Project 1 - Chocolate Sales

April 8, 2025

[1]:

```
import pandas as pd
```

[2]:

```
dataset = pd.read_csv("C:/Users/Sam Fisher/Documents/Kaggle Datasets/Chocolate␣
  ↪Sales.csv")
```

[3]:

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1094 entries, 0 to 1093
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Sales Person   1094 non-null   object
 1   Country        1094 non-null   object
 2   Product        1094 non-null   object
 3   Date           1094 non-null   object
 4   Amount         1094 non-null   object
 5   Boxes Shipped  1094 non-null   int64
dtypes: int64(1), object(5)
memory usage: 51.4+ KB
```

[4]:

```
dataset.head()
```

[4]:

```
     Sales Person    Country              Product       Date    Amount  \
0  Jehu Rudeforth         UK       Mint Chip Choco  04-Jan-22   $5,320
1     Van Tuxwell      India         85% Dark Bars  01-Aug-22   $7,896
2    Gigi Bohling      India  Peanut Butter Cubes  07-Jul-22   $4,501
3    Jan Morforth  Australia  Peanut Butter Cubes  27-Apr-22  $12,726
4  Jehu Rudeforth         UK  Peanut Butter Cubes  24-Feb-22  $13,685

   Boxes Shipped
0            180
1             94
2             91
3            342
4            184
```

[5]:

```
dataset.describe() # summary stats for numerical columns
```

[5]:
```
       Boxes Shipped
count    1094.000000
mean      161.797989
std       121.544145
min         1.000000
25%        70.000000
50%       135.000000
75%       228.750000
max       709.000000
```

[6]:
```
# checking columns present-method 1
dataset.columns.tolist() # quick column list
```

[6]:
```
['Sales Person', 'Country', 'Product', 'Date', 'Amount', 'Boxes Shipped']
```

[7]:
```
# checking detailed view with datatype
dataset.info()
```

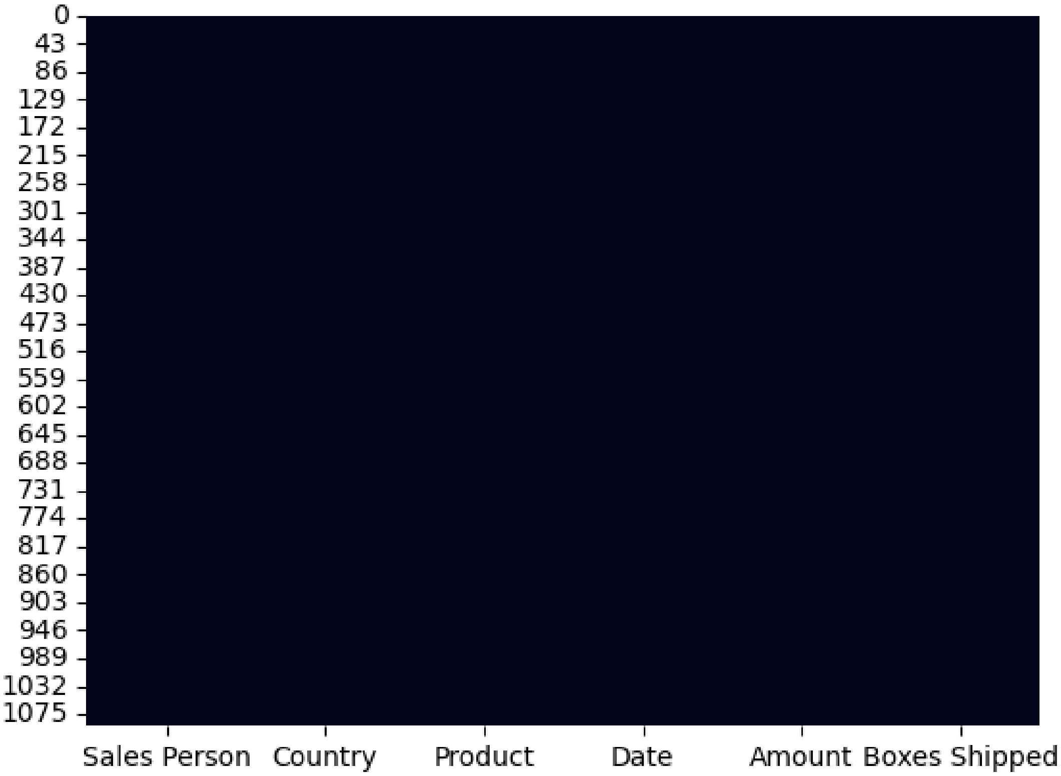
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1094 entries, 0 to 1093
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Sales Person   1094 non-null   object
 1   Country        1094 non-null   object
 2   Product        1094 non-null   object
 3   Date           1094 non-null   object
 4   Amount         1094 non-null   object
 5   Boxes Shipped  1094 non-null   int64
dtypes: int64(1), object(5)
memory usage: 51.4+ KB
```

[8]:
```
# count missing values per column
dataset.isnull().sum()
```

[8]:
```
Sales Person     0
Country          0
Product          0
Date             0
Amount           0
Boxes Shipped    0
dtype: int64
```

[9]:
```
# visualizing missing data
import seaborn as sns
sns.heatmap(dataset.isnull(), cbar=False)
```

[9]:
```
<Axes: >
```



[10]:
```
# checking for duplicate rows
print(f"total duplicates: {dataset.duplicated().sum()}")
```

```
total duplicates: 0
```

[11]:
```
# showing duplicate rows if present
print(dataset[dataset.duplicated(keep=False)])
```

```
Empty DataFrame
Columns: [Sales Person, Country, Product, Date, Amount, Boxes Shipped]
Index: []
```

[12]:
```
# finding time range
# convert 'data' column to 'datetime' if not already
dataset['Date'] = pd.to_datetime(dataset['Date'])
```

```
C:\Users\Sam Fisher\AppData\Local\Temp\ipykernel_8536\3776590236.py:3:
UserWarning: Could not infer format, so each element will be parsed
individually, falling back to `dateutil`. To ensure parsing is consistent and
as-expected, please specify a format.
  dataset['Date'] = pd.to_datetime(dataset['Date'])
```

[13]:
```
print(dataset['Date'].head())
```

```
0   2022-01-04
1   2022-08-01
2   2022-07-07
3   2022-04-27
4   2022-02-24
Name: Date, dtype: datetime64[ns]
```

[14]:

```
# finding time range
# convert 'data' column to 'datetime' if not already with format specified
dataset['Date'] = pd.to_datetime(dataset['Date'], format='%Y-%m-%d')
```

[15]:

```
# getting min/max dates
print(f"time range: {dataset['Date'].min()} to {dataset['Date'].max()}")
```

```
time range: 2022-01-03 00:00:00 to 2022-08-31 00:00:00
```

[16]:

```
# quick data summary
print(dataset[['Product', 'Amount']].describe())
```

```
        Product   Amount
count      1094     1094
unique       22      827
top     Eclairs  $2,317
freq         60        5
```

