```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib notebook
```

```
data = pd.read_csv('x1.csv')
```

```
data.head()
```

	ASIN	Brand	Title	Category	BuyBox	#	Fulfillment	Price	FBA fee	Sales	Revenue	BSR	Rating	Re C
0	B01CV2WJS0	Abbott	FreeStyle Lite Blood Glucose Test Strips - 50 ...	Health & Household	A 2 Z STORE	5	FBA	279.99	45.0	48	13,439.52	104,791	5.0	
1	B08BG5HPPQ	McKesson	McKesson TRUE METRIX Blood Glucose Test Strips...	Health & Household	SimplyMedical	2	NaN	219.81	0.0	NaN	NaN	NaN	4.5	
2	B078YGC2XQ	ABBOTT DIABETES CARE	FreeStyle Lite Blood Glucose Test Strips - 50 ...	Health & Household	Accordo Health and Groceries	4	FBA	184.90	31.0	206	38,089.40	51,877	4.5	
3	B078YGC2XQ	ABBOTT DIABETES CARE	FreeStyle Lite Blood Glucose Test Strips - 50 ...	Health & Household	Accordo Health and Groceries	4	FBA	184.90	31.0	206	38,089.40	51,877	4.5	
4	B078YGC2XQ	ABBOTT DIABETES CARE	FreeStyle Lite Blood Glucose Test Strips - 50 ...	Health & Household	Accordo Health and Groceries	4	FBA	184.90	31.0	206	38,089.40	51,877	4.5	

```
plt.figure()
sns.heatmap(data.isnull(), yticklabels=False, cbar=False) #white is the missing data in columns
plt.show()
```

```
data.info()
```

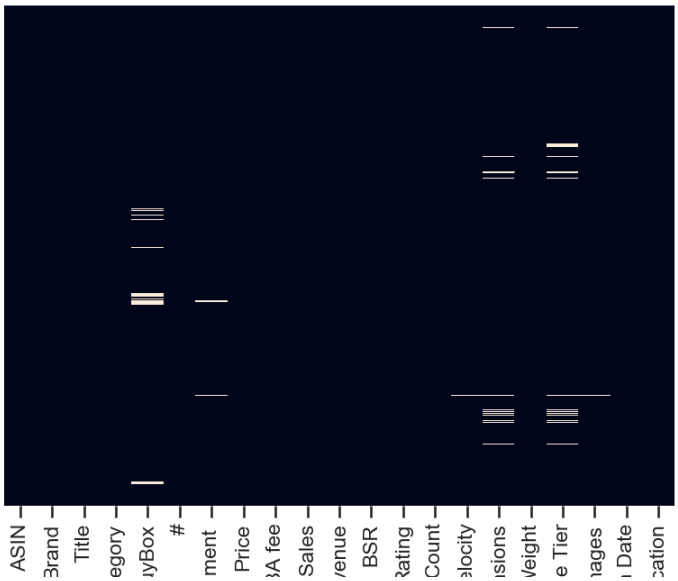
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 826 entries, 0 to 825
Data columns (total 20 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   ASIN                  826 non-null    object 
 1   Brand                 826 non-null    object 
 2   Title                 826 non-null    object 
 3   Category              826 non-null    object 
 4   BuyBox                802 non-null    object 
 5   #                     826 non-null    int64  
 6   Fulfillment           819 non-null    object 
 7   Price                 826 non-null    float64
 8   FBA fee               826 non-null    float64
 9   Sales                 796 non-null    object 
 10  Revenue               796 non-null    object 
 11  BSR                   800 non-null    object 
 12  Rating                826 non-null    float64
 13  Review Count          826 non-null    object 
 14  Review Velocity       821 non-null    float64
 15  Dimensions            805 non-null    object 
 16  Weight                799 non-null    float64
 17  Size Tier             799 non-null    object 
 18  Images                821 non-null    float64
 19  Listing Creation Date  826 non-null    object 
dtypes: float64(6), int64(1), object(13)
memory usage: 129.2+ KB
```

```
data['Clasification']= ''
for i in range(len(data)):
    if "($)" in (data.iloc[i]['Title']):
        data.at[i, 'Clasification'] ='Sponsored'

    else:
        data.at[i, 'Clasification'] ='None'
```

```
data['Revenue'] = data['Revenue'].str.replace(',','').astype('float')
data['Sales'] = data['Sales'].str.replace(',', '').astype('float')
data['BSR'] = data['BSR'].str.replace(',', '').astype('float')
data['Review Count'] = data['Review Count'].str.replace(',', '').astype('float')
#data['Price'] = data['Price'].str.replace(',','').astype('float')
```

```
data['Sales'] = data['Sales'].fillna((data['Sales'].mean()))
data['Revenue'] = data['Revenue'].fillna((data['Revenue'].mean()))
data['BSR'] = data['BSR'].fillna((data['BSR'].mean()))
data['Weight'] = data['Weight'].fillna((data['Weight'].mean()))
```

```
plt.figure()
sns.heatmap(data.isnull(), yticklabels=False, cbar=False) #white is the missing data in columns
plt.show()
```




```
newData = data.drop_duplicates(subset = ["ASIN"])
```

```
newData.describe()
```

	#	Price	FBA fee	Sales	Revenue	BSR	Rating	Review Count	Review Velocity	We
count	198.000000	198.000000	198.000000	198.000000	198.000000	198.000000	198.000000	198.000000	196.000000	198.00
mean	4.106061	31.710051	7.185455	920.528615	25797.175516	55078.369394	4.500000	588.722222	69.673469	0.28
std	4.045912	29.448224	4.507198	1829.801796	57897.819273	73885.422994	0.780472	1076.107291	158.135273	0.25
min	1.000000	5.990000	0.000000	0.000000	0.000000	1230.000000	0.000000	0.000000	-105.000000	0.01
25%	1.000000	16.990000	5.000000	173.750000	3849.820000	17387.000000	4.500000	82.250000	8.000000	0.10
50%	3.000000	24.990000	6.770000	355.000000	9411.895000	37018.500000	4.500000	240.500000	23.000000	0.25
75%	5.000000	39.515000	9.000000	923.500000	28070.955000	64918.750000	5.000000	672.000000	68.750000	0.40
max	21.000000	279.990000	45.000000	15974.000000	608449.660000	814721.000000	5.000000	9061.000000	1272.000000	1.55

