Environment Setup in Google Colab

```
# Install Required Libraries
!pip install pymongo pandas numpy matplotlib seaborn rdflib dask
textblob

Collecting pymongo
  Downloading pymongo-4.10.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (22 kB)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (2.2.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (3.8.0)
Requirement already satisfied: seaborn in
/usr/local/lib/python3.10/dist-packages (0.13.2)
Collecting rdflib
  Downloading rdflib-7.1.1-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: dask in /usr/local/lib/python3.10/dist-
packages (2024.10.0)
Requirement already satisfied: textblob in
/usr/local/lib/python3.10/dist-packages (0.17.1)
Collecting dnspython<3.0.0,>=1.16.0 (from pymongo)
  Downloading dnspython-2.7.0-py3-none-any.whl.metadata (5.8 kB)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.1)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.55.3)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.7)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.2)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.2.0)
Collecting isodate<1.0.0,>=0.7.2 (from rdflib)
  Downloading isodate-0.7.2-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: click>=8.1 in
/usr/local/lib/python3.10/dist-packages (from dask) (8.1.7)
```

```
Requirement already satisfied: cloudpickle>=3.0.0 in
/usr/local/lib/python3.10/dist-packages (from dask) (3.1.0)
Requirement already satisfied: fsspec>=2021.09.0 in
/usr/local/lib/python3.10/dist-packages (from dask) (2024.10.0)
Requirement already satisfied: partd>=1.4.0 in
/usr/local/lib/python3.10/dist-packages (from dask) (1.4.2)
Requirement already satisfied: pyyaml>=5.3.1 in
/usr/local/lib/python3.10/dist-packages (from dask) (6.0.2)
Requirement already satisfied: toolz>=0.10.0 in
/usr/local/lib/python3.10/dist-packages (from dask) (0.12.1)
Requirement already satisfied: importlib-metadata>=4.13.0 in
/usr/local/lib/python3.10/dist-packages (from dask) (8.5.0)
Requirement already satisfied: nltk>=3.1 in
/usr/local/lib/python3.10/dist-packages (from textblob) (3.9.1)
Requirement already satisfied: zipp>=3.20 in
/usr/local/lib/python3.10/dist-packages (from importlib-
metadata>=4.13.0->dask) (3.21.0)
Requirement already satisfied: joblib in
/usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob)
(1.4.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk>=3.1->textblob)
(2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from nltk>=3.1->textblob) (4.67.1)
Requirement already satisfied: locket in
/usr/local/lib/python3.10/dist-packages (from partd>=1.4.0->dask)
(1.0.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2-
>pandas) (1.17.0)
Downloading pymongo-4.10.1-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.4 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 36.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 562.4/562.4 kB 26.3 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 313.6/313.6 kB 19.5 MB/s eta
0:00:00
ongo
Successfully installed dnspython-2.7.0 isodate-0.7.2 pymongo-4.10.1
rdflib-7.1.1
```

```python
# Import Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pymongo import MongoClient
from rdflib import Graph, Literal, RDF, URIRef
```

```
from rdflib.namespace import FOAF, XSD
import dask.dataframe as dd
from textblob import TextBlob
```

```
/usr/local/lib/python3.10/dist-packages/dask/dataframe/__init__.py:42:
FutureWarning:
Dask dataframe query planning is disabled because dask-expr is not
installed.

You can install it with `pip install dask[dataframe]` or `conda
install dask`.
This will raise in a future version.

  warnings.warn(msg, FutureWarning)
```

```
import pandas as pd

# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Load datasets
structured_data = pd.read_csv('/content/sample_dataset.csv')
unstructured_data =
pd.read_json('/content/Musical_Instruments_5.json', lines=True)
```

```
Drive already mounted at /content/drive; to attempt to forcibly
remount, call drive.mount("/content/drive", force_remount=True).
```

```
structured_data.head()
```


```
{"summary":"{\n  \"name\": \"structured_data\",\n  \"rows\": 50000,\n
\"fields\": [\n    {\n      \"column\": \"Customer ID\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
288232,\n        \"min\": 29,\n        \"max\": 999997,\n
\"num_unique_values\": 50000,\n        \"samples\": [\n
612016,\n          852349,\n          59423\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Name\",\n      \"properties\": {\n
\"dtype\": \"category\",\n        \"num_unique_values\": 690,\n
\"samples\": [\n          \"Jake\",\n          \"Carl\",\n
\"Gina\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Surname\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 1000,\n
\"samples\": [\n          \"Sutton\",\n          \"Everett\",\n
\"Gilbert\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Gender\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 2,\n        \"samples\":
[\n          \"M\",\n          \"F\"\n        ],\n
```

```
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Birthdate\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 58,\n        \"samples\": [\n          \"2002-
10-20\",\n          \"2001-10-20\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Transaction Amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
631.6697236039846,\n        \"min\": 5.01,\n        \"max\": 2999.88,\
n        \"num_unique_values\": 34665,\n        \"samples\": [\n
434.28,\n          71.44\n        ],\n        \"semantic_type\":
\"\",\n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"Date\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"num_unique_values\": 287,\n        \"samples\":
[\n          \"2023-06-07\",\n          \"2023-08-11\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Merchant Name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 36939,\n        \"samples\": [\n
\"Soto, Stewart and Jackson\",\n          \"Davenport, Moreno and
Joseph\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Category\",\n      \"properties\": {\n        \"dtype\":
\"category\",\n        \"num_unique_values\": 6,\n        \"samples\":
[\n          \"Cosmetic\",\n          \"Travel\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\n}","type":"dataframe","variable_name":"structured_data"}
```


```
unstructured_data.head()
```


```
{"summary":"{\n  \"name\": \"unstructured_data\",\n  \"rows\": 10261,\
n  \"fields\": [\n    {\n      \"column\": \"reviewerID\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1429,\n        \"samples\": [\n
\"A3OJ0RGAECAGH8\",\n          \"AXMYGK3WC8BPP\",\n
\"A34WEXT7SIRFE4\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"asin\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 900,\n        \"samples\": [\n
\"B0002D0CNA\",\n          \"B007IHYBV2\",\n          \"B0002MJTZ8\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"reviewerName\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1397,\n        \"samples\": [\n          \"K.
Swanson\",\n          \"Andrew Walker\",\n          \"Texman\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"helpful\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"reviewText\",\n
```

```
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 10255,\n        \"samples\": [\n
\"It's hard not to love a cord that carries electrons all the way to
the end. Not sure what, other than complete failure would take stars
away.\",\n          \"If you are looking to walk around while you play
your guitar, then you might want to look into something longer. As a
spare cable to play at home though, it's pretty good.\",\n
\"I bought these to replace a pair of ATH-M45s I had for 6 years. I
used my old headphones for listening to music and for tracking in my
recording studio. These have assumed the same role and perform
wonderfully. I play a lot of electric drums and these cans do a great
job of isolating the outside world from what I hear inside the
headphones. Listening to music on them is a joy. Everyone has an
opinion about a &#34;burn in&#34; period, mine seemed to open up at
about 50-60 hours. I listen to music about 3 - 4 hours a day.\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"overall\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0,\n        \"min\": 1,\n        \"max\": 5,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          3,\n
1,\n          4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"summary\",\n      \"properties\": {\n        \"dtype\": \"string\",\
n        \"num_unique_values\": 8852,\n        \"samples\": [\n
\"5 plugs, 1 power source...this is a good thing\",\n          \"Great
Package. It finally stayed in Tune after multiple adjustments\",\n
\"Good enough for professional use\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"unixReviewTime\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
37797350,\n        \"min\": 1095465600,\n        \"max\": 1405987200,\
n        \"num_unique_values\": 1570,\n        \"samples\": [\n
1266710400,\n          1308700800,\n          1318204800\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"reviewTime\",\n
\"properties\": {\n        \"dtype\": \"object\",\n
\"num_unique_values\": 1570,\n        \"samples\": [\n          \"02
21, 2010\",\n          \"06 22, 2011\",\n          \"10 10, 2011\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    }\n  ]\
n}","type":"dataframe","variable_name":"unstructured_data"}
```


Data Exploration and Cleaning

```
# Display dataset information
structured_data_info = structured_data.info()

# Check for missing values
missing_values = structured_data.isnull().sum()
```

```python
# Check for duplicates
duplicate_rows = structured_data.duplicated().sum()

# Display basic statistics
basic_statistics = structured_data.describe()

# Display unique categories in 'Category'
unique_categories = structured_data['Category'].unique()

# Show the results
structured_data_info, missing_values, duplicate_rows,
basic_statistics, unique_categories
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Customer ID         50000 non-null  int64
 1   Name                50000 non-null  object
 2   Surname             50000 non-null  object
 3   Gender              44953 non-null  object
 4   Birthdate           50000 non-null  object
 5   Transaction Amount  50000 non-null  float64
 6   Date                50000 non-null  object
 7   Merchant Name       50000 non-null  object
 8   Category            50000 non-null  object
dtypes: float64(1), int64(1), object(7)
memory usage: 3.4+ MB
```

```
(None,
 Customer ID            0
 Name                   0
 Surname                0
 Gender              5047
 Birthdate              0
 Transaction Amount     0
 Date                   0
 Merchant Name          0
 Category               0
 dtype: int64,
 0,
         Customer ID  Transaction Amount
 count   50000.00000        50000.000000
 mean   500136.79696          442.119239
 std    288232.43164          631.669724
 min        29.00000            5.010000
 25%    251191.50000           79.007500
 50%    499520.50000          182.195000
```

```
 75%    749854.25000          470.515000
 max    999997.00000         2999.880000,
 array(['Cosmetic', 'Travel', 'Clothing', 'Electronics', 'Restaurant',
        'Market'], dtype=object))
```

