Notes On Data

Derek R Neilson

September 26, 2024

Abstract

This document contains notes on the data. The notes are intended to demonstrate how I filter and manipulate the data, and are purely for my instructor to review.

1 Introduction

In data analysis, the ability to effectively filter and manipulate data is crucial for extracting meaningful insights. This document outlines the methodologies and tools I employ to pre-process and analyze the dataset. The primary focus is on cleaning the data, handling missing values, and transforming data to suit the analytical objectives. These notes serve as a comprehensive guide for understanding my data processing workflow.

2 Data Collection

The data was collected from `https://fdc.nal.usda.gov/`. The dataset is 2.9GB and is labeled `Branded` and is in JSON format. I chose this dataset because it is large and one can assume that it has the most rows because it is so large.

To download the data, I used the following commands:

```
wget https://fdc.nal.usda.gov/fdc-datasets/FoodData_Central_branded_food_json_2024-04-18.zip
unzip FoodData_Central_branded_food_json_2024-04-18.zip
rm FoodData_Central_branded_food_json_2024-04-18.zip
```

Listing 1: Download, Extract, and Remove Zip File

As shown in Listing 1, the commands download, extract, and remove the dataset file.

It is worth noting that I am using git to track changes in the code and data. The git commands will not be shown in this document for brevity.

3 Data Inspection

I received the following files after extracting:

- `brandedDownload.json` - I am assuming that this is the main file
- `foundationDownload.json` - I am assuming that this is a supporting file

The first step to inspecting the data is to view it.

```
less brandedDownload.json # the output is too large to show here and is not useful
# I am going to use jq to view the data
jq . brandedDownload.json # this results in a segmentation fault because the file is too large
# I am going to use a stonger server to veiw the data
# For security resons, the ip address and username are redacted
sftp -P port username@ip_address
put brandedDownload.json DEV/Project/Data
put foundationDownload.json DEV/Project/Data
bye
ssh username@ip_address -P port
```

Listing 2: View the Data

As shown in Listing 2, the file is too large to view on my local machine. I will use a stronger server to view the data. Note that there is a assumption that all commands that follow are run on the server. From here on, I will refer to the server as being the machine that I am using to view the data. To get the data on the server, I used sftp to transfer the file to the server.

```
jq . brandedDownload.json | less # failed because the file is too large
jq --stream . brandedDownload.json | less # this works because it streams the data
jq --stream . foundationDownload.json | less
```

Listing 3: View the Data on the Server

After looking at the head of the data, I can see that the data is in unstructured JSON format. I will use the CSV data instead. The JSON data will not be included in this document for size reasons.

4 Data Collection (CSV)

```
rm brandedDownload.json foundationDownload.json # remove the JSON files
wget https://fdc.nal.usda.gov/fdc-datasets/FoodData_Central_branded_food_csv_2024-04-18.zip #
  ↪ download the CSV file
unzip FoodData_Central_branded_food_csv_2024-04-18.zip # unzip the file
mv FoodData_Central_branded_food_csv_2024-04-18/* . # move the files to the current directory
```

Listing 4: Download, Extract, and Remove Zip File

As shown in Listing 4, the commands download, extract, and remove the dataset file in CSV format. I will use the CSV data for the rest of the analysis.

```
total 2853M
-rw-r--r-- 1 derek derek  870M Apr  5 12:07 branded_food.csv
drwxr-xr-x 3 derek derek    1M Sep 26 09:01 build
-rw-r--r-- 1 derek derek    1M Apr  5 12:12 Download API Field Descriptions.xlsx
-rw-r--r-- 1 derek derek  123M Apr  5 12:18 food_attribute.csv
-rw-r--r-- 1 derek derek    1M Apr  5 12:09 food_attribute_type.csv
-rw-r--r-- 1 derek derek  351M Apr  5 12:18 food.csv
-rw-r--r-- 1 derek derek 1387M Apr  5 12:23 food_nutrient.csv
-rw-r--r-- 1 derek derek  124M Apr  5 12:18 food_update_log_entry.csv
-rw-r--r-- 1 derek derek    1M Sep 25 19:22 makecsv.py
-rw-r--r-- 1 derek derek    1M Apr  5 12:09 measure_unit.csv
-rw-r--r-- 1 derek derek    1M Apr  5 12:08 microbe.csv
-rw-r--r-- 1 derek derek    1M Sep 26 08:10 notes.tex
-rw-r--r-- 1 derek derek    1M Apr  5 12:08 nutrient.csv
-rw-rw-r-- 1 derek derek    1M Apr  5 12:12 nutrient_incoming_name.csv
-rw-r--r-- 1 derek derek    0M Sep 26 09:05 sizes.log
```