```
newData['Brand'].nunique()
```

```
103
```

```
newData['ASIN'].count()
```

```
198
```

```
newData['Price'].dropna(axis=0)
```

```
0      279.99
1      219.81
2      184.90
7       30.99
14      26.99
        ...
801      6.99
802      6.99
803      6.99
819      6.95
823     42.99
Name: Price, Length: 198, dtype: float64
```

```
newData['Price'].mean()
```

```
31.710050505050404
```

```
newData['Price'].dropna(axis=0)
```

```
0      279.99
1      219.81
2      184.90
7       30.99
14      26.99
        ...
801      6.99
802      6.99
803      6.99
819      6.95
823     42.99
Name: Price, Length: 198, dtype: float64
```

```
newData.head()
```

	ASIN	Brand	Title	Category	BuyBox	#	Fulfillment	Price	FBA fee	Sales	...	BSR	Rat
0	B01CV2WJS0	Abbott	FreeStyle Lite Blood Glucose Test Strips - 50 ...	Health & Household	A 2 Z STORE	5	FBA	279.99	45.0	48.000000	...	104791.0000	
1	B08BG5HPPQ	McKesson	McKesson TRUE METRIX Blood Glucose Test Strips...	Health & Household	SimplyMedical	2	NaN	219.81	0.0	1311.851759	...	40119.7675	
2	B078YGC2XQ	ABBOTT DIABETES CARE	FreeStyle Lite Blood Glucose Test Strips - 50 ...	Health & Household	Accordo Health and Groceries	4	FBA	184.90	31.0	206.000000	...	51877.0000	
7	B076VSN7TR	Care Touch	Care Touch Diabetes Testing Kit – Care Touch B...	Health & Household	Amazon Warehouse	3	FBA	30.99	8.0	12892.000000	...	1641.0000	
14	B07QQC4R1K	($) Metene	($) Diabetes Testing Kit, 50 Lancets, 50 Gluco...	($) Health & Household	($) metene	1	FBA	26.99	7.0	6458.000000	...	2929.0000	

5 rows × 21 columns

```
price=newData.groupby('Clasification')['Price'].mean().sort_values(ascending=False)
price_df=pd.DataFrame({'Product Type':price.index, 'Average Price':price.values})
price_df.head(10)
```

	Product Type	Average Price
0	None	33.036039
1	Sponsored	27.069091

```
onlyTrue = newData.copy()
onlyTrue['Price'].describe()
```

```
count    198.000000
mean      31.710051
std       29.448224
min        5.990000
25%       16.990000
```

```
50%       24.990000
75%       39.515000
max      279.990000
Name: Price, dtype: float64
```

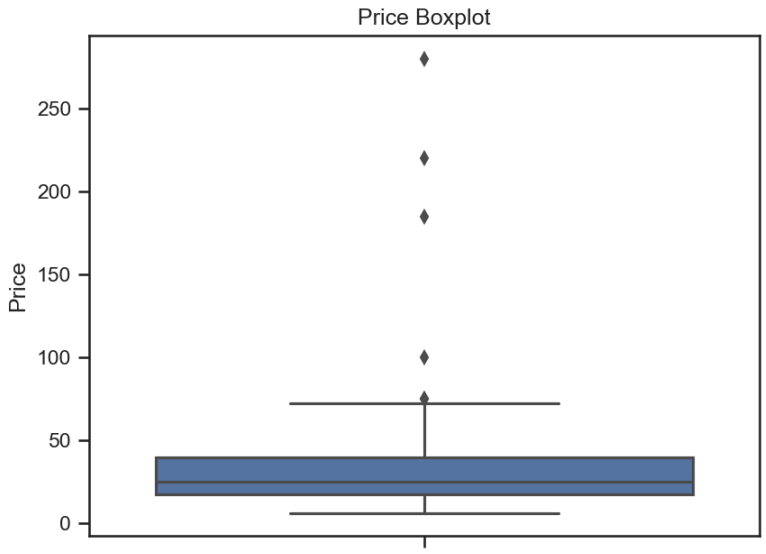
```
onlyTrue.describe()
```

	#	Price	FBA fee	Sales	Revenue	BSR	Rating	Review Count	Review Velocity	We
count	198.000000	198.000000	198.000000	198.000000	198.000000	198.000000	198.000000	198.000000	196.000000	198.00
mean	4.106061	31.710051	7.185455	920.528615	25797.175516	55078.369394	4.500000	588.722222	69.673469	0.28
std	4.045912	29.448224	4.507198	1829.801796	57897.819273	73885.422994	0.780472	1076.107291	158.135273	0.25
min	1.000000	5.990000	0.000000	0.000000	0.000000	1230.000000	0.000000	0.000000	-105.000000	0.01
25%	1.000000	16.990000	5.000000	173.750000	3849.820000	17387.000000	4.500000	82.250000	8.000000	0.10
50%	3.000000	24.990000	6.770000	355.000000	9411.895000	37018.500000	4.500000	240.500000	23.000000	0.25
75%	5.000000	39.515000	9.000000	923.500000	28070.955000	64918.750000	5.000000	672.000000	68.750000	0.40
max	21.000000	279.990000	45.000000	15974.000000	608449.660000	814721.000000	5.000000	9061.000000	1272.000000	1.55

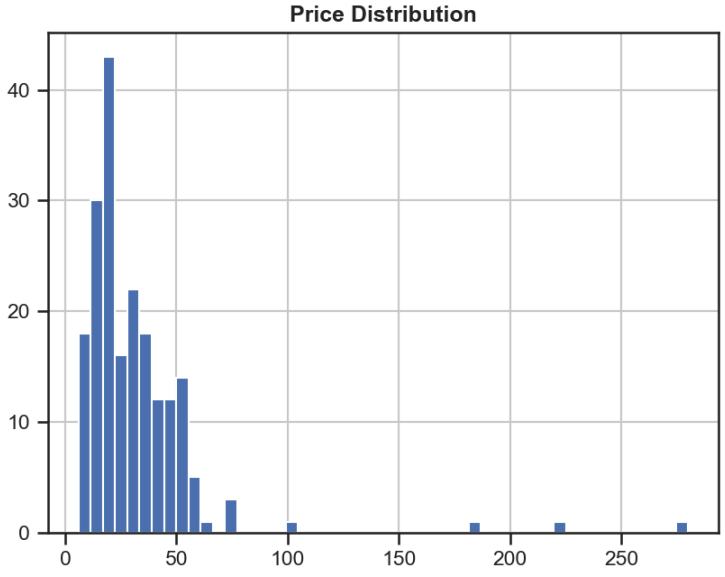
```
Brands=onlyTrue.groupby('Title')['Price'].mean().sort_values(ascending=False)
Brands.values
```