Next Steps for Structured Dataset: Data Cleaning

```
# Fill missing Gender values with 'Unknown'
structured_data['Gender'].fillna('Unknown', inplace=True)

# Convert 'Birthdate' and 'Date' columns to datetime
structured_data['Birthdate'] =
pd.to_datetime(structured_data['Birthdate'], errors='coerce')
structured_data['Date'] = pd.to_datetime(structured_data['Date'],
errors='coerce')

# Verify changes
missing_values_after_cleaning = structured_data.isnull().sum()
data_types = structured_data.dtypes

# Display results
missing_values_after_cleaning, data_types
```

```
<ipython-input-11-120c14811117>:2: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  structured_data['Gender'].fillna('Unknown', inplace=True)

(Customer ID           0
 Name                  0
 Surname               0
 Gender                0
 Birthdate             0
 Transaction Amount    0
 Date                  0
 Merchant Name         0
 Category              0
 dtype: int64,
 Customer ID                        int64
 Name                              object
```

```
 Surname                            object
 Gender                             object
 Birthdate                  datetime64[ns]
 Transaction Amount                float64
 Date                       datetime64[ns]
 Merchant Name                      object
 Category                           object
 dtype: object)
```

Unstructured Dataset (JSON) Exploration

```
# Load the unstructured dataset (JSON)
unstructured_dataset_path = 'Musical_Instruments_5.json'
unstructured_data = pd.read_json(unstructured_dataset_path,
lines=True)

# Display dataset information
unstructured_data_info = unstructured_data.info()

# Display the first few rows
unstructured_data_head = unstructured_data.head()

# Check for missing values
unstructured_missing_values = unstructured_data.isnull().sum()

# Display column names
unstructured_columns = unstructured_data.columns

# Show results
unstructured_data_info, unstructured_data_head,
unstructured_missing_values, unstructured_columns

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10261 entries, 0 to 10260
Data columns (total 9 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   reviewerID      10261 non-null  object
 1   asin            10261 non-null  object
 2   reviewerName    10234 non-null  object
 3   helpful         10261 non-null  object
 4   reviewText      10261 non-null  object
 5   overall         10261 non-null  int64
 6   summary         10261 non-null  object
 7   unixReviewTime  10261 non-null  int64
 8   reviewTime      10261 non-null  object
dtypes: int64(2), object(7)
memory usage: 721.6+ KB
```

```
(None,
        reviewerID        asin  \
 0  A2IBPI20UZIR0U  1384719342
 1  A14VAT5EAX3D9S  1384719342
 2  A195EZSQDW3E21  1384719342
 3  A2C00NNG1ZQQG2  1384719342
 4   A94QU4C90B1AX  1384719342

                                       reviewerName   helpful  \
 0  cassandra tu "Yeah, well, that's just like, u...    [0, 0]
 1                                              Jake  [13, 14]
 2                        Rick Bennette "Rick Bennette"    [1, 1]
 3                            RustyBill "Sunday Rocker"    [0, 0]
 4                                     SEAN MASLANKA    [0, 0]

                                          reviewText  overall  \
 0  Not much to write about here, but it does exac...        5
 1  The product does exactly as it should and is q...        5
 2  The primary job of this device is to block the...        5
 3  Nice windscreen protects my MXL mic and preven...        5
 4  This pop filter is great. It looks and perform...        5

                                  summary  unixReviewTime   reviewTime

 0                                   good      1393545600  02 28, 2014

 1                                   Jake      1363392000  03 16, 2013

 2                   It Does The Job Well      1377648000  08 28, 2013

 3          GOOD WINDSCREEN FOR THE MONEY      1392336000  02 14, 2014

 4  No more pops when I record my vocals.      1392940800  02 21, 2014
,
 reviewerID         0
 asin               0
 reviewerName      27
 helpful            0
 reviewText         0
 overall            0
 summary            0
 unixReviewTime     0
 reviewTime         0
 dtype: int64,
 Index(['reviewerID', 'asin', 'reviewerName', 'helpful', 'reviewText',
        'overall', 'summary', 'unixReviewTime', 'reviewTime'],
       dtype='object'))
```

Next Steps for Unstructured Dataset: Data Cleaning

```python
# Fill missing reviewerName with 'Anonymous'
unstructured_data['reviewerName'].fillna('Anonymous', inplace=True)

# Convert 'reviewTime' to datetime
unstructured_data['reviewTime'] =
pd.to_datetime(unstructured_data['reviewTime'], errors='coerce')

# Verify changes
unstructured_missing_values_after_cleaning =
unstructured_data.isnull().sum()
unstructured_data_types = unstructured_data.dtypes

# Display results
unstructured_missing_values_after_cleaning, unstructured_data_types
```

```
<ipython-input-14-6256c6f45d21>:2: FutureWarning: A value is trying to
be set on a copy of a DataFrame or Series through chained assignment
using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never
work because the intermediate object on which we are setting values
always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try
using 'df.method({col: value}, inplace=True)' or df[col] =
df[col].method(value) instead, to perform the operation inplace on the
original object.


  unstructured_data['reviewerName'].fillna('Anonymous', inplace=True)
```

```
(reviewerID        0
 asin              0
 reviewerName      0
 helpful           0
 reviewText        0
 overall           0
 summary           0
 unixReviewTime    0
 reviewTime        0
 dtype: int64,
 reviewerID                object
 asin                      object
 reviewerName              object
 helpful                   object
 reviewText                object
 overall                    int64
 summary                   object
 unixReviewTime             int64
 reviewTime        datetime64[ns]
 dtype: object)
```

Next Step: Data Processing and Transformation

Structured Dataset (CSV) Processing

```
from sklearn.preprocessing import LabelEncoder, MinMaxScaler

# 1. Feature Engineering: Calculate Customer Age
current_year = pd.Timestamp.now().year
structured_data['Age'] = current_year -
structured_data['Birthdate'].dt.year

# 2. Categorical Encoding
le_gender = LabelEncoder()
structured_data['Gender'] =
le_gender.fit_transform(structured_data['Gender'])

le_category = LabelEncoder()
structured_data['Category'] =
le_category.fit_transform(structured_data['Category'])

# 3. Extract Year, Month, and Day from Date
structured_data['Transaction_Year'] = structured_data['Date'].dt.year
structured_data['Transaction_Month'] =
structured_data['Date'].dt.month
structured_data['Transaction_Day'] = structured_data['Date'].dt.day

# 4. Normalize Transaction Amount
scaler = MinMaxScaler()
structured_data['Transaction Amount'] =
scaler.fit_transform(structured_data[['Transaction Amount']])

# Display the first few rows after processing
structured_data.head()
```


{"summary":"{\n \"name\": \"structured_data\",\n \"rows\": 50000,\n
\"fields\": [\n   {\n   \"column\": \"Customer ID\",\n
\"properties\": {\n   \"dtype\": \"number\",\n   \"std\":
288232,\n   \"min\": 29,\n   \"max\": 999997,\n
\"num_unique_values\": 50000,\n   \"samples\": [\n
612016,\n   852349,\n   59423\n   ],\n
\"semantic_type\": \"\",\n   \"description\": \"\"\n   }\
n   },\n   {\n   \"column\": \"Name\",\n   \"properties\": {\n
\"dtype\": \"category\",\n   \"num_unique_values\": 690,\n
\"samples\": [\n   \"Jake\",\n   \"Carl\",\n
\"Gina\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":
\"Surname\",\n   \"properties\": {\n   \"dtype\":
\"category\",\n   \"num_unique_values\": 1000,\n
\"samples\": [\n   \"Sutton\",\n   \"Everett\",\n
\"Gilbert\"\n   ],\n   \"semantic_type\": \"\",\n
\"description\": \"\"\n   }\n   },\n   {\n   \"column\":

```
\"Gender\",\n      \"properties\": {\n        \"dtype\": \"number\",\n
\"std\": 0,\n        \"min\": 0,\n        \"max\": 2,\n
\"num_unique_values\": 3,\n        \"samples\": [\n          0,\n
2,\n          1\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Birthdate\",\n      \"properties\": {\n        \"dtype\": \"date\",\
n        \"min\": \"1948-11-02 00:00:00\",\n        \"max\": \"2005-
10-19 00:00:00\",\n        \"num_unique_values\": 58,\n
\"samples\": [\n          \"2002-10-20 00:00:00\",\n          \"2001-
10-20 00:00:00\",\n          \"1963-10-30 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Transaction Amount\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.21091724302022544,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 34665,\n        \"samples\": [\n
0.14333510302617478,\n          0.022181263293565328,\n
0.3136162838453756\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Date\",\n      \"properties\": {\n        \"dtype\": \"date\",\n
\"min\": \"2023-01-01 00:00:00\",\n        \"max\": \"2023-10-14
00:00:00\",\n        \"num_unique_values\": 287,\n        \"samples\":
[\n          \"2023-06-07 00:00:00\",\n          \"2023-08-11
00:00:00\",\n          \"2023-04-12 00:00:00\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Merchant Name\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
\"num_unique_values\": 36939,\n        \"samples\": [\n
\"Soto, Stewart and Jackson\",\n          \"Davenport, Moreno and
Joseph\",\n          \"Berg, Spears and Robinson\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Category\",\n      \"properties\":
{\n        \"dtype\": \"number\",\n        \"std\": 1,\n
\"min\": 0,\n        \"max\": 5,\n        \"num_unique_values\": 6,\n
\"samples\": [\n          1,\n          5,\n          3\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"Age\",\n      \"properties\": {\n
\"dtype\": \"int32\",\n        \"num_unique_values\": 58,\n
\"samples\": [\n          22,\n          23,\n          61\
n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Transaction_Year\",\n      \"properties\": {\n        \"dtype\":
\"int32\",\n        \"num_unique_values\": 1,\n        \"samples\": [\
n          2023\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Transaction_Month\",\n      \"properties\": {\n        \"dtype\":
\"int32\",\n        \"num_unique_values\": 10,\n        \"samples\":
[\n          3\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"Transaction_Day\",\n      \"properties\": {\n        \"dtype\":
```

```
\"int32\",\n        \"num_unique_values\": 31,\n        \"samples\":
[\n          4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    }\n  ]\
n}","type":"dataframe","variable_name":"structured_data"}
```