[17]:

```
# For categorical columns (product, region)

print(dataset['Product'].value_counts()) # top products
print(dataset['Country'].nunique()) # number of unique countries
```

```
Product
Eclairs                 60
50% Dark Bites          60
Smooth Sliky Salty      59
White Choc              58
Drinking Coco           56
Spicy Special Slims     54
Organic Choco Syrup     52
85% Dark Bars           50
Fruit & Nut Bars        50
After Nines             50
Peanut Butter Cubes     49
99% Dark & Pure         49
Milk Bars               49
Raspberry Choco         48
Almond Choco            48
Orange Choco            47
Mint Chip Choco         45
Manuka Honey Choco      45
```

```
Caramel Stuffed Bars    43
70% Dark Bites          42
Baker's Choco Chips     41
Choco Coated Almonds    39
Name: count, dtype: int64
6
```

[18]:

```python
print(dataset['Country'].nunique()) # number of unique countries
```

```
6
```

[19]:

```python
# checking for missing values
print(dataset.isnull().sum())
```

```
Sales Person     0
Country          0
Product          0
Date             0
Amount           0
Boxes Shipped    0
dtype: int64
```

[20]:

```python
# drop duplicates
dataset = dataset.drop_duplicates()
```

[21]:

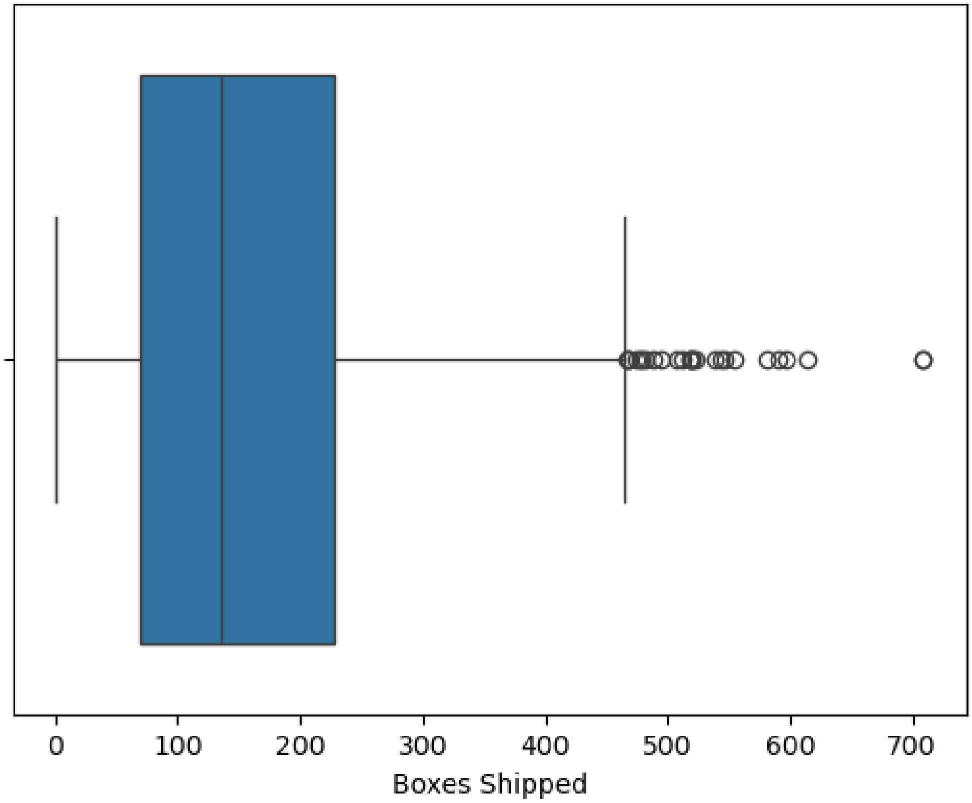
```python
# Handle missing values (example: fill with median for numerical columns)

dataset['Boxes Shipped'] = dataset['Boxes Shipped'].fillna(dataset['Boxes␣
  ↪Shipped'].median())
```

[22]:

```python
# Boxplot to spot outliers in sales

sns.boxplot(x=dataset['Boxes Shipped'])
```

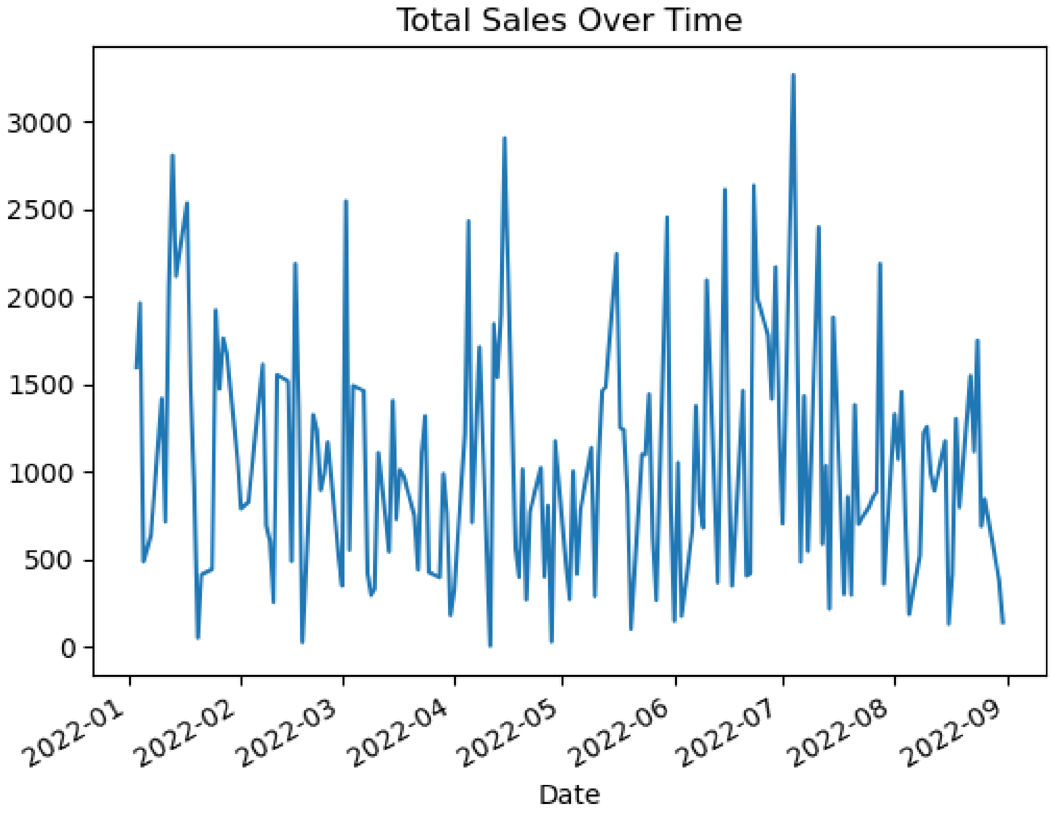
[22]:

```
<Axes: xlabel='Boxes Shipped'>
```