```
array([279.99      , 219.81      , 184.9       ,  99.9       ,
        75.        ,  74.95      ,  72.        ,  62.5       ,
        59.99      ,  59.9       ,  58.99      ,  57.95      ,
        57.2       ,  54.8       ,  53.83      ,  52.75      ,
        52.24333333,  52.        ,  50.        ,  49.99      ,
        49.99      ,  49.99      ,  49.99      ,  49.95      ,
        49.9       ,  48.99      ,  48.39      ,  47.99      ,
        47.98      ,  47.76      ,  47.27      ,  47.        ,
        46.99      ,  45.99      ,  44.99      ,  44.99      ,
        44.95      ,  42.99      ,  42.99      ,  42.2       ,
        42.        ,  42.        ,  41.49      ,  39.99      ,
        39.99      ,  39.95      ,  39.69      ,  38.99      ,
        38.44      ,  38.09      ,  37.99      ,  37.99      ,
        37.9       ,  36.85      ,  35.99      ,  35.95      ,
        35.54      ,  34.99      ,  34.97      ,  34.94      ,
        34.9       ,  34.88      ,  34.5       ,  34.05      ,
        33.95      ,  33.88      ,  32.99      ,  32.75      ,
        32.75      ,  31.95      ,  31.9       ,  31.85      ,
        31.5       ,  30.99      ,  30.99      ,  30.99      ,
        30.3       ,  30.        ,  29.99      ,  29.99      ,
        29.99      ,  28.99      ,  28.8       ,  28.58      ,
        28.5       ,  28.04      ,  27.99      ,  27.99      ,
        27.        ,  26.99      ,  26.99      ,  26.85      ,
        25.99      ,  24.99      ,  24.99      ,  24.99      ,
        24.99      ,  24.95      ,  24.        ,  24.        ,
        22.99      ,  22.99      ,  22.99      ,  22.83      ,
        22.25      ,  22.22      ,  21.99      ,  21.99      ,
        21.98      ,  21.94      ,  21.25      ,  20.99      ,
        20.99      ,  20.95      ,  20.        ,  19.99      ,
        19.99      ,  19.99      ,  19.99      ,  19.99      ,
        19.99      ,  19.99      ,  19.99      ,  19.99      ,
        19.98      ,  19.95      ,  19.9       ,  19.9       ,
        19.8       ,  19.        ,  18.99      ,  18.59      ,
        18.5       ,  18.        ,  17.99      ,  17.99      ,
        17.95      ,  17.95      ,  17.95      ,  17.7       ,
        17.66      ,  17.49      ,  17.25      ,  16.99      ,
        16.99      ,  16.99      ,  16.99      ,  16.5       ,
        15.99      ,  15.99      ,  15.99      ,  15.53      ,
        14.99      ,  14.99      ,  14.99      ,  14.99      ,
        14.99      ,  14.99      ,  14.99      ,  14.9       ,
        14.9       ,  14.75      ,  14.5       ,  14.39      ,
        13.995     ,  13.95      ,  13.79      ,  13.76      ,
        13.25      ,  13.25      ,  12.99      ,  12.98      ,
        12.97      ,  12.5       ,  11.99      ,  11.5       ,
        11.4       ,  10.99      ,  10.99      ,  10.15      ,
         9.99      ,   9.99      ,   9.99      ,   9.85      ,
         9.71      ,   9.43      ,   8.75      ,   7.99      ,
         6.99      ,   6.99      ,   6.99      ,   6.99      ,
         6.95      ,   5.99      ])
```

```
plt.figure()
sns.boxplot(y=onlyTrue['Price'])
#sns.boxplot(y=1199, color="white")
plt.title("Price Boxplot")
plt.show()
```




```
plt.figure()
onlyTrue['Price'].hist(bins=50)
plt.title('Price Distribution', weight='bold')
plt.show()
```




```
onlyTrue['Sales'].mean()
```

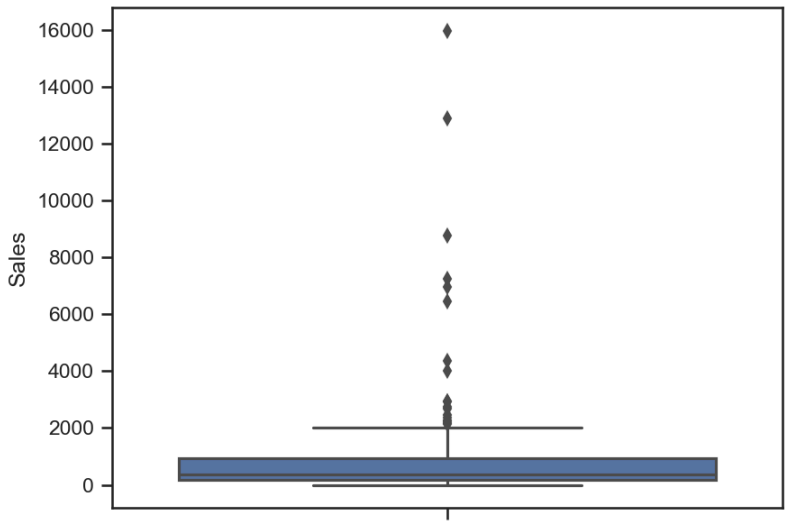
```
920.5286152987156
```

```
onlyTrue['Sales'].describe()
```

```
count      198.000000
mean       920.528615
std       1829.801796
min          0.000000
```

```
25%        173.750000
50%        355.000000
75%        923.500000
max      15974.000000
Name: Sales, dtype: float64
```