Unstructured Dataset (JSON) Processing

```
# Import library for sentiment analysis
from textblob import TextBlob

# 1. Sentiment Analysis: Calculate sentiment polarity for reviewText
unstructured_data['sentiment'] =
unstructured_data['reviewText'].apply(lambda x:
TextBlob(x).sentiment.polarity)

# 2. Helpfulness Score: Calculate helpfulness percentage
unstructured_data['helpfulness_score'] =
unstructured_data['helpful'].apply(lambda x: x[0] / x[1] if x[1] != 0
else 0)

# 3. Extract Review Year from reviewTime
unstructured_data['reviewYear'] =
unstructured_data['reviewTime'].dt.year

# Display the first few rows after processing
unstructured_data.head()
```


```
{"summary":"{\n  \"name\": \"unstructured_data\",\n  \"rows\": 10261,\
n  \"fields\": [\n    {\n      \"column\": \"reviewerID\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1429,\n        \"samples\": [\n
\"A3OJ0RGAECAGH8\",\n          \"AXMYGK3WC8BPP\",\n
\"A34WEXT7SIRFE4\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"asin\",\n      \"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 900,\n        \"samples\": [\n
\"B0002D0CNA\",\n          \"B007IHYBV2\",\n          \"B0002MJTZ8\"\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"reviewerName\",\n
\"properties\": {\n        \"dtype\": \"category\",\n
\"num_unique_values\": 1398,\n        \"samples\": [\n
\"Quaestor \\\"Raoul Duke\\\"\",\n          \"Wynn \\\"Nemesis\\\"\",\
n          \"F. Jones\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"helpful\",\n      \"properties\": {\n        \"dtype\":
\"object\",\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"reviewText\",\n      \"properties\": {\n        \"dtype\":
\"string\",\n        \"num_unique_values\": 10255,\n
\"samples\": [\n          \"It's hard not to love a cord that carries
```

```
electrons all the way to the end. Not sure what, other than complete
failure would take stars away.\",\n          \"If you are looking to
walk around while you play your guitar, then you might want to look
into something longer. As a spare cable to play at home though, it's
pretty good.\",\n          \"I bought these to replace a pair of ATH-
M45s I had for 6 years. I used my old headphones for listening to
music and for tracking in my recording studio. These have assumed the
same role and perform wonderfully. I play a lot of electric drums and
these cans do a great job of isolating the outside world from what I
hear inside the headphones. Listening to music on them is a joy.
Everyone has an opinion about a &#34;burn in&#34; period, mine seemed
to open up at about 50-60 hours. I listen to music about 3 - 4 hours a
day.\"\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"overall\",\n      \"properties\": {\n        \"dtype\": \"number\",\
n        \"std\": 0,\n        \"min\": 1,\n        \"max\": 5,\n
\"num_unique_values\": 5,\n        \"samples\": [\n          3,\n
1,\n          4\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"summary\",\n      \"properties\": {\n        \"dtype\": \"string\",\
n        \"num_unique_values\": 8852,\n        \"samples\": [\n
\"5 plugs, 1 power source...this is a good thing\",\n          \"Great
Package. It finally stayed in Tune after multiple adjustments\",\n
\"Good enough for professional use\"\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"unixReviewTime\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
37797350,\n        \"min\": 1095465600,\n        \"max\": 1405987200,\
n        \"num_unique_values\": 1570,\n        \"samples\": [\n
1266710400,\n          1308700800,\n          1318204800\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"reviewTime\",\n
\"properties\": {\n        \"dtype\": \"date\",\n        \"min\":
\"2004-09-18 00:00:00\",\n        \"max\": \"2014-07-22 00:00:00\",\n
\"num_unique_values\": 1570,\n        \"samples\": [\n
\"2010-02-21 00:00:00\",\n          \"2011-06-22 00:00:00\",\n
\"2011-10-10 00:00:00\"\n        ],\n        \"semantic_type\": \"\",\
n        \"description\": \"\"\n      }\n    },\n    {\n
\"column\": \"sentiment\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 0.19867055019794017,\n
\"min\": -0.8,\n        \"max\": 1.0,\n        \"num_unique_values\":
6569,\n        \"samples\": [\n          0.2773260073260073,\n
0.30734618916437095,\n          0.24345238095238095\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    },\n    {\n      \"column\": \"helpfulness_score\",\n
\"properties\": {\n        \"dtype\": \"number\",\n        \"std\":
0.41995144810006335,\n        \"min\": 0.0,\n        \"max\": 1.0,\n
\"num_unique_values\": 158,\n        \"samples\": [\n
0.7878787878787878,\n          0.09090909090909091,\n
```

```
0.9895833333333334\n        ],\n        \"semantic_type\": \"\",\n
\"description\": \"\"\n      }\n    },\n    {\n      \"column\":
\"reviewYear\",\n      \"properties\": {\n        \"dtype\":
\"int32\",\n        \"num_unique_values\": 11,\n        \"samples\":
[\n          2011,\n          2014,\n          2004\n        ],\n
\"semantic_type\": \"\",\n        \"description\": \"\"\n      }\
n    }\n  ]\
n}","type":"dataframe","variable_name":"unstructured_data"}
```


Key Outcomes from Step 4

Structured Dataset: Cleaned, encoded, normalized, and enriched with Age, Transaction_Year, and other derived fields. Unstructured Dataset: Enriched with sentiment, helpfulness_score, and reviewYear.