Listing 5: list the files

I then looked at the head of each CSV file to see what was in the files. I will not include the output here for brevity. `head -n 1 *.csv`. The only file that I am interested in is `branded_food.csv`. I will use this file for the rest of the analysis. But I also noticed that none of the files had caloric information. As a result, I will use a external dataset to get the caloric information. From a quick search, I found a api `https://platform.fatsecret.com/platform-api`. I will use this api to get the caloric information for each food. It dose cost money to use this API so I will calculate the cost of using this API before I use it. I stored the key in a external file called `.env`. I will not include the key in this document for security reasons. I will use a python script to get the caloric information for each food. At this time I will make a virtual environment to run the script. `python3.12 -m venv .venv` and `source .venv/bin/activate`. I will keep track of any dependencies that I use in a `requirements.txt` file. The next step is to validate the data.

```
import requests
import dotenv
import pandas as pd
import os
from scipy import stats
```

```
import numpy as np

# Load environment variables
dotenv.load_dotenv()

# Path to CSV
csv_file_path = "branded_food.csv"

# Check file existence
if not os.path.isfile(csv_file_path):
    raise FileNotFoundError(f"The file {csv_file_path} does not exist.")

# Load data
try:
    data = pd.read_csv(csv_file_path)
    print("Data loaded successfully.")
except Exception as e:
    print(f"An error occurred while loading the data: {e}")
    raise

print(f"Data shape: {data.shape}")

# 1. Inspect Data Types
print("\n--- Data Types ---")
print(data.dtypes)

# Convert specific columns if necessary
# Example: data['price'] = pd.to_numeric(data['price'], errors='coerce')

# 2. Check for Missing Values
print("\n--- Missing Values ---")
missing_values = data.isnull().sum()
print(missing_values)
missing_percentage = (missing_values / len(data)) * 100
print("\n--- Percentage of Missing Values ---")
print(missing_percentage)

# Handle missing values (Example: Fill numerical with mean, categorical with mode)
numerical_cols = data.select_dtypes(include=[np.number]).columns
categorical_cols = data.select_dtypes(include=["object", "category"]).columns

for col in numerical_cols:
    if data[col].isnull().sum() > 0:
        data[col].fillna(data[col].mean(), inplace=True)

for col in categorical_cols:
    if data[col].isnull().sum() > 0:
        data[col].fillna(data[col].mode()[0], inplace=True)

# 3. Identify Duplicates
print("\n--- Duplicates ---")
duplicate_rows = data.duplicated().sum()
print(f"Number of duplicate rows: {duplicate_rows}")

if duplicate_rows > 0:
    data = data.drop_duplicates()
    print(f"Data shape after removing duplicates: {data.shape}")

# 4. Summary Statistics
print("\n--- Summary Statistics ---")
print(data.describe())

# 5. Detect Outliers using Z-Score
```

```
print("\n--- Detecting Outliers with Z-Score ---")
z_scores = np.abs(stats.zscore(data.select_dtypes(include=[np.number])))
threshold = 3
outliers = (z_scores > threshold).any(axis=1)
print(f"Number of outliers: {outliers.sum()}")

# Optionally remove outliers
data = data[~outliers]
print(f"Data shape after removing outliers: {data.shape}")

# 6. Validate Categorical Data
print("\n--- Categorical Data Validation ---")
for col in categorical_cols:
    unique_vals = data[col].unique()
    print(f"\nUnique values in '{col}':\n", unique_vals)
    # Example: Standardize to lowercase
    data[col] = data[col].str.lower()
    # Example: Replace known inconsistencies
    # data[col] = data[col].replace({'old_value': 'new_value'})

# 7. Ensure Data Consistency and Integrity
print("\n--- Data Consistency Checks ---")
# Example: Date consistency
if "manufacture_date" in data.columns and "expiry_date" in data.columns:
    data["manufacture_date"] = pd.to_datetime(data["manufacture_date"], errors="coerce")
    data["expiry_date"] = pd.to_datetime(data["expiry_date"], errors="coerce")
    invalid_dates = data[data["expiry_date"] < data["manufacture_date"]]
    print(f"Records with expiry_date before manufacture_date: {invalid_dates.shape[0]}")
    # Remove invalid dates
    data = data[data["expiry_date"] >= data["manufacture_date"]]

# Example: Logical numerical relationships
if "calories" in data.columns:
    negative_calories = data[data["calories"] < 0].shape[0]
    print(f"Records with negative calories: {negative_calories}")
    # Remove negative calories
    data = data[data["calories"] >= 0]

print("\n--- Final Data Shape ---")
print(data.shape)
```