[23]:
```
# Total sales over time


dataset.groupby('Date')['Boxes Shipped'].sum().plot(title="Total Sales Over␣
 ↪Time")
```

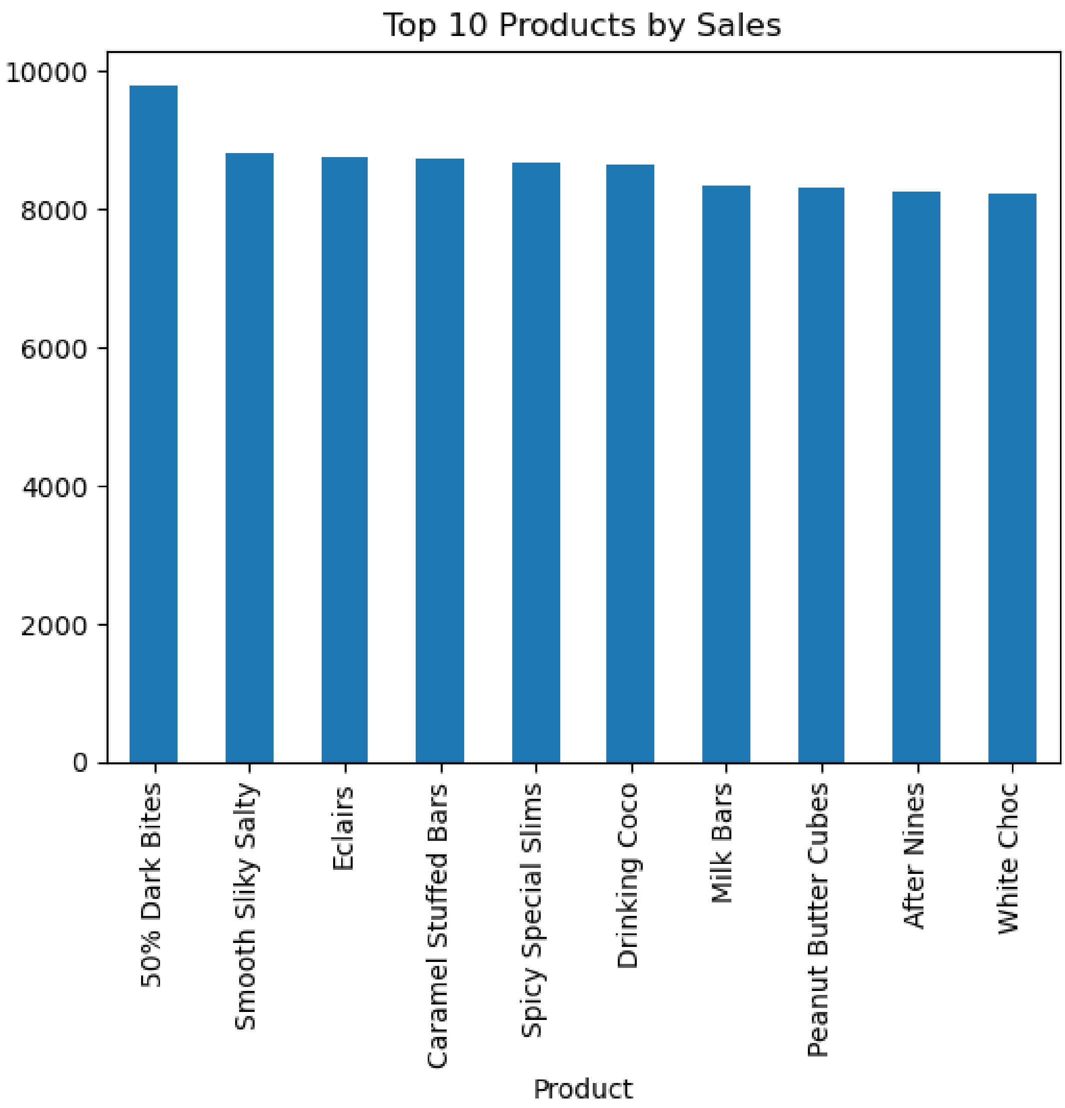
[23]:
```
<Axes: title={'center': 'Total Sales Over Time'}, xlabel='Date'>
```



```
[24]: # Top 10 products by sales

      top_products = dataset.groupby('Product')['Boxes Shipped'].sum().nlargest(10)
```

```
[25]: top_products.plot(kind='bar', title="Top 10 Products by Sales")
```

```
[25]: <Axes: title={'center': 'Top 10 Products by Sales'}, xlabel='Product'>
```



[26]:

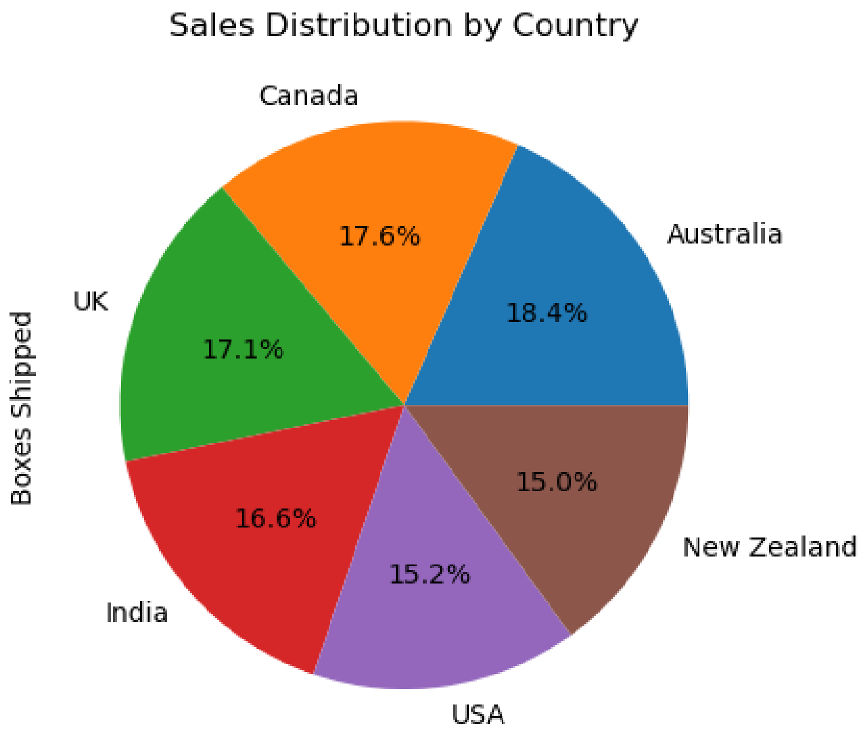
```
# Sales by country

country_sales = dataset.groupby('Country')['Boxes Shipped'].sum().
 ↪sort_values(ascending=False)
```

[27]:

```
country_sales.plot(kind='pie', autopct="%.1f%%", title="Sales Distribution by␣
 ↪Country")
```

[27]:

```
<Axes: title={'center': 'Sales Distribution by Country'}, ylabel='Boxes
Shipped'>
```

