women|alfred dunner                   3.000
2       jcpenney|women|view all                   2.625

                                         product_url  total_number_reviews  \
0  http://www.jcpenney.com/alfred-dunner-essentia…                     8
1  http://www.jcpenney.com/alfred-dunner-essentia…                     8
2  http://www.jcpenney.com/alfred-dunner-essentia…                     8

                                        reviews_list  \
0  [{'User': 'fsdv4141', 'Review': 'You never hav…
1  [{'User': 'tpcu2211', 'Review': 'You never hav…
2  [{'User': 'pcfg3234', 'Review': 'You never hav…

                                    bought_with_list
0  [898e42fe937a33e8ce5e900ca7a4d924, 8c02c262567…
1  [bc9ab3406dcaa84a123b9da862e6367d, 18eb69e8fc2…
2  [3ce70f519a9cfdd85cdbdecd358e5347, b0295c96d2b…
```

3.2.4 Customer Sales Reviews

3.2.5 States & Territories

Additional information collated and loaded for all US states and territories. Used to validate provided state names and augment the data provided to assist later analysis. Contains 51 states and 6 territories.

The data was sourced from: - JCP's store locator, see website - US Census Bureau, see website

Data Content After review and vailidation the resulting Pandas dataframe, states_df, 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 Loaded from the JCP_Stores_State_Collated.csv file and retained for use in validation and analysis of the JCP customer data.

[ ]:
```
# 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': 'state_name',
                                      '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 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
```

3.2.6 Customer Details

Details of customers that have completed a review of a purchase made. With 5,000 unique customer records.

Data Content After review and vailidation the created dataframe, customers_df, consists of: - customer_id - Unique alphanumeric id - DOB - Date of birth - state_ISO - ISO code for the state or territory. NB used as a key to the states dataframe - uniq_id_list - List of uniq_id to cross-reference back to the sales activity records ***4030 unique vs 5000, also appear to be some empty lists***

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 give them a new unique customer_id - Date of Birth: Surprisingly for 5,000 customers only 52 birth dates were found. Closer examination revealed that a day, month sequence was incremented accross years; with the dates range only being from 26 July to 8 August. The dates appear to be artificially generated. Arguably this field should be dropped as it will not provide any meaningful results. However, it has been converted to a date field and retained purely so that it can be used to demonstrate later analysis - 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

???? - ***4999 in the CSV file what have they done with the duplicate***

[ ]:
```
# 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)
```

```python
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 = customers_df.groupby('DOB').size().reset_index(name='counts')
print(f'Dates Count:')
display(dates)
# Drop the date as looks artifically generated and so of no real use in later␣
 ↪analysis
# customers_df = customers_df.drop('DOB', axis=1)

# States validation - lookup ISO codes, add to customer data and check for␣
 ↪invalid matches
customers_df['state_ISO'] = customers_df['state_name'].apply(lambda x:␣
 ↪get_state(x)['state_ISO'] if get_state(x) is not None else None)
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)['state_ISO'] if get_state(x) is not None else None)
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
states = customers_df.groupby('state_ISO').size().reset_index(name='counts')
print(f'Customers by State:')
display(states)

# Reviewed validate
# TODO: x-check these to sales activity and to reviews to make sure consistent

# Tidy up
del duplicates_flag, duplicated, dup_count, new_id, row
del dates
del unmatched_states, states
```

3.2.7 Data Structure

Sales Activity - 7982 rows in the file jcpenney_products.json - uniq_id: this uniquely identifies each of the 7982 rows. A random string - list_price: assumed to be the price at the point of sale in $ ***?? prices consistent accross channels, any pattern?, 18 missing*** *** convert to numeric **- category_tree: breakdown of the stock catgeorisation eg 'jcpenny|women|skechers' - category: lowest level of category** * drop this? why unique 1169 vs 1997 for the treee also ?? not same match to SKU etc eg the use of 'view all' ?? ***- average_product_rating: rating 1 to 5*** a float with some wierd values .. should be categorical? convert? **,** is this a calculated average from individual reviews? check ***- product_url: link to website product details*** 7982 And these are unique to the review, not the product … why? need us vpn to use this? ***- product_image_urls: link to product image*** 6519, more images than product number, less than product URL ***- Bought with: series of items linked using the uniq_id*** check these link up? and rename what is is useful for?***

Customer Reviews - unique items in the file jcpenney_products.json - total_number_reviews: number of customer reviews for each uniq_id / Sales Channel Activity *** is this needed, or just for a cross-check later? convert to numeric** - Reviews: a json item with several customer reviews, each with: user, review text, score (1 to 5)

3.2.8 Customer Reviews

In the "jcpenney_products.json" file the series of JSON encoded review details were extracted and a new customer reviews dataframe was created, with individual review records linked back to the sales activiy using the uniq_id. The table below summarises the fields decoded and key counts.

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 automaticaly generated or that customer ids are being created to generate false reviews. This data has not been dropped from the dataset, although later sentiment analysis of the reviews could be misleading.

The scores for reviews in the CSV file have a large number of zero vslues (11,265 out of 39,063) and a quick examination showed that many scores differ between the JSON and CSV source. Therefore the “reviews.CSV” data source was rejected and only the JSON source was used.

[ ]:
```
# Establish an initial customer reviews dataframe
# By extracting the series of JSON reviews originaly in the jcpenney_products.
 ↪json