```
plt.figure()
sns.boxplot(y=onlyTrue['Sales'])
plt.show()
```



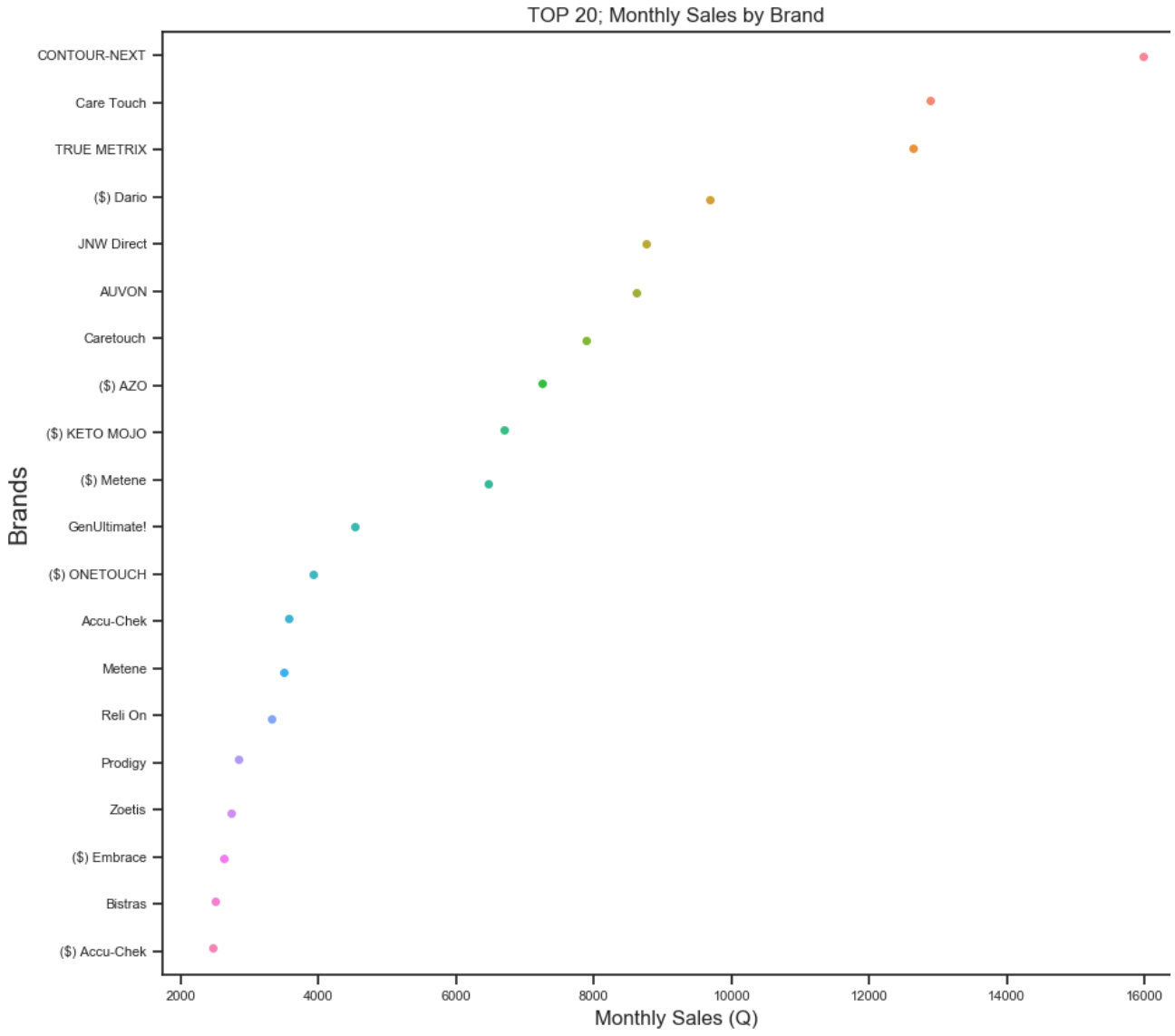

```
salesbrands=onlyTrue.groupby('Brand')['Sales'].sum().sort_values(ascending=False)
salesbrands_df=pd.DataFrame({'ASIN':salesbrands.index, 'Monthly Sales (Q)':salesbrands.values})
salesbrands_df.head(20)
```

	ASIN	Monthly Sales (Q)
0	CONTOUR-NEXT	15974.000000
1	Care Touch	12892.000000
2	TRUE METRIX	12633.000000
3	($) Dario	9685.000000
4	JNW Direct	8764.000000
5	AUVON	8617.000000
6	Caretouch	7884.000000
7	($) AZO	7248.000000
8	($) KETO MOJO	6689.000000
9	($) Metene	6458.000000
10	GenUltimate!	4532.000000
11	($) ONETOUCH	3929.000000
12	Accu-Chek	3570.000000
13	Metene	3490.000000
14	Reli On	3323.703518
15	Prodigy	2837.000000
16	Zoetis	2728.000000
17	($) Embrace	2623.703518
18	Bistras	2498.000000
19	($) Accu-Chek	2466.000000

```
salesbrands_df.describe()
```

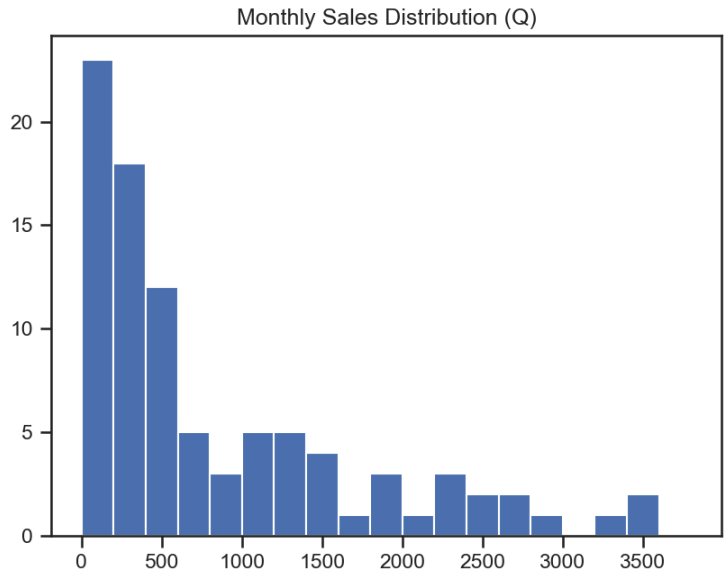
	Monthly Sales (Q)
count	103.000000
mean	1769.559862
std	2934.920751
min	0.000000
25%	220.500000
50%	580.000000
75%	1874.925879
max	15974.000000