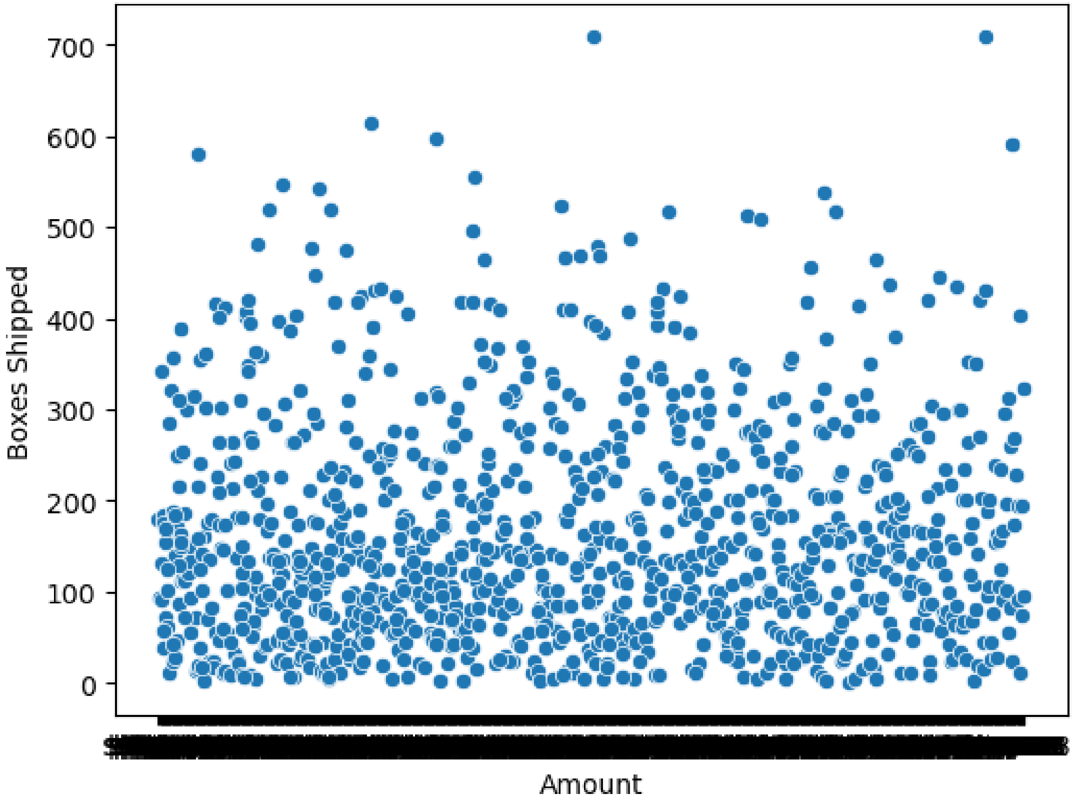


[28]:
```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1094 entries, 0 to 1093
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Sales Person   1094 non-null   object
 1   Country        1094 non-null   object
 2   Product        1094 non-null   object
 3   Date           1094 non-null   datetime64[ns]
 4   Amount         1094 non-null   object
 5   Boxes Shipped  1094 non-null   int64
dtypes: datetime64[ns](1), int64(1), object(4)
memory usage: 51.4+ KB
```

[29]:
```
# Scatter plot of price vs. sales

sns.scatterplot(x=dataset['Amount'], y=dataset['Boxes Shipped'])
```

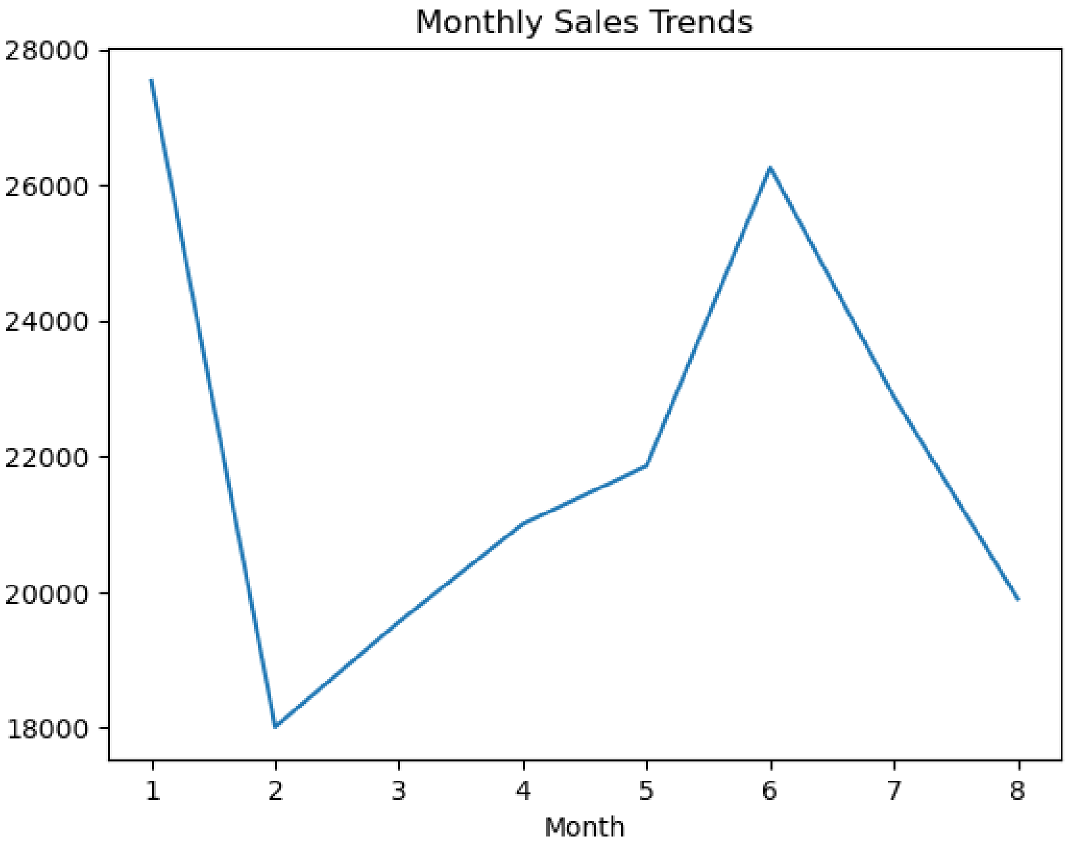
[29]:
```
<Axes: xlabel='Amount', ylabel='Boxes Shipped'>
```



```
[30]: # whether sales are higher during certain periods
      # dataset['Date'] = pd.to_datetime(dataset['Date']).dt.month # modifies␣
       ↪original date column

      dataset['Month'] = pd.to_datetime(dataset['Date']).dt.month # keeps original␣
       ↪date column unmodified and extracts month into a new column
```

```
[31]: # visualizing monthly sales trends
      dataset.groupby('Month')['Boxes Shipped'].sum().plot(kind='line',␣
       ↪title="Monthly Sales Trends")
```

```
[31]: <Axes: title={'center': 'Monthly Sales Trends'}, xlabel='Month'>
```