# Create a new dataframe for all customer reviews
customer_reviews_df = pd.DataFrame()

# Iterate through all rows of the orginal products data, to extract and decode␣
 ↪the series of JSON data
# Create customer review rows, each using a foreign key uniq_id for the␣
 ↪relevant sales activity
# TODO: This takes 4 seconds to run, replace with a more efficient approach
for next_row in source_jcp_products_df.itertuples(index=False):
    temp_reviews = next_row.Reviews
    #print(f'UI: {next_row.uniq_id}, {temp_reviews}, {type(temp_reviews)}')
    temp_dict_string = json.dumps(temp_reviews)
    temp_reviews_df = pd.DataFrame(json.loads(temp_dict_string))
    temp_reviews_df.insert(0, 'uniq_id', next_row.uniq_id)
    #print(temp_reviews_df)
    customer_reviews_df = pd.concat([customer_reviews_df, temp_reviews_df])

# Cross-check the customer_id to the customers data frame to make sure all exist
if not customer_reviews_df['User'].isin(customers_df['customer_id']).all():
    print(f'Error: Not all customers setup in customers list')

# Tidy up
del temp_reviews
del temp_dict_string
del temp_reviews_df

# Rename customer column names and validate content for each
customer_reviews_df = customer_reviews_df.rename(columns={'User': 'customer_id',
                                                          'Review': 'review',␣
 ↪'Score': 'score'})
```

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

# ?? create products ... sales activity file without reviews
# Ensure a valid customer record exists in the customers dataframe
```

[ ]:
```
# Look at how many reviews are duplicates and how many customers are linked to␣
  ↪these

duplicates_by_customer = customer_reviews_df.groupby('review')['customer_id'].
  ↪size().reset_index(name='cust_count')
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'].sum()
max_duplicates = reviews_duplicated['cust_count'].max()

print(f'Out of a total of {len(customer_reviews_df)} reviews␣
  ↪{count_reviews_duplicated} are duplicates.')
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 review comments.')

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

[ ]:
```
# Load the CSV reviews file to cross-check against the data extracted from the␣
  ↪JSON sourced reviews

# 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'].count()
count_zero_scoresJSON = customer_reviews_df[customer_reviews_df['score'] ==␣
  ↪0]['score'].count()

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['score'].mean():.1f} vs␣
  ↪{source_reviewsCSV_df['Score'].mean():.1f}')
```

```
[ ]: # Sentiment analysis of reviews

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

def sentiment_categorise(sentiment):
    '''
    Positive: Compound score >= 0.05
    Neutral: Compound score > -0.05 and < 0.05
    Negative: Compound score <= -0.05
    '''
    if sentiment <= -0.05:
        return 1
    elif sentiment < 0.05:
        return 3
    else:
        return 5

review_sentiments = [analyser.polarity_scores(review_text)['compound'] for␣
  ↪review_text in customer_reviews_df['review']]
customer_reviews_df['sentimentC'] = [sentiment_categorise(x) for x in␣
  ↪review_sentiments]
customer_reviews_df['sentiment'] = review_sentiments

# TODO: Appears to be an inconsistent use of score, some appear to have 1 as␣
  ↪best whilst others using 5 as best
```

```
# TODO: Classify sentiment anlaysis as good, bad, neutral or 1 3 5? Correlate/
↪fit with score? Evidence to support above?
```

3.2.9 Observations

- 4993 users x-check

4 Data Visualisation

3. Data Visualisation - Gain an overall understanding of the data with visualisations

- Initial review, plots etc
- Initial observations, insights

5 Data Analysis

4. Data Analysis = Set some questions and use the data to answer them
5. Data Augmentation - Add new data from another source to bring new insights to the data you already have

6 References

See the CRISP-DM in the intro - https://www.datascience-pm.com/crisp-dm-2/ - (Ncr et al., 1999) …….. Ncr, P.C. et al. (1999) ‘CRISP-DM 1.0’.