```
plt.figure(figsize=(10,10))
a=sns.stripplot(y="ASIN", x="Monthly Sales (Q)", data=salesbrands_df.head(20))
a.set_ylabel("Brands",fontsize=15)
a.tick_params(labelsize=8)
plt.title("TOP 20; Monthly Sales by Brand")
```

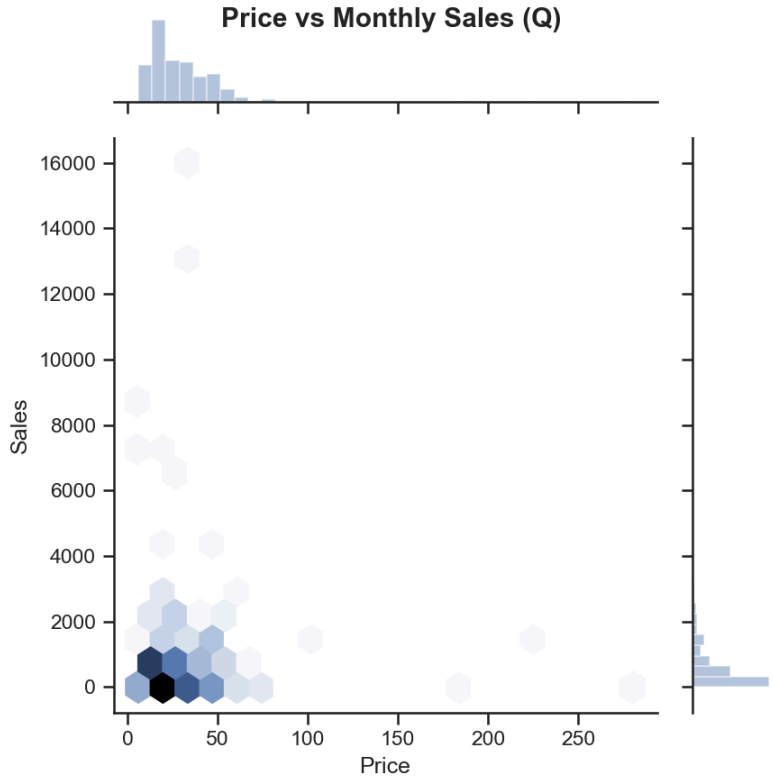



```
Text(0.5, 1.0, 'TOP 20; Monthly Sales by Brand')
```

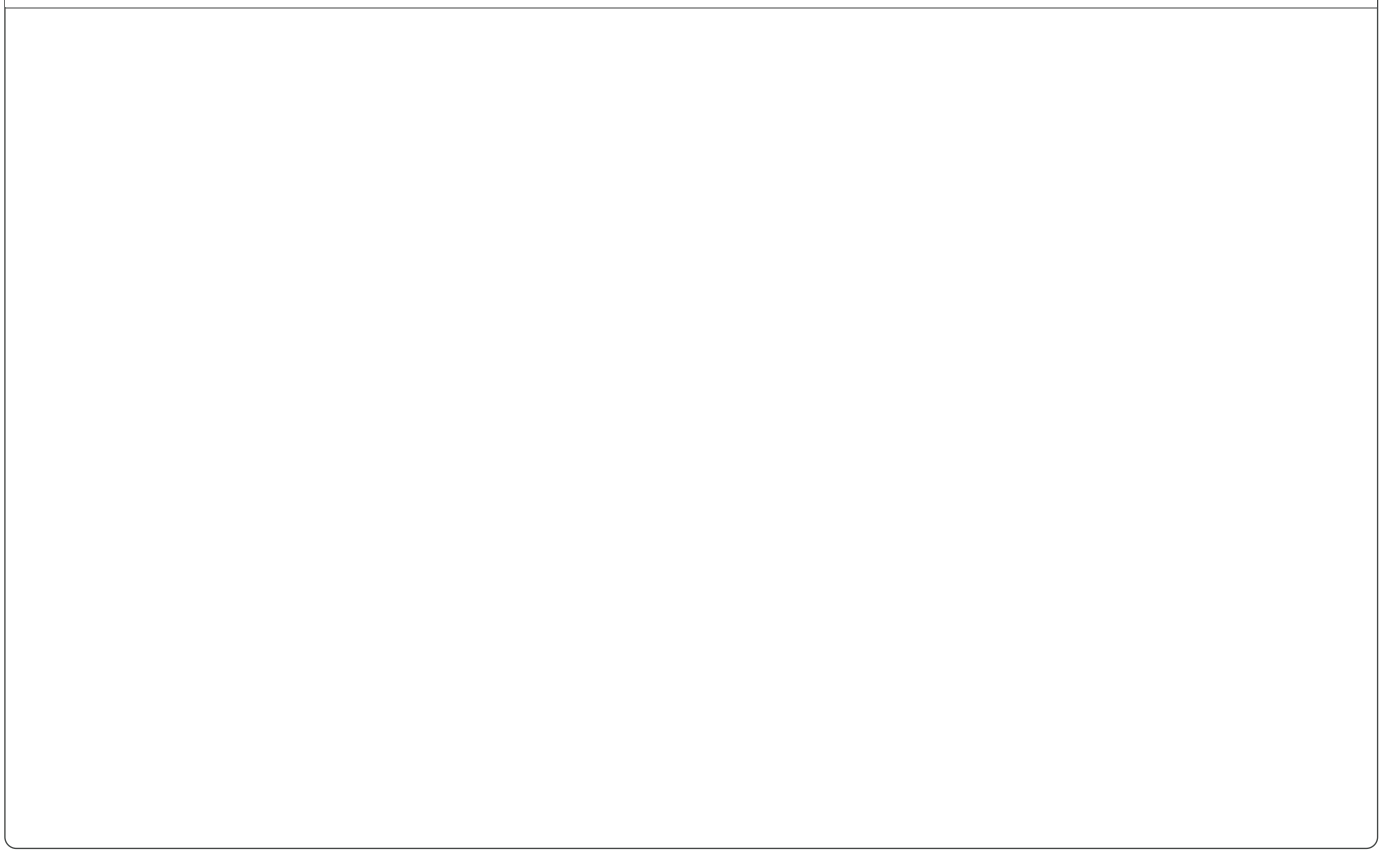
```
import math
observations = salesbrands_df['Monthly Sales (Q)'].count()
r=max(salesbrands_df['Monthly Sales (Q)']) - min(salesbrands_df['Monthly Sales (Q)'])
intervals = math.sqrt(observations)
width= r/intervals
binss=list(range(0, 4000, 200))
plt.figure()
plt.hist(salesbrands_df['Monthly Sales (Q)'], bins=binss)
plt.title("Monthly Sales Distribution (Q)")
plt.show()
```