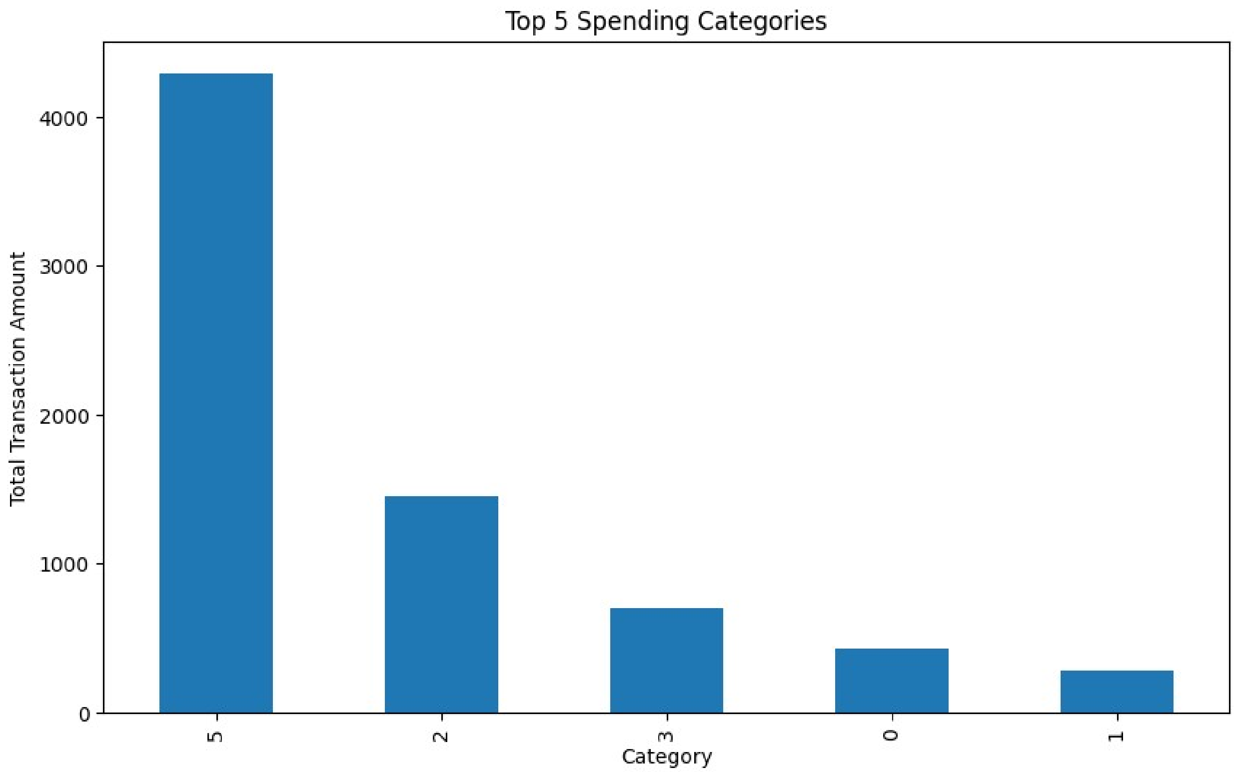
Data Analysis

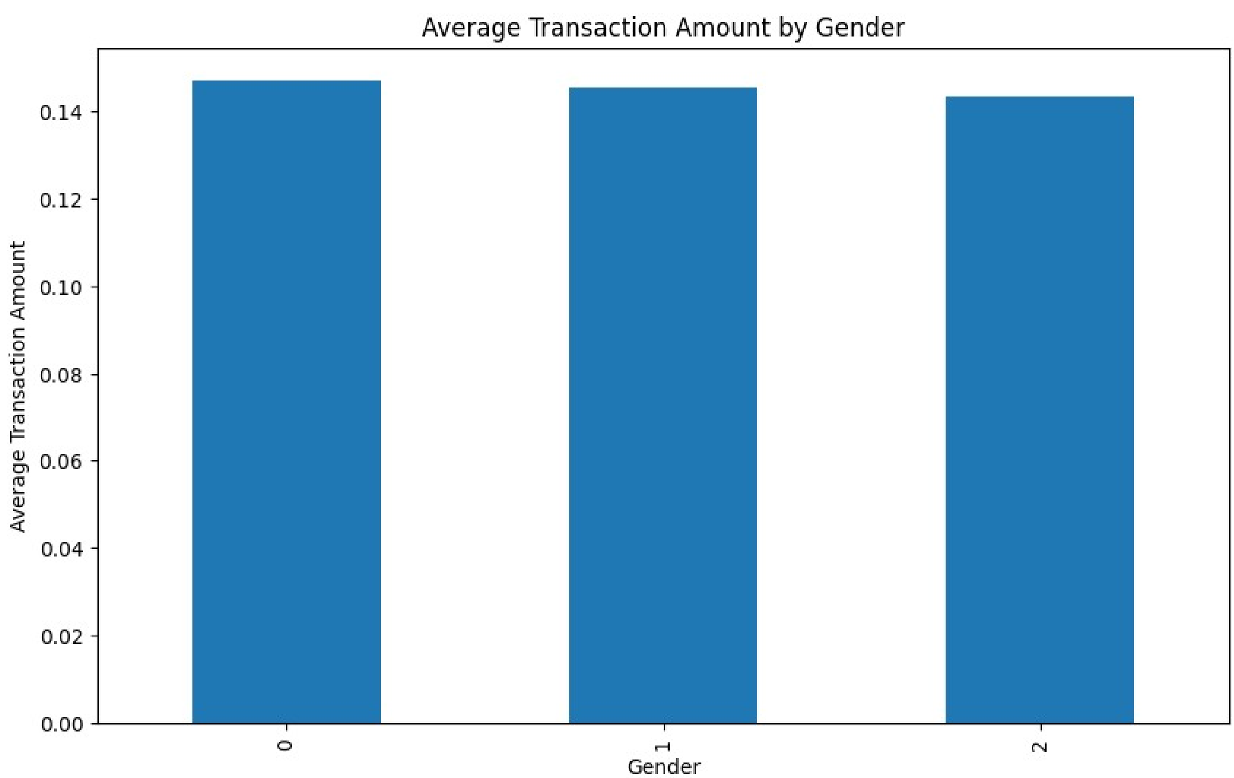
Structured Dataset Analysis

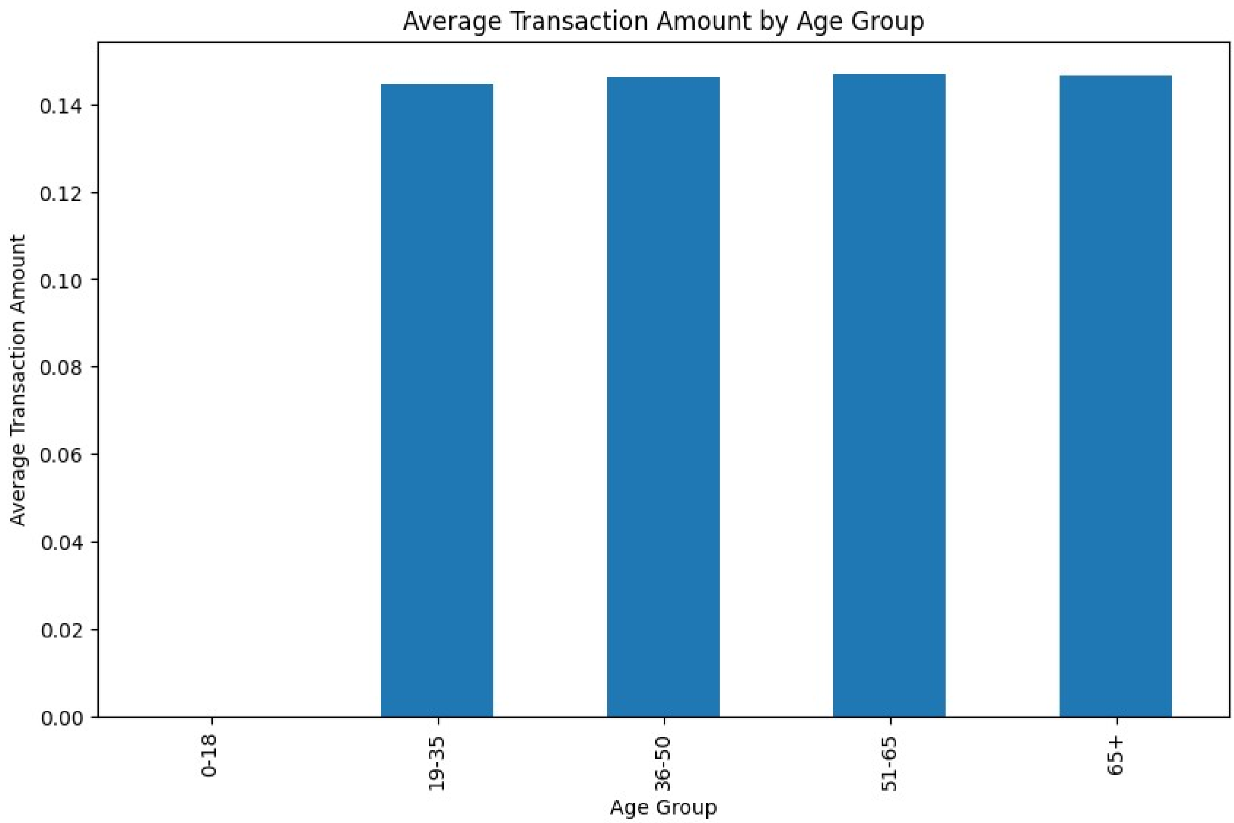
```
import matplotlib.pyplot as plt
import seaborn as sns

# 1. Top Spending Categories
top_categories = structured_data.groupby('Category')['Transaction
Amount'].sum().sort_values(ascending=False).head(5)

# 2. Average Transaction Amount by Gender
avg_transaction_by_gender = structured_data.groupby('Gender')
['Transaction Amount'].mean()

# 3. Average Transaction Amount by Age Group
structured_data['Age_Group'] = pd.cut(structured_data['Age'], bins=[0,
18, 35, 50, 65, 100], labels=['0-18', '19-35', '36-50', '51-65',
'65+'])
avg_transaction_by_age_group = structured_data.groupby('Age_Group')
['Transaction Amount'].mean().sort_index()

# 4. Monthly Transaction Trends
monthly_trends = structured_data.groupby('Transaction_Month')
['Transaction Amount'].sum()

# Plotting results

# Top Spending Categories
plt.figure(figsize=(10, 6))
top_categories.plot(kind='bar')
plt.title('Top 5 Spending Categories')
plt.xlabel('Category')
plt.ylabel('Total Transaction Amount')
plt.show()
```

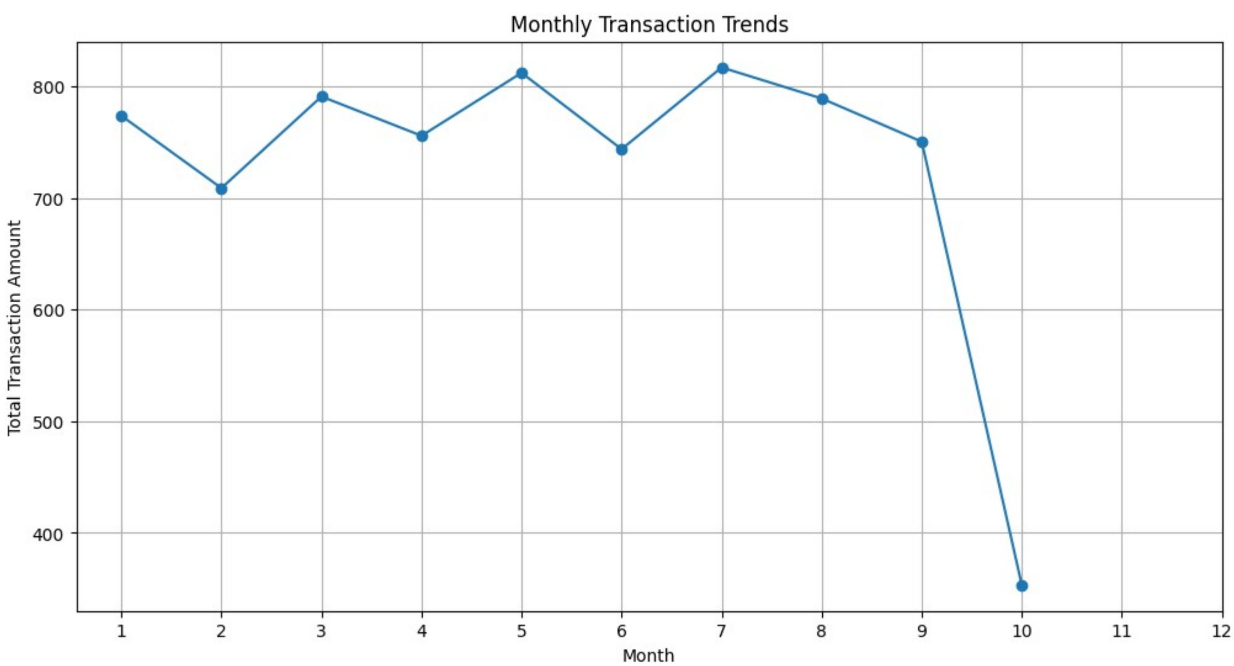
```
# Average Transaction Amount by Gender
plt.figure(figsize=(10, 6))
avg_transaction_by_gender.plot(kind='bar')
plt.title('Average Transaction Amount by Gender')
plt.xlabel('Gender')
plt.ylabel('Average Transaction Amount')
plt.show()

# Average Transaction Amount by Age Group
plt.figure(figsize=(10, 6))
avg_transaction_by_age_group.plot(kind='bar')
plt.title('Average Transaction Amount by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Average Transaction Amount')
plt.show()

# Monthly Transaction Trends
plt.figure(figsize=(12, 6))
monthly_trends.plot(kind='line', marker='o')
plt.title('Monthly Transaction Trends')
plt.xlabel('Month')
plt.ylabel('Total Transaction Amount')
plt.xticks(range(1, 13))
plt.grid(True)
plt.show()

# Return summarized data for reference
top_categories, avg_transaction_by_gender,
avg_transaction_by_age_group, monthly_trends
```

```
<ipython-input-17-48762c66d4d3>:12: FutureWarning: The default of
observed=False is deprecated and will be changed to True in a future
version of pandas. Pass observed=False to retain current behavior or
observed=True to adopt the future default and silence this warning.
  avg_transaction_by_age_group = structured_data.groupby('Age_Group')
['Transaction Amount'].mean().sort_index()
```