[32]:

```
# investigate which countries purchase the most expensive chocolate

dataset.groupby('Country')['Amount'].mean().sort_values(ascending=False)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\groupby\groupby.
 ↪py:1942, in GroupBy._agg_py_fallback(self, how, values, ndim, alt)
   1941 try:
-> 1942     res_values = self._grouper.agg_series(ser, alt, preserve_dtype=True
   1943 except Exception as err:

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\groupby\ops.py
 ↪864, in BaseGrouper.agg_series(self, obj, func, preserve_dtype)
    862     preserve_dtype = True
--> 864 result = self._aggregate_series_pure_python(obj, func)
    866 npvalues = lib.maybe_convert_objects(result, try_float=False)

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\groupby\ops.py
 ↪885, in BaseGrouper._aggregate_series_pure_python(self, obj, func)
```

```
    884 for i, group in enumerate(splitter):
--> 885     res = func(group)
    886     res = extract_result(res)

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\groupby\groupby.
 ↪py:2454, in GroupBy.mean.<locals>.<lambda>(x)
   2451 else:
   2452     result = self._cython_agg_general(
   2453         "mean",
-> 2454         alt=lambda x:␣
 ↪Series(x, copy=False).mean(numeric_only=numeric_only),
   2455         numeric_only=numeric_only,
   2456     )
   2457     return result.__finalize__(self.obj, method="groupby")

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\series.py:6549 ␣
 ↪in Series.mean(self, axis, skipna, numeric_only, **kwargs)
   6541 @doc(make_doc("mean", ndim=1))
   6542 def mean(
   6543     self,
   (…)
   6547     **kwargs,
   6548 ):
-> 6549     return NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\generic.py:
 ↪12420, in NDFrame.mean(self, axis, skipna, numeric_only, **kwargs)
  12413 def mean(
  12414     self,
  12415     axis: Axis | None = 0,
   (…)
  12418     **kwargs,
  12419 ) -> Series | float:
> 12420     return self._stat_function(
  12421         "mean", nanops.nanmean, axis, skipna, numeric_only, **kwargs
  12422     )

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\generic.py:
 ↪12377, in NDFrame._stat_function(self, name, func, axis, skipna, numeric_only ␣
 ↪**kwargs)
  12375 validate_bool_kwarg(skipna, "skipna", none_allowed=False)
> 12377 return self._reduce(
  12378     func, name=name, axis=axis, skipna=skipna, numeric_only=numeric_onl
  12379 )

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\series.py:6457 ␣
 ↪in Series._reduce(self, op, name, axis, skipna, numeric_only, filter_type,␣
 ↪**kwds)
```

```
   6453     raise TypeError(
   6454         f"Series.{name} does not allow {kwd_name}={numeric_only} "
   6455         "with non-numeric dtypes."
   6456     )
-> 6457 return op(delegate, skipna=skipna, **kwds)

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\nanops.py:147,␣
 ↪in bottleneck_switch.__call__.<locals>.f(values, axis, skipna, **kwds)
    146 else:
--> 147     result = alt(values, axis=axis, skipna=skipna, **kwds)
    149 return result

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\nanops.py:404,␣
 ↪in _datetimelike_compat.<locals>.new_func(values, axis, skipna, mask, **kwargs)
    402     mask = isna(values)
--> 404 result = func(values, axis=axis, skipna=skipna, mask=mask, **kwargs)
    406 if datetimelike:

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\nanops.py:720,␣
 ↪in nanmean(values, axis, skipna, mask)
    719 the_sum = values.sum(axis, dtype=dtype_sum)
--> 720 the_sum = _ensure_numeric(the_sum)
    722 if axis is not None and getattr(the_sum, "ndim", False):

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\nanops.py:1701 ␣
 ↪in _ensure_numeric(x)
   1699 if isinstance(x, str):
   1700     # GH#44008, GH#36703 avoid casting e.g. strings to numeric
-> 1701     raise TypeError(f"Could not convert string '{x}' to numeric")
   1702 try:

TypeError: Could not convert string '$12,726 $3,080 $2,835 $6,790 $6,888 $7,672␣
 ↪$4,284 $3,654 $6,979 $8,575 $91 $15,421 $4,438 $1,603 $273 $2,030 $19,453 $28␣
 ↪$5,859 $7,182 $6,881 $1,743 $1,827 $5,740 $5,579 $623 $6,013 $11,550 $7,273␣
 ↪$8,897 $2,464 $2,765 $4,116 $12,516 $2,758 $6,048 $854 $2,779 $1,043 $5,194␣
 ↪$13,706 $8,113 $7,287 $3,472 $3,325 $3,472 $9,660 $7,357 $5,124 $735 $3,199␣
 ↪$3,136 $5,460 $7,161 $7,910 $3,108 $7,350 $3,752 $3,192 $3,745 $14,658 $2,807␣
 ↪$2,240 $6,979 $392 $7,294 $14,889 $2,058 $2,541 $5,523 $7,882 $6,832 $3,010␣
 ↪$6,916 $602 $5,936 $2,912 $1,575 $5,691 $3,178 $4,676 $2,317 $6,790 $6,797␣
 ↪$4,466 $4,669 $7,490 $6,993 $637 $6,034 $5,775 $13,125 $994 $1,043 $3,402␣
 ↪$10,507 $238 $7,672 $4,186 $7,406 $2,611 $8,001 $6,678 $5,222 $6,706 $7,434␣
 ↪$2,751 $2,786 $2,303 $12,271 $11,298 $6,342 $3,185 $8,225 $4,102 $11,116␣
 ↪$13,076 $8,715 $4,046 $4,396 $5,439 $1,435 $679 $10,486 $17,626 $8,757 $10,03␣
 ↪$12,565 $504 $2,961 $1,981 $7,959 $10,794 $6,944 $3,171 $112 $6,223 $3,969␣
 ↪$5,810 $4,403 $5,796 $6,713 $10,031 $6,678 $2,933 $6,524 $15,750 $910 $8,659␣
 ↪$3,087 $3,605 $8,498 $700 $644 $7,910 $1,456 $9,744 $63 $2,821 $6,916 $8,995␣
 ↪$7,252 $329 $3,192 $4,326 $9,527 $4,879 $1,372 $5,012 $2,303 $13,258 $721␣
 ↪$9,114 $7,091 $9,268 $1,645 $7,063 $4,200 $6,832 $6,321 $3,906 $5,768 $994␣
 ↪$574 $938 $4,879 $10,199 $11,389 $10,822 $4,158 $4,263 $13,846 $2,226 $5,250␣
 ↪$8,400 $1,288 $3,647 $7,952 $1,470 $2,674 $6,818 $3,710 $6,055 $301 $4,410 '␣
 ↪to numeric

The above exception was the direct cause of the following exception:
```

```
TypeError                                 Traceback (most recent call last)
Cell In[32], line 3
      1 # investigate which countries purchase the most expensive chocolate
----> 3 dataset.groupby('Country')['Amount'].mean().sort_values(ascending=False

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\groupby\groupb .
 ↪py:2452, in GroupBy.mean(self, numeric_only, engine, engine_kwargs)
   2445     return self._numba_agg_general(
   2446         grouped_mean,
   2447         executor.float_dtype_mapping,
   2448         engine_kwargs,
   2449         min_periods=0,
   2450     )
   2451 else:
-> 2452     result = self._cython_agg_general(
   2453         "mean",
   2454␣
 ↪        alt=lambda x: Series(x, copy=False).mean(numeric_only=numeric_only),
   2455         numeric_only=numeric_only,
   2456     )
   2457     return result.__finalize__(self.obj, method="groupby")

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\groupby\groupb .
 ↪py:1998, in GroupBy._cython_agg_general(self, how, alt, numeric_only,␣
 ↪min_count, **kwargs)
   1995     result = self._agg_py_fallback(how, values, ndim=data.ndim, alt=alt
   1996     return result
-> 1998 new_mgr = data.grouped_reduce(array_func)
   1999 res = self._wrap_agged_manager(new_mgr)
   2000 if how in ["idxmin", "idxmax"]:

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\internals\base
 ↪py:367, in SingleDataManager.grouped_reduce(self, func)
    365 def grouped_reduce(self, func):
    366     arr = self.array
--> 367     res = func(arr)
    368     index = default_index(len(res))
    370     mgr = type(self).from_array(res, index)

File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\groupby\groupb .
 ↪py:1995, in GroupBy._cython_agg_general.<locals>.array_func(values)
   1992     return result
   1994 assert alt is not None
-> 1995 result = self._agg_py_fallback(how, values, ndim=data.ndim, alt=alt)
   1996 return result
```

```
File C:\Miniconda\envs\pydata-book\lib\site-packages\pandas\core\groupby\groupb .
 ↪py:1946, in GroupBy._agg_py_fallback(self, how, values, ndim, alt)
   1944     msg = f"agg function failed [how->{how},dtype->{ser.dtype}]"
   1945     # preserve the kind of exception that raised
-> 1946     raise type(err)(msg) from err
   1948 if ser.dtype == object:
   1949     res_values = res_values.astype(object, copy=False)

TypeError: agg function failed [how->mean,dtype->object]
```