```
sns.set()
sns.set_style("ticks")
chart = sns.jointplot(x='Price', y='Sales', data=onlyTrue, kind='hex', joint_kws=dict(gridsize=20))
#chart.grid(False)
#plt.title('Rating vs Price', weight='bold')
chart.fig.suptitle("Price vs Monthly Sales (Q)", weight='bold')
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-small'
);
plt.tight_layout()
```



```
sns.set()
sns.set_style("ticks")
chart = sns.jointplot(x='Weight', y='Sales', data=onlyTrue, kind='hex', joint_kws=dict(gridsize=20))
#chart.grid(False)
#plt.title('Rating vs Price', weight='bold')
chart.fig.suptitle("Weight vs Monthly Sales (Q)", weight='bold')
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-small'
);
plt.tight_layout()
```



```
sns.lmplot( x="Price", y="Sales", data=onlyTrue, fit_reg=False)
```

```
<seaborn.axisgrid.FacetGrid at 0x1a80c862908>
```

```
revbrands=onlyTrue.groupby('Brand')['Revenue'].sum().sort_values(ascending=False)
revbrands_df=pd.DataFrame({'Brand':revbrands.index, 'Monthly Revenue ($)':revbrands.values})
revbrands_df.head(20)
```

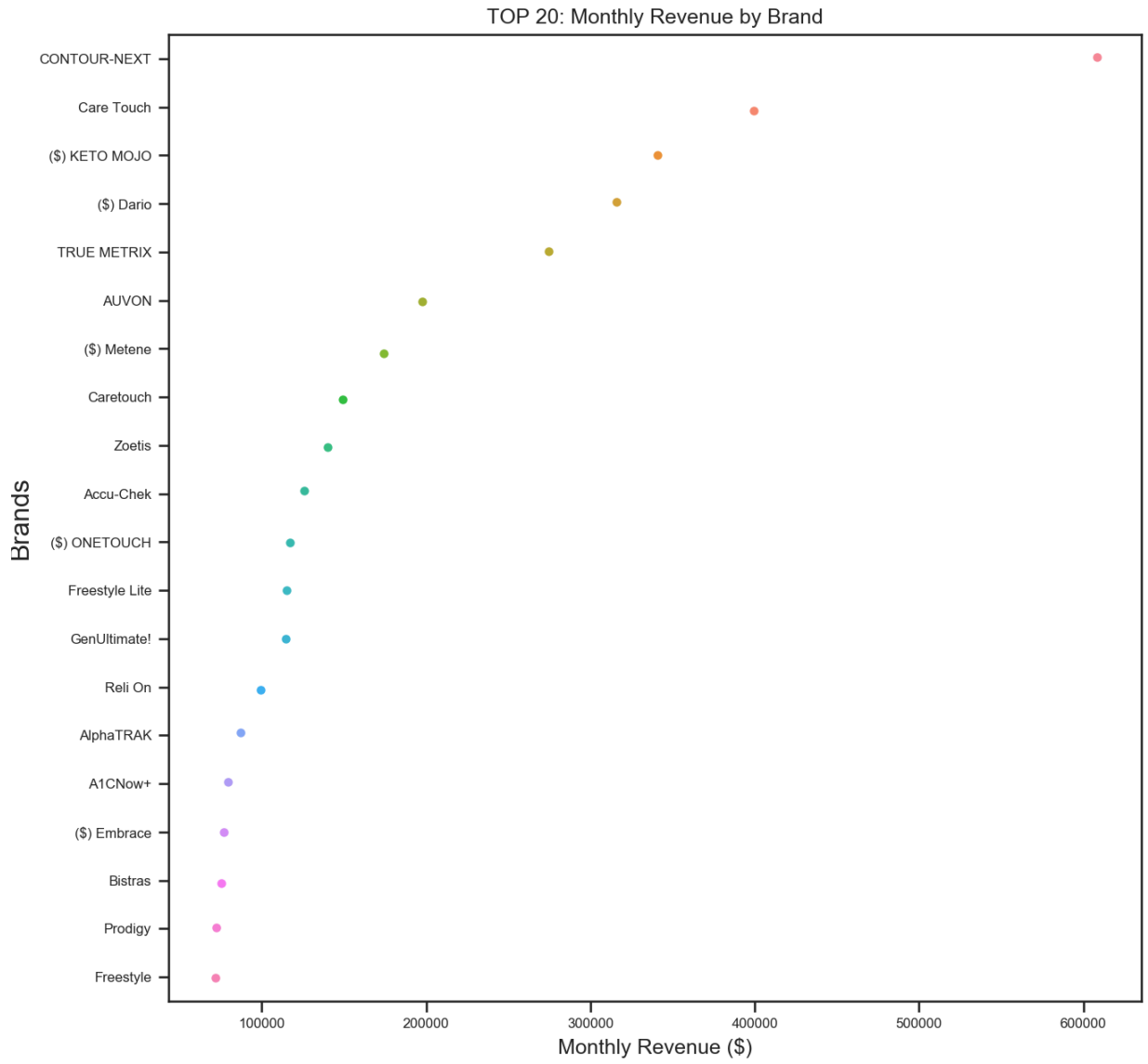
	Brand	Monthly Revenue ($)
0	CONTOUR-NEXT	608449.660000
1	Care Touch	399523.080000
2	($) KETO MOJO	341048.110000
3	($) Dario	315893.700000
4	TRUE METRIX	274566.000000
5	AUVON	197478.830000
6	($) Metene	174301.420000
7	Caretouch	149150.160000
8	Zoetis	140081.040000
9	Accu-Chek	125880.730000
10	($) ONETOUCH	116900.710000
11	Freestyle Lite	115126.500000
12	GenUltimate!	114795.400000
13	Reli On	99489.371583
14	AlphaTRAK	86918.280000
15	A1CNow+	79506.910000
16	($) Embrace	76733.631583
17	Bistras	75575.100000
18	Prodigy	72537.450000
19	Freestyle	71834.890000