Top 5 Spending Categories
Total Transaction Amount
0
1000
2000
3000
4000
5
2
3
0
1
Category


Average Transaction Amount by Gender
Average Transaction Amount
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
1
2
Gender





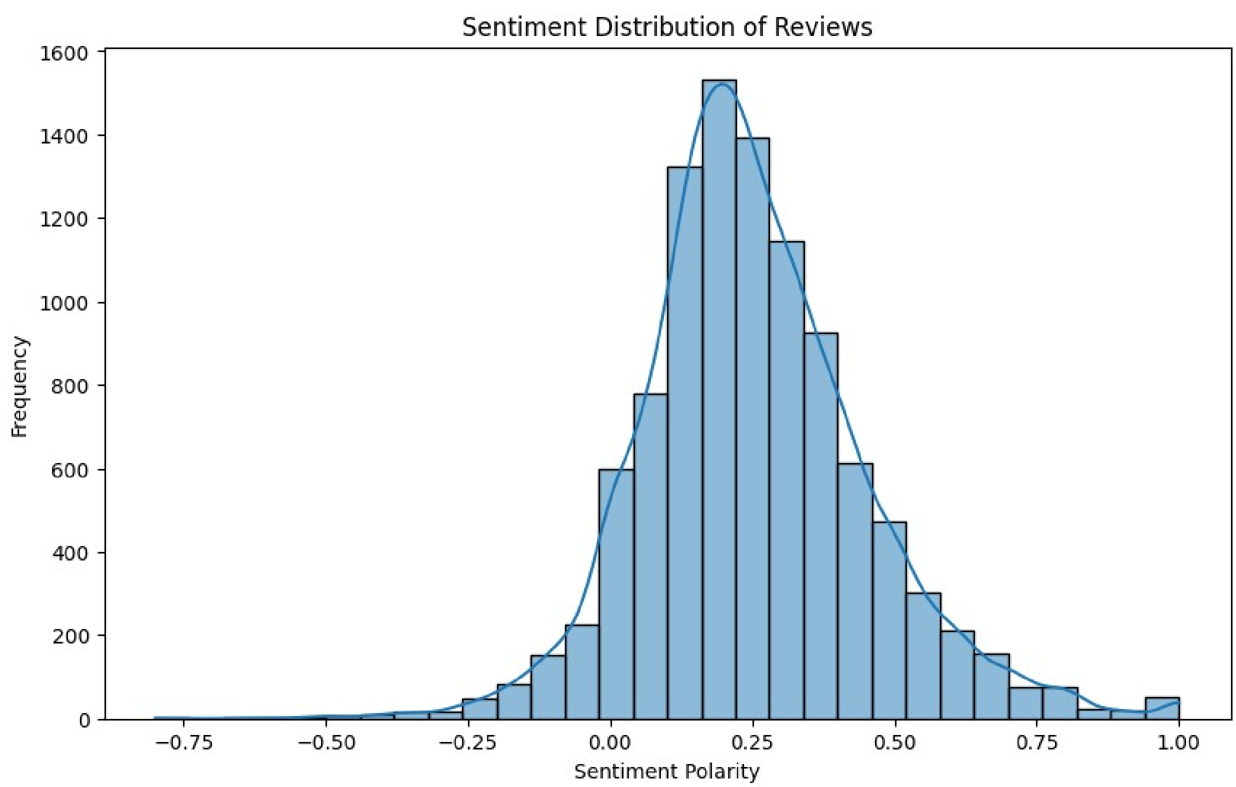

```
(Category
 5    4293.429488
 2    1453.281398
 3     704.251136
```

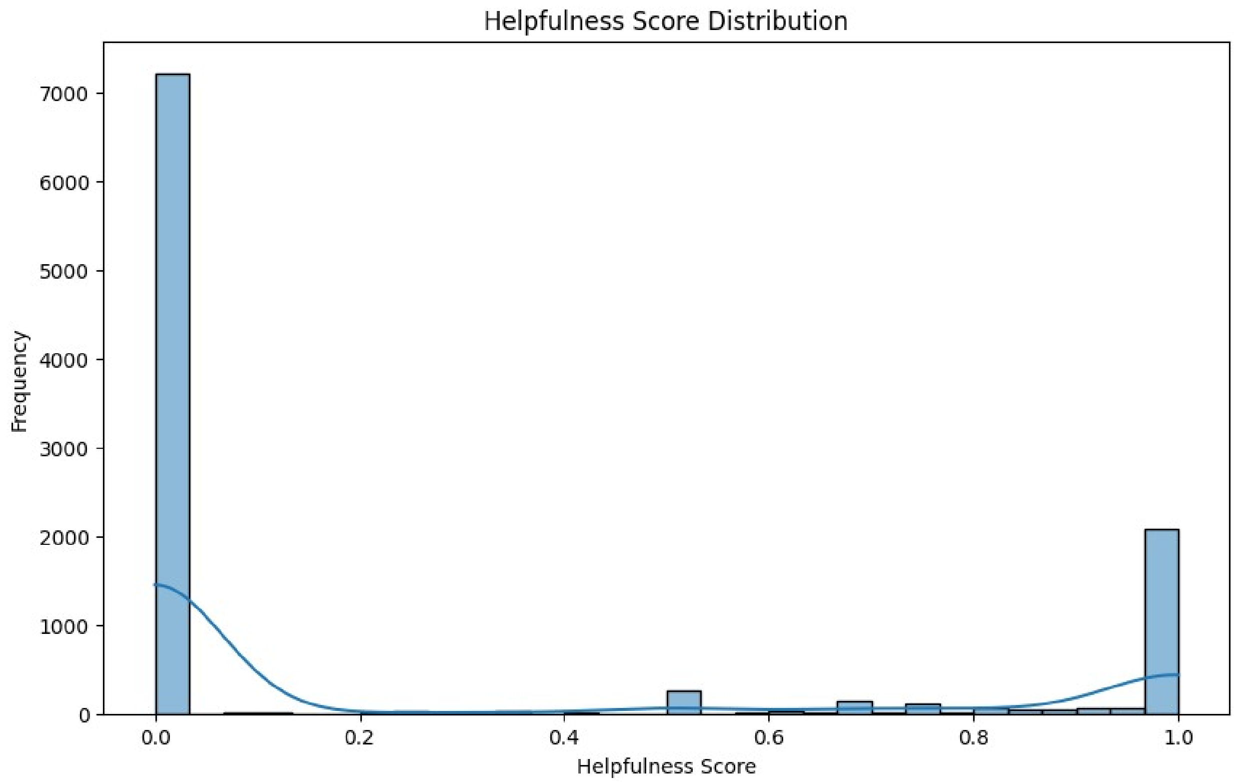
```
0     426.714635
1     278.935316
Name: Transaction Amount, dtype: float64,
Gender
0    0.147089
1    0.145384
2    0.143345
Name: Transaction Amount, dtype: float64,
Age_Group
0-18          NaN
19-35    0.144454
36-50    0.146153
51-65    0.147060
65+      0.146503
Name: Transaction Amount, dtype: float64,
Transaction_Month
1     773.974753
2     709.116726
3     791.229252
4     756.034616
5     812.488232
6     744.022221
7     817.296560
8     789.483754
9     750.760177
10    353.226651
Name: Transaction Amount, dtype: float64)
```