[ ]:
```
# converting 'Amount' column to numeric, forcing errors to NaN
dataset['Amount'] = pd.to_numeric(dataset['Amount'], errors='coerce')
```

[ ]:
```
# now, group by 'Country' and calculate the mean of 'Amount'
dataset.groupby('Country')['Amount'].mean().sort_values(ascending=False)
```

[ ]:
```
# investigate which countries purchase the highest numbers

dataset.groupby('Country')['Boxes Shipped'].mean().sort_values(ascending=False)
```

[ ]:
```
dataset.head()
```

[ ]:
```
dataset = pd.read_csv("C:/Users/Sam Fisher/Documents/Kaggle Datasets/Chocolate␣
 ↪Sales.csv") # reverting back to original 'Amount' column
```

[ ]:
```
dataset.head()
```

[ ]:
```
# investigate which countries purchase the highest numbers

mean_amount_by_country = dataset.groupby('Country')['Boxes Shipped'].mean().
 ↪sort_values(ascending=False)
```

[ ]:
```
mean_amount_by_country
```

[ ]:
```
# replacing dollar sign and commas from amount column to convert to numeric␣
 ↪values

dataset['Price'] = pd.to_numeric(dataset['Amount'].replace({'\$': '', ',': ''},␣
 ↪regex=True), errors='coerce')
```

[ ]:
```
print(dataset['Price'].head())  # check the first few values
```

[ ]:
```
print(dataset['Price'].dtype)   # ensure the column is numeric (e.g., float64)
```

[ ]:
```
# whether higher priced product sell less

sns.boxplot(x='Product', y='Price', data=dataset)
```

[ ]:

```
from statsmodels.tsa.arima.model import ARIMA

# Aggregate daily sales to monthly
monthly_sales = dataset.groupby(pd.to_datetime(dataset['Date'], format='mixed').
 ↪dt.to_period('M'))['Boxes Shipped'].sum()

# Fit ARIMA model
model = ARIMA(monthly_sales, order=(1,1,1))
results = model.fit()
results.predict(start=0, end=24)  # Forecast 2 years
```

[33]:

```
dataset.head()
```

[33]:

```
     Sales Person    Country              Product        Date    Amount  \
0  Jehu Rudeforth         UK       Mint Chip Choco 2022-01-04    $5,320
1      Van Tuxwell      India        85% Dark Bars 2022-08-01    $7,896
2     Gigi Bohling      India  Peanut Butter Cubes 2022-07-07    $4,501
3     Jan Morforth  Australia  Peanut Butter Cubes 2022-04-27   $12,726
4  Jehu Rudeforth         UK  Peanut Butter Cubes 2022-02-24   $13,685

   Boxes Shipped  Month
0            180      1
1             94      8
2             91      7
3            342      4
4            184      2
```

[34]:

```
# clean amount column

dataset['Amount'] = pd.to_numeric(dataset['Amount'].replace('[\$,]', '',␣
 ↪regex=True)).astype(float)
```

[35]:

```
# converting date to datetime

dataset['Date'] = pd.to_datetime(dataset['Date'], format='%d-%b-%y')
```

[36]:

```
# add month, year columns

dataset['Month'] = dataset['Date'].dt.month_name()
dataset['Year'] = dataset['Date'].dt.year
```

[37]:

```
print(dataset.describe()) # basic stats
```

```
                                Date       Amount  Boxes Shipped    Year
count                           1094  1094.000000    1094.000000  1094.0
mean   2022-05-03 09:04:56.160877568  5652.308044     161.797989  2022.0
min              2022-01-03 00:00:00     7.000000       1.000000  2022.0
25%              2022-03-02 00:00:00  2390.500000      70.000000  2022.0
```

```
50%            2022-05-11 00:00:00   4868.500000    135.000000  2022.0
75%            2022-07-04 00:00:00   8027.250000    228.750000  2022.0
max            2022-08-31 00:00:00  22050.000000    709.000000  2022.0
std                            NaN   4102.442014    121.544145     0.0
```

[38]:
```python
# top sales persons
top_salespeople = dataset.groupby('Sales Person')['Amount'].sum().nlargest(5)
```

[39]:
```python
top_salespeople
```

[39]:
```
Sales Person
Ches Bonnell        320901.0
Oby Sorrel          316645.0
Madelene Upcott     316099.0
Brien Boise         312816.0
Kelci Walkden       311710.0
Name: Amount, dtype: float64
```

[40]:
```python
# top products
top_products = dataset.groupby('Product')['Amount'].sum().nlargest(5)
```

[41]:
```python
top_products
```

[41]:
```
Product
Smooth Sliky Salty      349692.0
50% Dark Bites          341712.0
White Choc              329147.0
Peanut Butter Cubes     324842.0
Eclairs                 312445.0
Name: Amount, dtype: float64
```