```
revbrands_df['Monthly Revenue ($)'].describe()
```

```
count       103.000000
mean      49590.686914
std       90678.856827
min           0.000000
25%        4861.875000
```

```
50%       14440.090000
75%       60406.230000
max      608449.660000
Name: Monthly Revenue ($), dtype: float64
```

```
plt.figure(figsize=(10,10))
a=sns.stripplot(y="ASIN", x="Monthly Revenue ($)", data=revbrands_df.head(20))
a.set_ylabel("Brands",fontsize=15)
a.tick_params(labelsize=8)
plt.title("TOP 20: Monthly Revenue by Brand")
```




```
Text(0.5, 1.0, 'TOP 20: Monthly Revenue by Brand')
```

```
reviews=onlyTrue.groupby('Brand')['Review Count'].mean().sort_values(ascending=False)
reviews_df=pd.DataFrame({'Brand':reviews.index, 'Review Count':reviews.values})
reviews_df.head(20)
```

	Brand	Review Count
0	Care Touch	9061.000000
1	JNW Direct	7699.000000
2	CONTOUR-NEXT	4562.000000
3	($) Metene	4434.000000
4	($) AZO	3889.000000
5	True Point Generic Test Strips	2177.000000
6	HealthyWiser	1834.000000
7	Caretouch	1813.000000
8	Generic	1694.000000
9	($) KetoSens	1683.000000
10	Active Forward	1548.000000
11	Freestyle Lite	1385.000000
12	Zoetis	1362.000000
13	A1CNow+	1264.000000
14	On Call	1103.500000
15	AUVON	992.714286
16	Freestyle	979.000000
17	($) ijCare	972.000000
18	AlphaTRAK	915.000000
19	($) Dario	872.666667

```
reviews_df['Review Count'].describe()
```

```
count     103.000000
mean      674.875659
std      1361.807819
min         0.000000
25%        83.333333
50%       251.000000
75%       677.900000
max      9061.000000
Name: Review Count, dtype: float64
```

```
reviews_df['Review Count'].describe()
```

```
count     103.000000
mean      674.875659
std      1361.807819
min         0.000000
25%        83.333333
50%       251.000000
75%       677.900000
max      9061.000000
Name: Review Count, dtype: float64
```

```
rating=onlyTrue.groupby('Brand')['Rating'].mean().sort_values(ascending=False)
rating_df=pd.DataFrame({'Brand':rating.index, 'Rating':rating.values})
rating_df.head(20)
```

	Brand	Rating
0	Evencare	5.000
1	Contour Next	5.000
2	Omnis Health	5.000
3	On Call Express	5.000
4	Infinity	5.000
5	Home Diagnostics	5.000
6	Precision Brand	5.000
7	Generic	5.000
8	Quintet	5.000
9	GE	5.000
10	MHC	5.000
11	Caretouch	5.000
12	Swift Lite	5.000
13	Bionime GE100	5.000
14	AlphaTRAK	5.000
15	AgaMatrix	5.000
16	True Track	5.000
17	Abbott	5.000
18	($) Ketone Testing Kit by Generical X	5.000
19	Reli On	4.875

```
rating_df['Rating'].describe()
```

```
count    108.000000
mean       4.409236
std        0.912799
min        0.000000
25%        4.500000
50%        4.500000
75%        4.754808
max        5.000000
Name: Rating, dtype: float64
```

```
Start coding or generate with AI.
```

```
data[['Brand','BSR']].sort_values(by='BSR', ascending=True).head(10)
```

	Brand	BSR
239	CONTOUR-NEXT	1230.0
235	CONTOUR-NEXT	1230.0
238	CONTOUR-NEXT	1230.0
236	CONTOUR-NEXT	1230.0
237	CONTOUR-NEXT	1230.0
240	CONTOUR-NEXT	1230.0
234	CONTOUR-NEXT	1230.0
13	Care Touch	1641.0
11	Care Touch	1641.0
10	Care Touch	1641.0

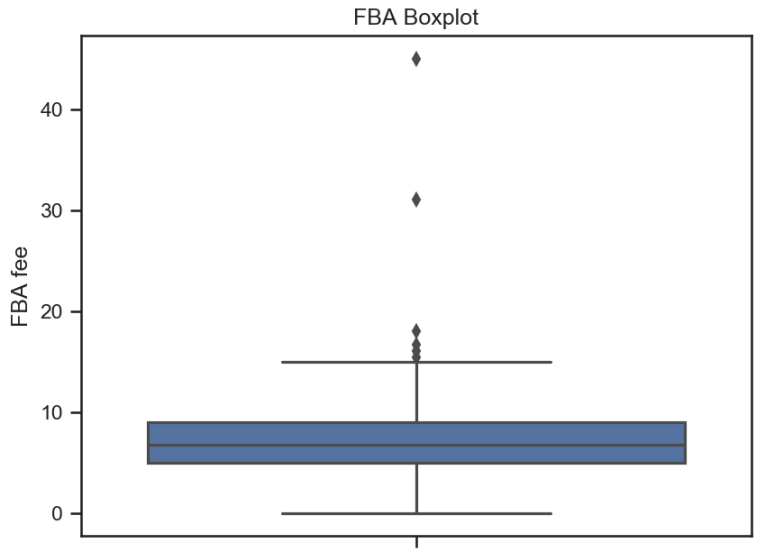
```
data['FBA fee'].mean()
```

```
7.41371670702179
```

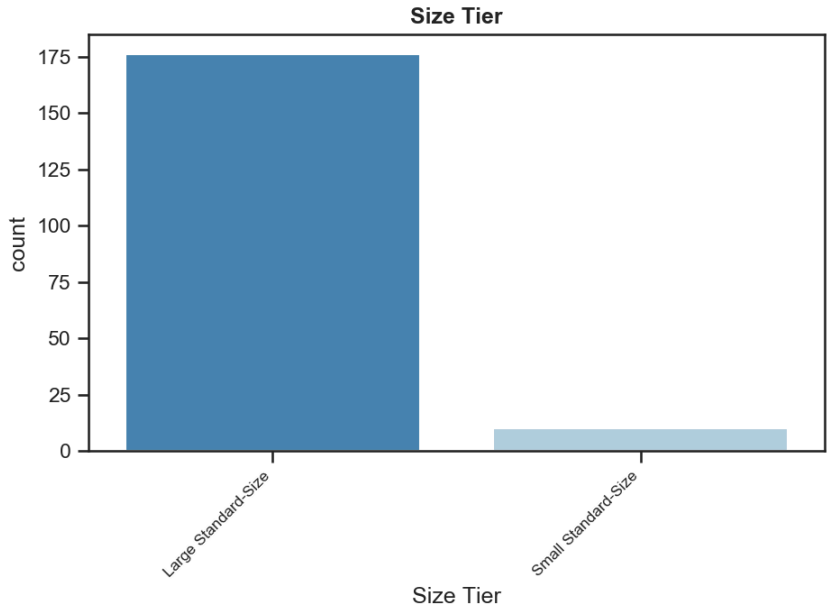
```
data['FBA fee'].describe()
```