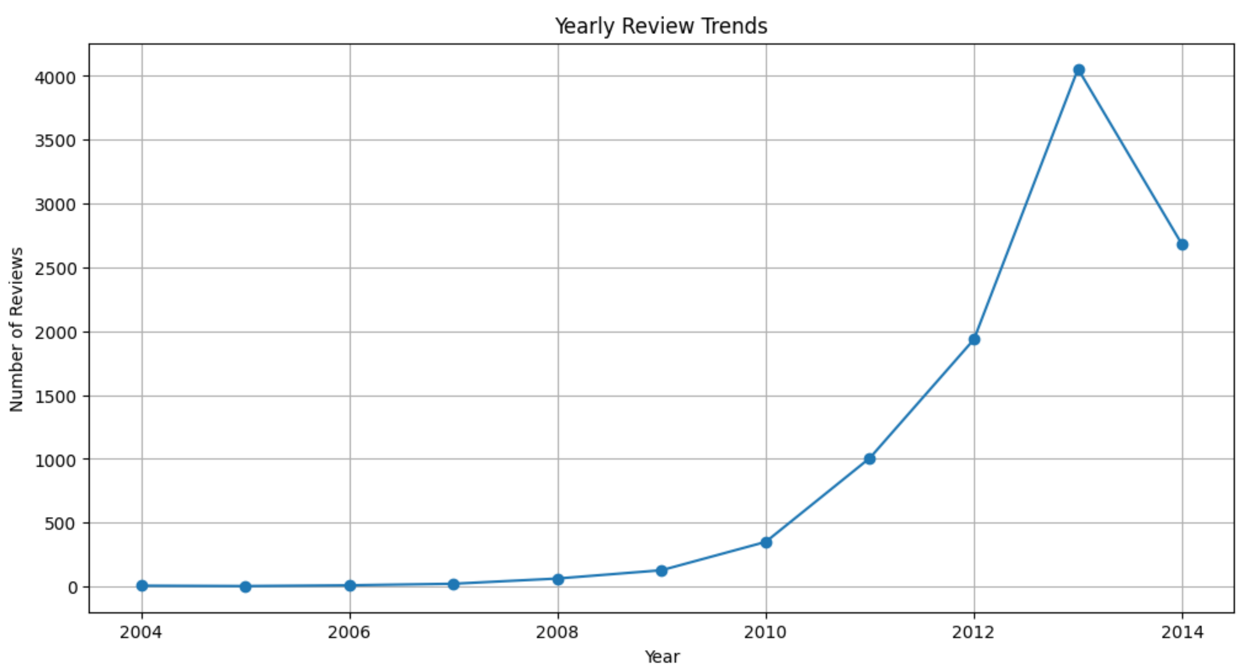
Unstructured Dataset Analysis

```
# 1. Sentiment Distribution
sentiment_distribution = unstructured_data['sentiment'].describe()

# 2. Helpfulness Score Distribution
helpfulness_distribution =
unstructured_data['helpfulness_score'].describe()

# 3. Yearly Review Trends
yearly_review_trends =
unstructured_data['reviewYear'].value_counts().sort_index()

# Plotting results

# Sentiment Distribution
plt.figure(figsize=(10, 6))
sns.histplot(unstructured_data['sentiment'], kde=True, bins=30)
plt.title('Sentiment Distribution of Reviews')
plt.xlabel('Sentiment Polarity')
plt.ylabel('Frequency')
```

```
plt.show()

# Helpfulness Score Distribution
plt.figure(figsize=(10, 6))
sns.histplot(unstructured_data['helpfulness_score'], kde=True,
bins=30)
plt.title('Helpfulness Score Distribution')
plt.xlabel('Helpfulness Score')
plt.ylabel('Frequency')
plt.show()

# Yearly Review Trends
plt.figure(figsize=(12, 6))
yearly_review_trends.plot(kind='line', marker='o')
plt.title('Yearly Review Trends')
plt.xlabel('Year')
plt.ylabel('Number of Reviews')
plt.grid(True)
plt.show()

# Return summarized data for reference
sentiment_distribution, helpfulness_distribution, yearly_review_trends
```








```
(count    10261.000000
 mean         0.253171
 std          0.198671
 min         -0.800000
 25%          0.132778
```

```
50%          0.233056
75%          0.361905
max          1.000000
Name: sentiment, dtype: float64,
count    10261.000000
mean         0.263753
std          0.419951
min          0.000000
25%          0.000000
50%          0.000000
75%          0.666667
max          1.000000
Name: helpfulness_score, dtype: float64,
reviewYear
2004       7
2005       4
2006      10
2007      22
2008      63
2009     128
2010     350
2011    1007
2012    1936
2013    4055
2014    2679
Name: count, dtype: int64)
```

Key Outcomes from Step 5

Structured Dataset:

Top spending categories and peak months were identified. Age and gender patterns in spending were revealed. Unstructured Dataset:

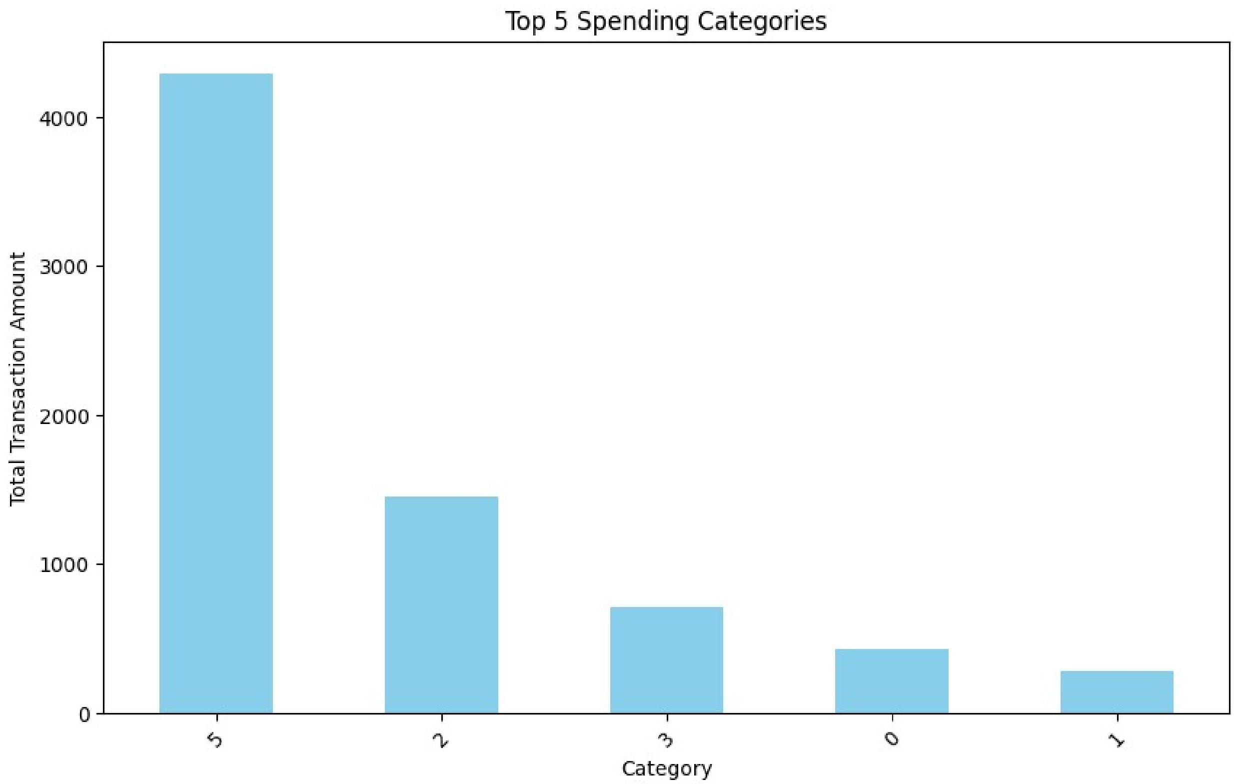
Sentiment analysis showed generally positive feedback. Helpfulness scores revealed most reviews had low helpfulness votes. Review activity peaked in 2013.