[42]:
```python
# top countries
top_countries = dataset.groupby('Country')['Amount'].sum().nlargest(5)
```

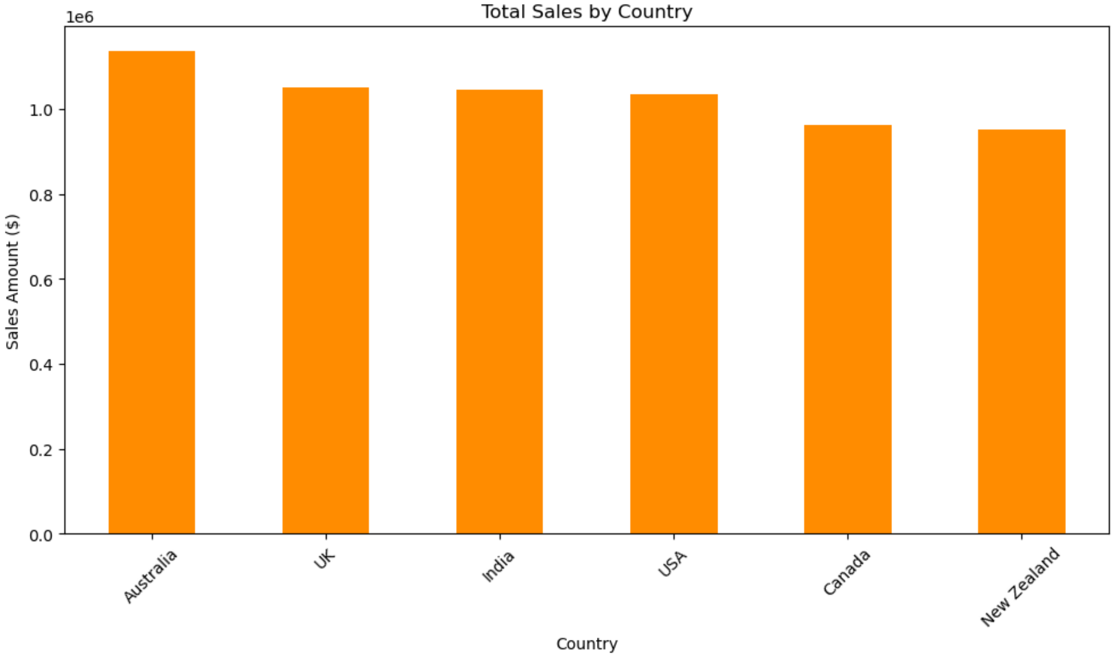
[43]:
```python
top_countries
```

[43]:
```
Country
Australia    1137367.0
UK           1051792.0
India        1045800.0
USA          1035349.0
Canada        962899.0
Name: Amount, dtype: float64
```

[45]:
```python
import matplotlib.pyplot as plt
# sales performance by country
```

```
country_sales = dataset.groupby('Country')['Amount'].sum().
 ↪sort_values(ascending=False)
print(f"Country Sales Distribution:\n{country_sales}")

plt.figure(figsize=(10,6))
country_sales.plot(kind='bar', color='darkorange')
plt.title('Total Sales by Country')
plt.ylabel('Sales Amount ($)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('country_sales.png')
plt.show()
```

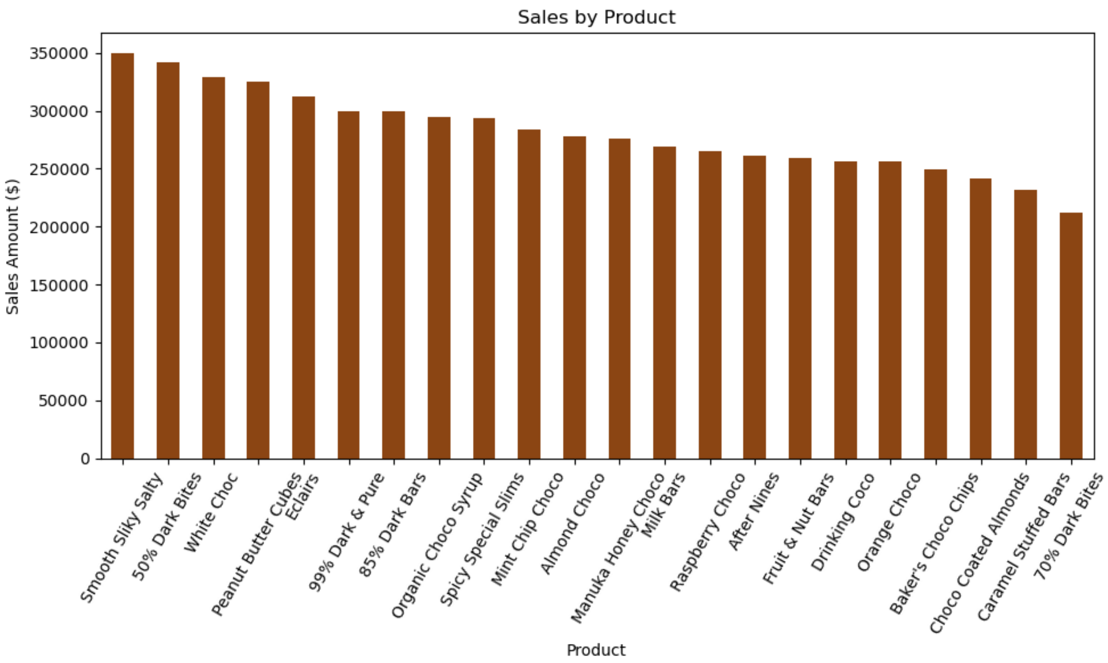
```
Country Sales Distribution:
Country
Australia      1137367.0
UK             1051792.0
India          1045800.0
USA            1035349.0
Canada          962899.0
New Zealand     950418.0
Name: Amount, dtype: float64
```



[48]:
```
# product performance
```

```python
product_performance = dataset.groupby('Product')['Amount'].sum().
 ↪sort_values(ascending=False)
print(f"\nProduct Performance:\n{product_performance}")

plt.figure(figsize=(10,6))
product_performance.plot(kind='bar', color='saddlebrown')
plt.title('Sales by Product')
plt.ylabel('Sales Amount ($)')
plt.xticks(rotation=60)
plt.tight_layout()
plt.savefig('product_sales.png')
plt.show()
```