Listing 6: Validate the Data

The code in Listing 6 performs the following tasks:

- Load the data from the CSV file
- Check the data types of each column
- Identify and handle missing values
- Identify and remove duplicate rows
- Display summary statistics of the data
- Detect and optionally remove outliers using Z-Score
- Validate categorical data
- Ensure data consistency and integrity

It is nondestructive and will not change the data. The next step is to clean the data.

```
import pandas as pd
import dotenv
import os
```

```
4
5  # Load environment variables (if applicable)
6  dotenv.load_dotenv()
7
8  # Path to your CSV file
9  csv_file_path = "branded_food.csv"
10
11 # Check if the file exists
12 if not os.path.isfile(csv_file_path):
13     raise FileNotFoundError(f"The file {csv_file_path} does not exist.")
14
15 # Load the CSV data into a pandas DataFrame
16 try:
17     data = pd.read_csv(csv_file_path)
18     print("Data loaded successfully.")
19 except Exception as e:
20     print(f"An error occurred while loading the data: {e}")
21     raise
22
23 # Calculate the percentage of missing values per column
24 missing_percentage = (data.isnull().sum() / len(data)) * 100
25 print("\n--- Percentage of Missing Values ---")
26 print(missing_percentage)
27
28 # Define the threshold for missing values
29 threshold = 90  # in percentage
30
31 # Identify columns to drop
32 columns_to_drop = missing_percentage[missing_percentage > threshold].index
33 print("\nColumns to be dropped (>90% missing values):")
34 print(columns_to_drop.tolist())
35
36 # Drop the columns with >90% missing values
37 data_cleaned = data.drop(columns=columns_to_drop)
38 print(f"\nData shape after dropping columns: {data_cleaned.shape}")
39
40 # Define the path for the cleaned CSV
41 cleaned_csv_path = "branded_food_cleaned.csv"
42
43 # Save the cleaned DataFrame to a new CSV
44 data_cleaned.to_csv(cleaned_csv_path, index=False)
45 print(f"\nCleaned data saved to '{cleaned_csv_path}'.")
```

Listing 7: Clean the CSV

The code in Listing 7 performs the following tasks:

- Load the data from the CSV file
- Calculate the percentage of missing values per column
- Identify columns with more than 90% missing values
- Drop columns with more than 90% missing values
- Save the cleaned data to a new CSV file

The cleaned data is saved to a new CSV file called `branded_food_cleaned.csv`. The next step is to analyze the data.