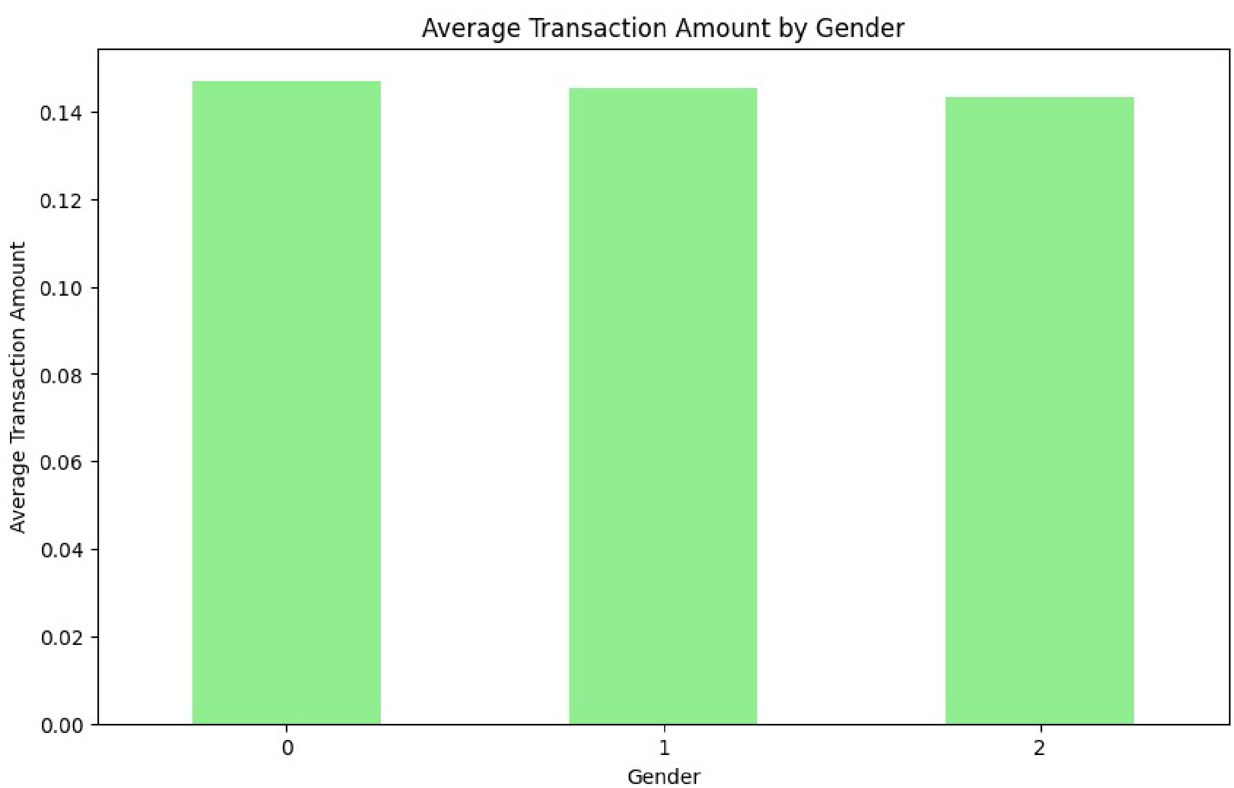
Step 6: Data Visualization

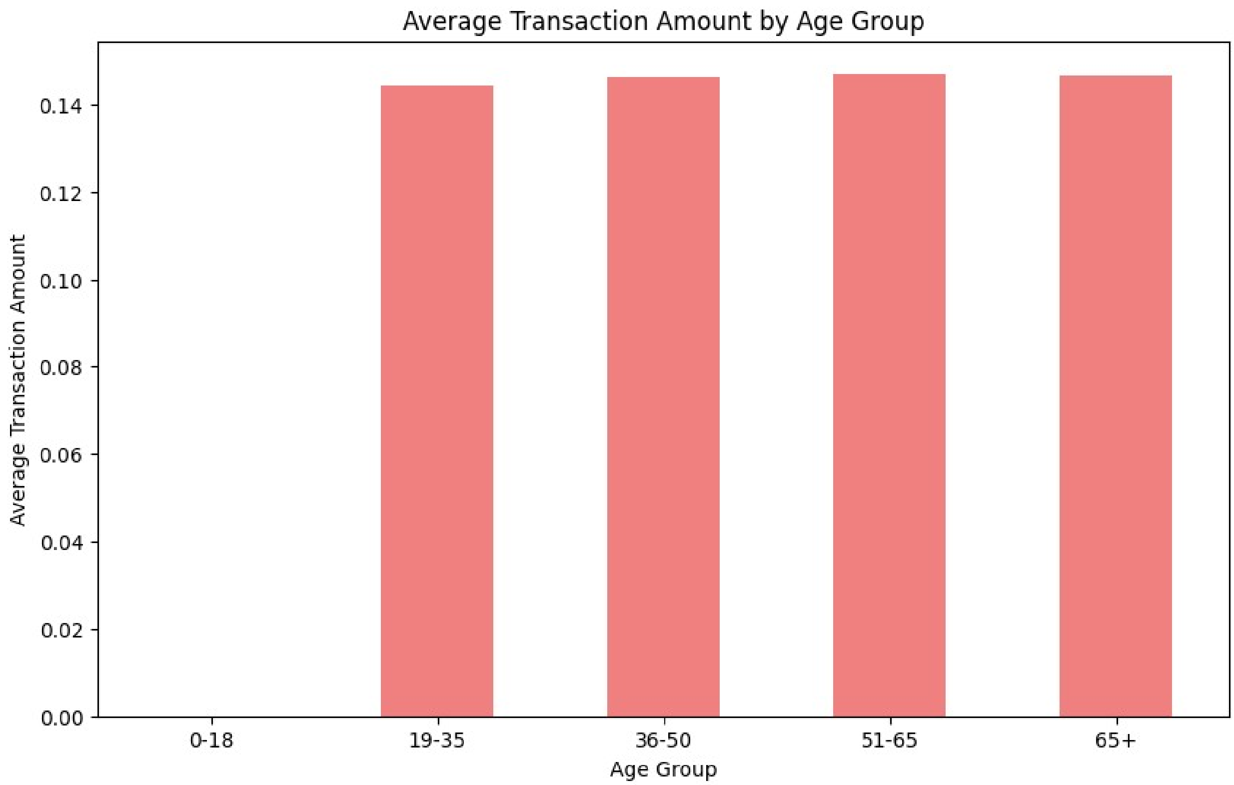
Structured Dataset Visualizations

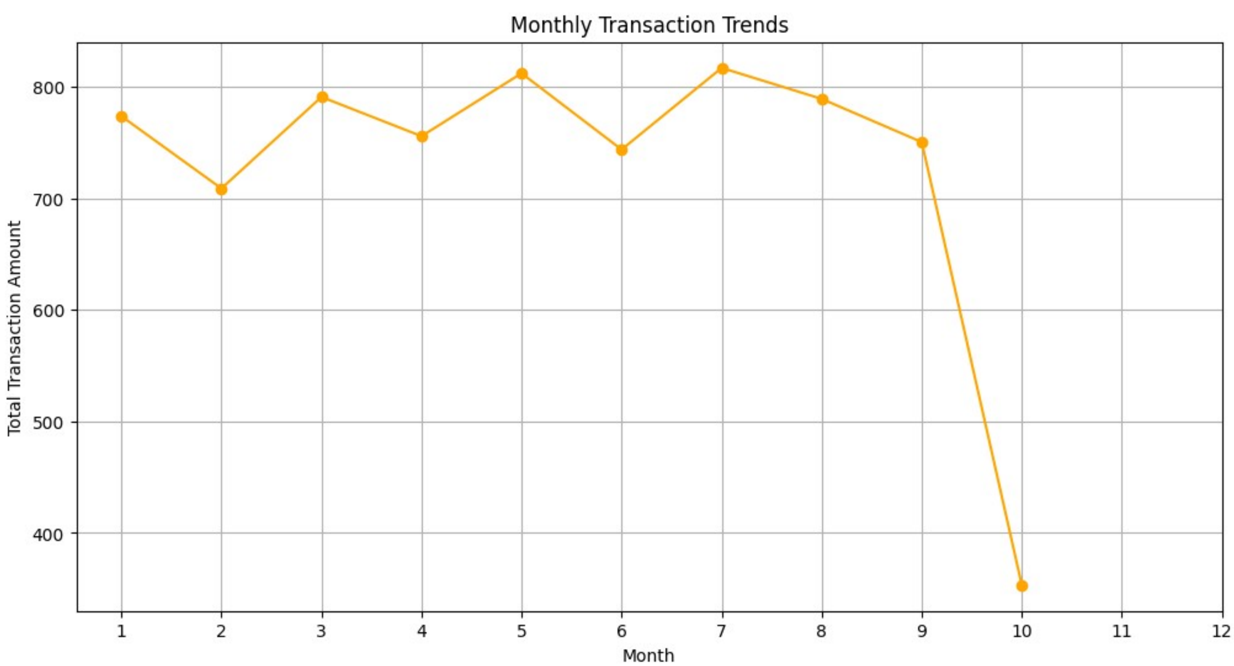
```
# Top Spending Categories Visualization
plt.figure(figsize=(10, 6))
top_categories.plot(kind='bar', color='skyblue')
plt.title('Top 5 Spending Categories')
plt.xlabel('Category')
plt.ylabel('Total Transaction Amount')
plt.xticks(rotation=45)
plt.show()
```

```python
# Average Transaction Amount by Gender
plt.figure(figsize=(10, 6))
avg_transaction_by_gender.plot(kind='bar', color='lightgreen')
plt.title('Average Transaction Amount by Gender')
plt.xlabel('Gender')
plt.ylabel('Average Transaction Amount')
plt.xticks(rotation=0)
plt.show()

# Average Transaction Amount by Age Group
plt.figure(figsize=(10, 6))
avg_transaction_by_age_group.plot(kind='bar', color='lightcoral')
plt.title('Average Transaction Amount by Age Group')
plt.xlabel('Age Group')
plt.ylabel('Average Transaction Amount')
plt.xticks(rotation=0)
plt.show()

# Monthly Transaction Trends
plt.figure(figsize=(12, 6))
monthly_trends.plot(kind='line', marker='o', color='orange')
plt.title('Monthly Transaction Trends')
plt.xlabel('Month')
plt.ylabel('Total Transaction Amount')
plt.xticks(range(1, 13))
plt.grid(True)
plt.show()
```


Top 5 Spending Categories
Total Transaction Amount
0
1000
2000
3000
4000
5
2
3
0
1
Category


Average Transaction Amount by Gender
Average Transaction Amount
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0.14
0
1
2
Gender