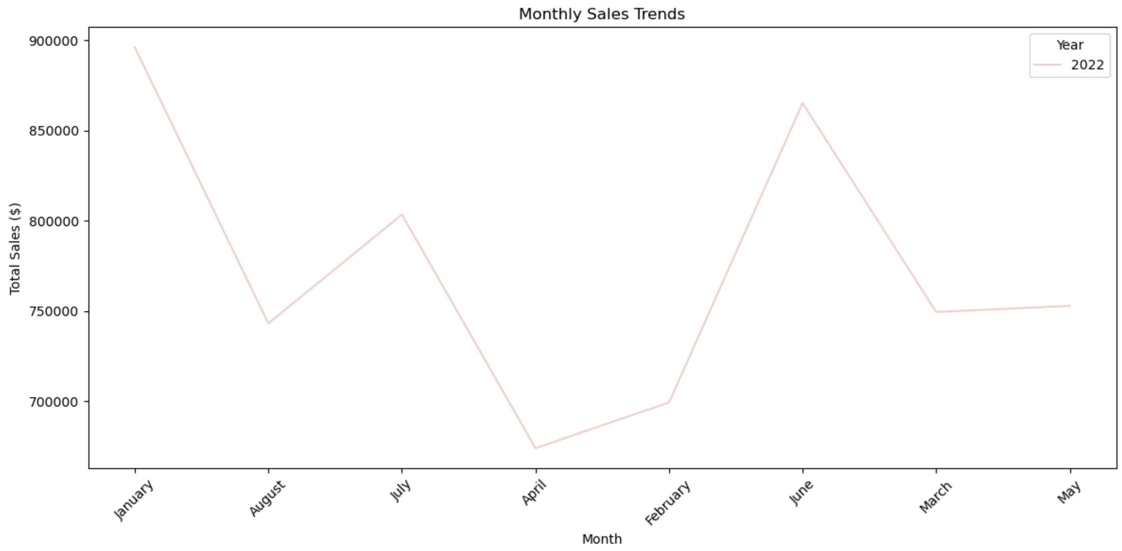
```
Product Performance:
Product
Smooth Sliky Salty      349692.0
50% Dark Bites          341712.0
White Choc              329147.0
Peanut Butter Cubes     324842.0
Eclairs                 312445.0
99% Dark & Pure         299796.0
85% Dark Bars           299229.0
Organic Choco Syrup     294700.0
Spicy Special Slims     293454.0
Mint Chip Choco         283969.0
Almond Choco            277536.0
Manuka Honey Choco      275541.0
Milk Bars               269248.0
Raspberry Choco         264740.0
After Nines             261331.0
Fruit & Nut Bars        259147.0
Drinking Coco           256655.0
Orange Choco            256144.0
Baker's Choco Chips     249613.0
Choco Coated Almonds    241486.0
Caramel Stuffed Bars    231588.0
70% Dark Bites          211610.0
Name: Amount, dtype: float64
```



[50]:

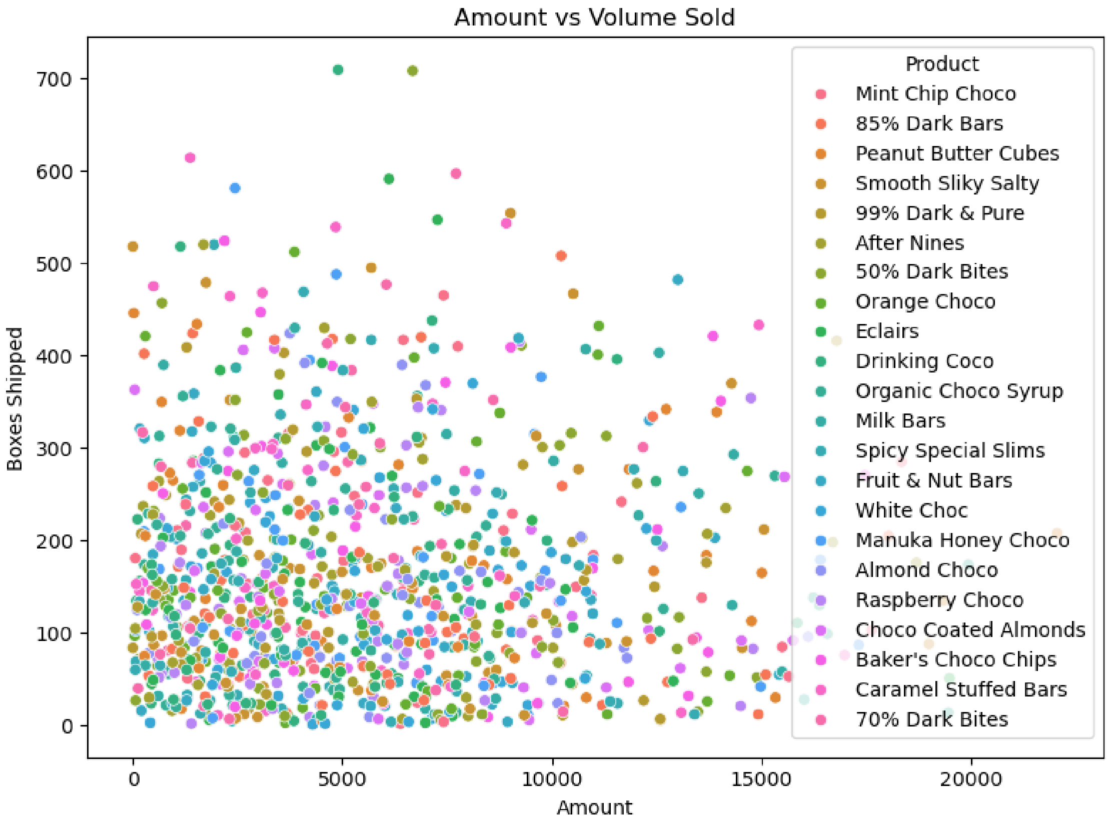
```
# time series analysis

monthly_sales = dataset.groupby(['Year', 'Month'])['Amount'].sum().unstack()
print("\nMonthly Sales Trends:")
print(monthly_sales)


plt.figure(figsize=(12,6))
sns.lineplot(data=dataset, x='Month', y='Amount', hue='Year', estimator='sum',␣
  ↪errorbar=None)
plt.title('Monthly Sales Trends')
plt.ylabel('Total Sales ($)')
plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('monthly_trends.png')
plt.show()
```

```
Monthly Sales Trends:
Month     April    August  February   January      July      June     March  \
Year
2022   674051.0  743148.0  699377.0  896105.0  803425.0  865144.0  749483.0

Month       May
Year
2022   752892.0
```



[51]:

```
# amount and sales relationship

plt.figure(figsize=(8,6))
sns.scatterplot(data=dataset, x='Amount', y='Boxes Shipped', hue='Product')
plt.title('Amount vs Volume Sold')
plt.tight_layout()
plt.savefig('amount_vs_volume.png')
plt.show()
```


Amount vs Volume Sold
Boxes Shipped
Amount
0
100
200
300
400
500
600
700
0
5000
10000
15000
20000
Product
Mint Chip Choco
85% Dark Bars
Peanut Butter Cubes
Smooth Sliky Salty
99% Dark & Pure
After Nines
50% Dark Bites
Orange Choco
Eclairs
Drinking Coco
Organic Choco Syrup
Milk Bars
Spicy Special Slims
Fruit & Nut Bars
White Choc
Manuka Honey Choco
Almond Choco
Raspberry Choco
Choco Coated Almonds
Baker's Choco Chips
Caramel Stuffed Bars
70% Dark Bites

```
[ ]:
```