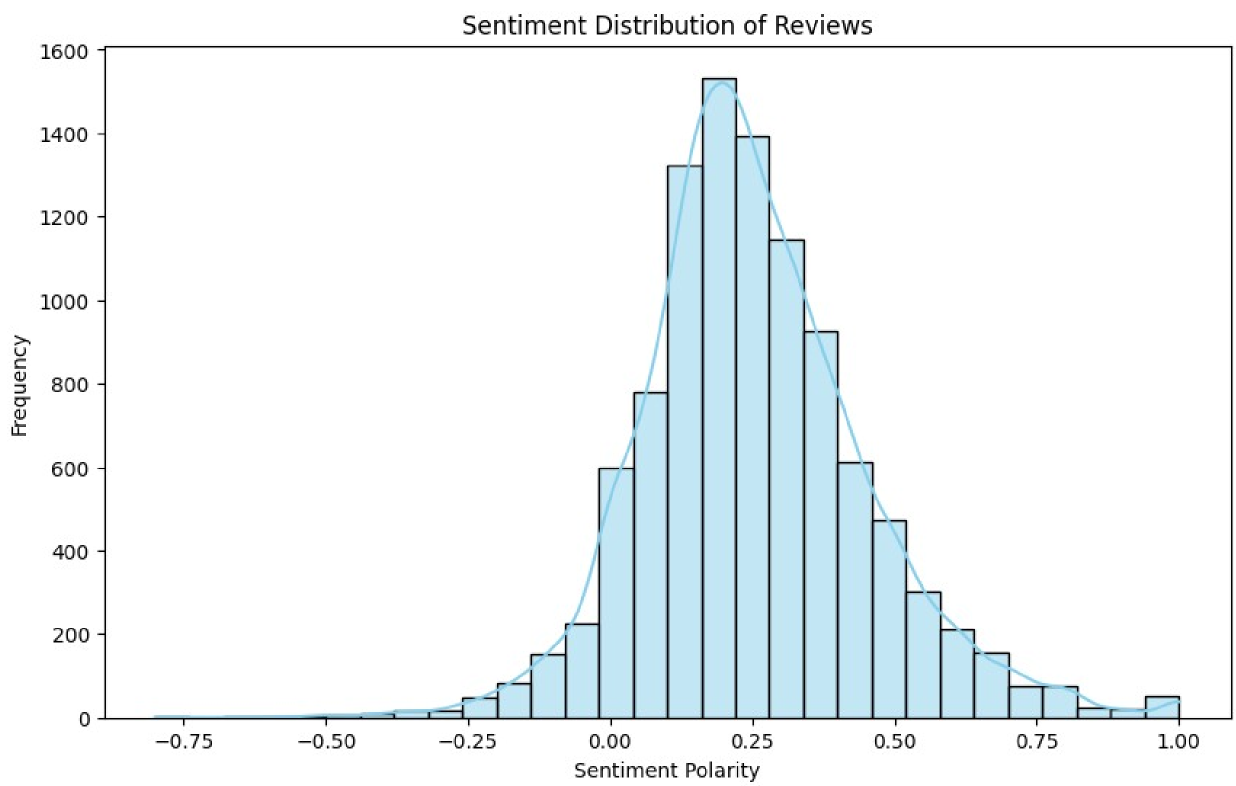


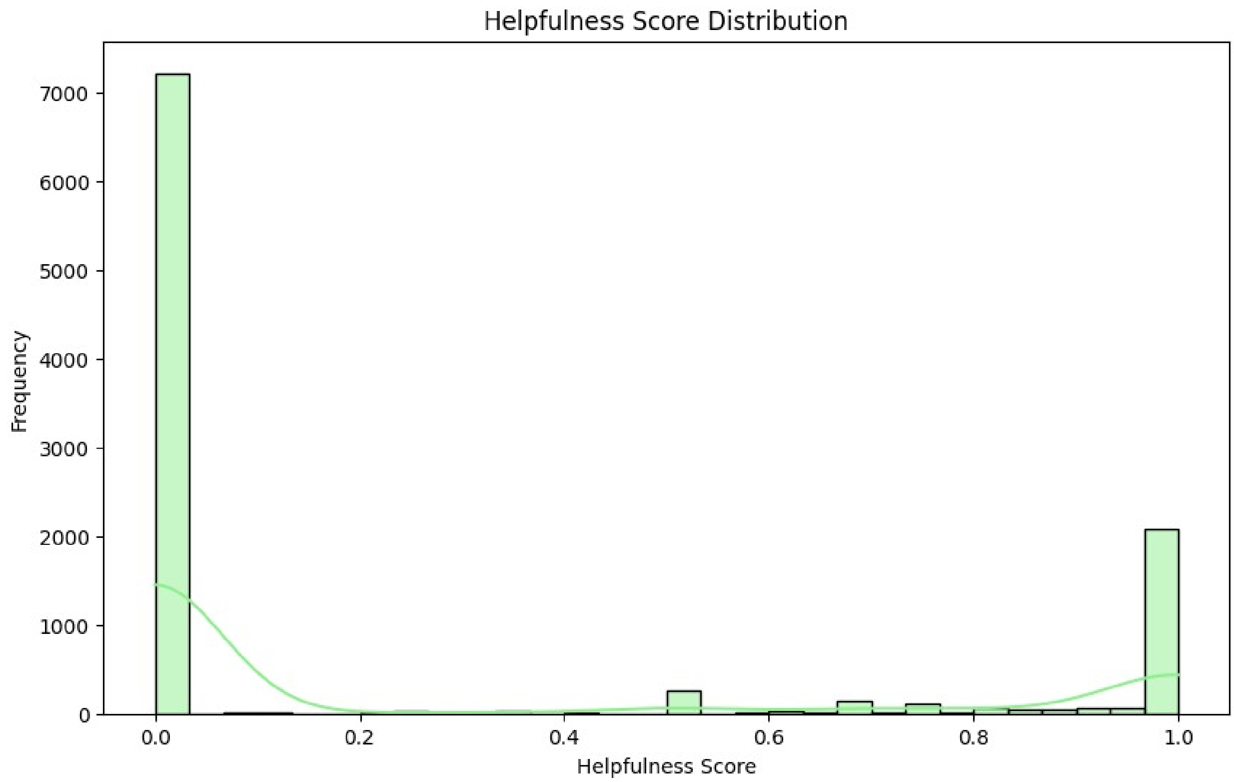


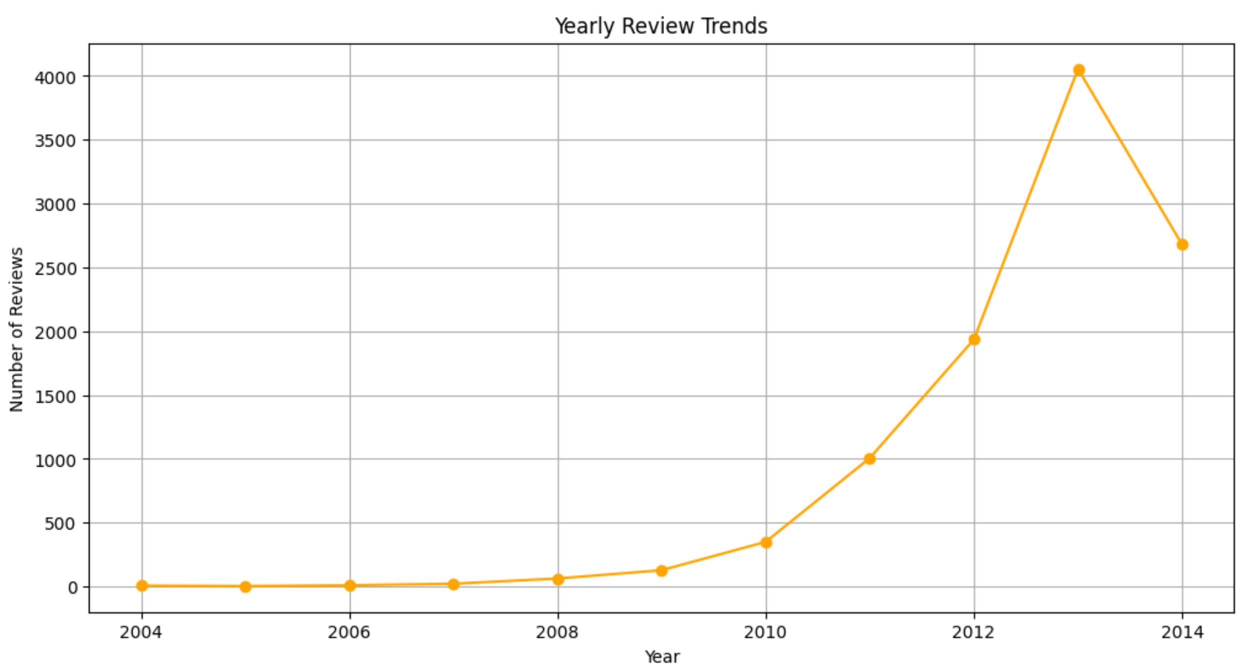
Unstructured Dataset Visualizations

```
# Sentiment Distribution Visualization
plt.figure(figsize=(10, 6))
sns.histplot(unstructured_data['sentiment'], kde=True, bins=30,
```

```
color='skyblue')
plt.title('Sentiment Distribution of Reviews')
plt.xlabel('Sentiment Polarity')
plt.ylabel('Frequency')
plt.show()

# Helpfulness Score Distribution
plt.figure(figsize=(10, 6))
sns.histplot(unstructured_data['helpfulness_score'], kde=True,
bins=30, color='lightgreen')
plt.title('Helpfulness Score Distribution')
plt.xlabel('Helpfulness Score')
plt.ylabel('Frequency')
plt.show()

# Yearly Review Trends
plt.figure(figsize=(12, 6))
yearly_review_trends.plot(kind='line', marker='o', color='orange')
plt.title('Yearly Review Trends')
plt.xlabel('Year')
plt.ylabel('Number of Reviews')
plt.grid(True)
plt.show()
```







Key Takeaways from Step 6

Structured Dataset: Spending is highest in Category 5. Age group 51-65 spends the most on average. Peak spending occurs in July and May. Unstructured Dataset: Sentiment is generally positive across reviews. Most reviews lack high helpfulness ratings. Review activity surged in 2013.

Ontology Creation

Ontology Creation Using rdflib

```
from rdflib import Graph, URIRef, Literal, RDF, Namespace
from rdflib.namespace import FOAF, XSD

# Create an RDF graph
g = Graph()

# Define a custom namespace
EX = Namespace("http://example.org/")

# Bind the namespace to a prefix
g.bind("ex", EX)

# Add triples for a few sample customers, products, and reviews

# Sample data from the structured dataset
sample_customer = structured_data.iloc[0]
customer_uri = URIRef(EX + f"Customer_{sample_customer['Customer
ID']}")
g.add((customer_uri, RDF.type, FOAF.Person))
g.add((customer_uri, EX.hasAge, Literal(sample_customer['Age'],
datatype=XSD.integer)))
g.add((customer_uri, EX.hasGender, Literal(sample_customer['Gender'],
datatype=XSD.integer)))

# Sample data from the unstructured dataset
sample_review = unstructured_data.iloc[0]
review_uri = URIRef(EX + f"Review_{sample_review['reviewerID']}")
product_uri = URIRef(EX + f"Product_{sample_review['asin']}")

g.add((review_uri, RDF.type, EX.Review))
g.add((review_uri, EX.hasSentiment,
Literal(sample_review['sentiment'], datatype=XSD.float)))
g.add((review_uri, EX.hasRating, Literal(sample_review['overall'],
datatype=XSD.integer)))

# Relationships
g.add((customer_uri, EX.writesReview, review_uri))
g.add((review_uri, EX.ratesProduct, product_uri))
g.add((customer_uri, EX.buysProduct, product_uri))

# Serialize the graph to a string in RDF/XML format
rdf_output = g.serialize(format='xml')

# Display RDF Output
print(rdf_output)
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF
   xmlns:ex="http://example.org/"
   xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
>
  <rdf:Description
rdf:about="http://example.org/Review_A2IBPI20UZIR0U">
    <rdf:type rdf:resource="http://example.org/Review"/>
    <ex:hasSentiment
rdf:datatype="http://www.w3.org/2001/XMLSchema#float">0.25</ex:hasSent
iment>
    <ex:hasRating
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">5</ex:hasRatin
g>
    <ex:ratesProduct
rdf:resource="http://example.org/Product_1384719342"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Customer_752858">
    <rdf:type rdf:resource="http://xmlns.com/foaf/0.1/Person"/>
    <ex:hasAge
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">22</ex:hasAge>
    <ex:hasGender
rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">0</ex:hasGende
r>
    <ex:writesReview
rdf:resource="http://example.org/Review_A2IBPI20UZIR0U"/>
    <ex:buysProduct
rdf:resource="http://example.org/Product_1384719342"/>
  </rdf:Description>
</rdf:RDF>
```


```python
# Save the RDF graph to a file
rdf_file_path = 'ontology_output.rdf'
g.serialize(destination=rdf_file_path, format='xml')

# Return the file path for user download
rdf_file_path
```


```json
{"type":"string"}
```