```
count    826.000000
mean       7.413717
std        3.735473
min        0.000000
25%        5.000000
50%        7.000000
75%        9.000000
max       45.000000
Name: FBA fee, dtype: float64
```

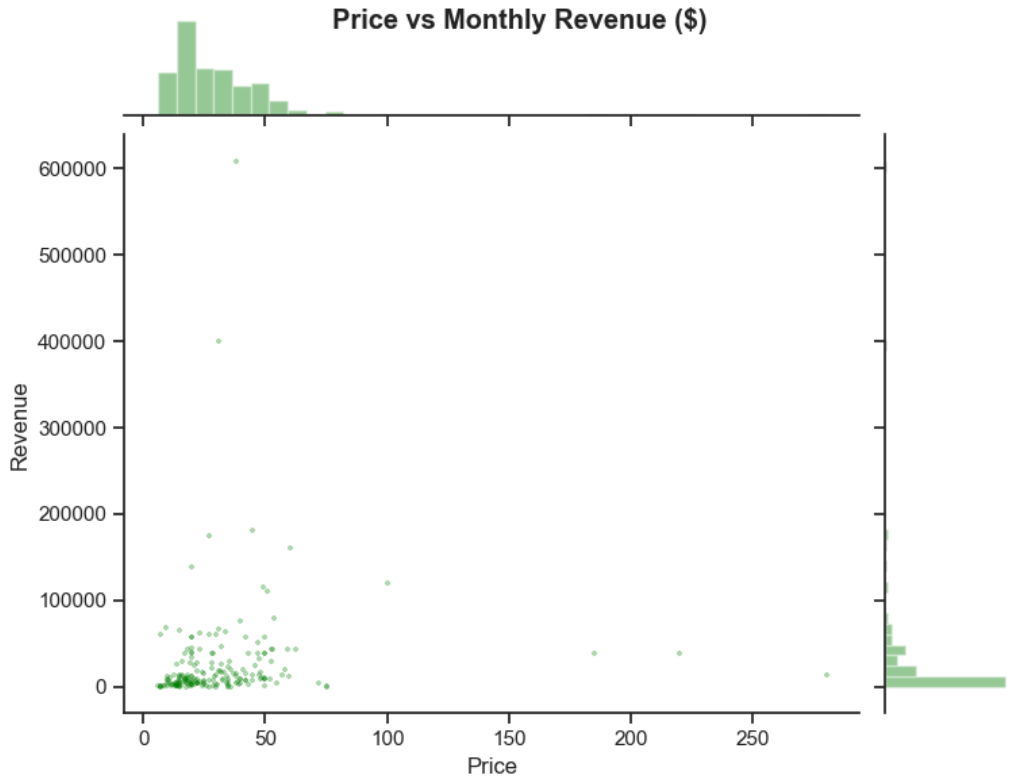
```
plt.figure()
sns.boxplot(y=onlyTrue['FBA fee'])
plt.title("FBA Boxplot")
plt.show()
```




```
plt.figure()
sns.set()
sns.set_style("ticks")
chart = sns.countplot(x=onlyTrue['Size Tier'], data=onlyTrue, palette="Blues_r")
chart.grid(False)
plt.title('Size Tier', weight='bold')
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-small'
)
plt.tight_layout()
```

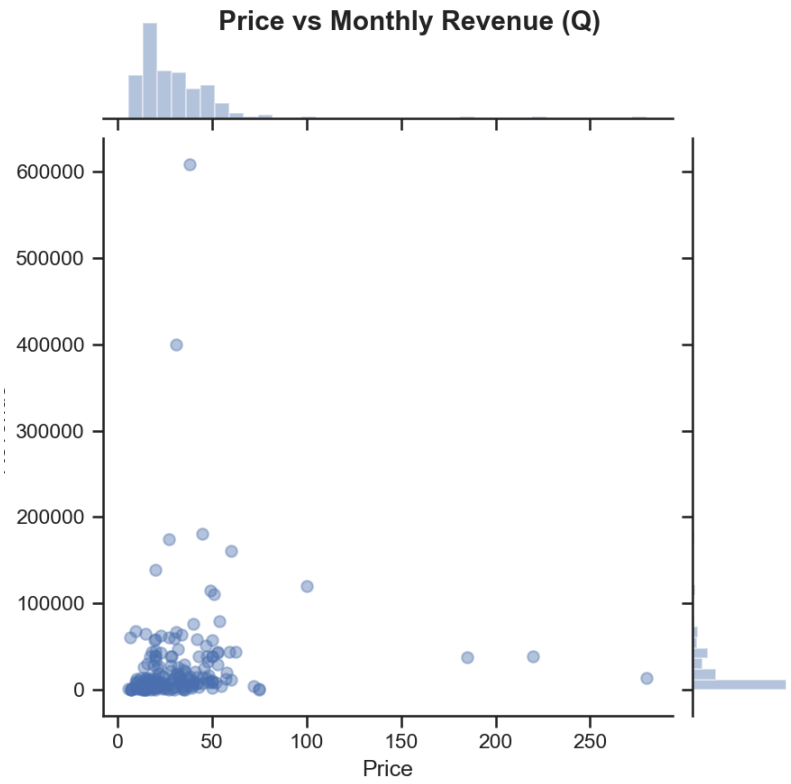



```
c=sns.jointplot(onlyTrue['Price'], onlyTrue['Revenue'], alpha=0.3, marker='+', s=5, color='green');
c.fig.suptitle("Price vs Monthly Revenue ($)", weight='bold')

plt.show()
```




```
b=sns.jointplot(onlyTrue['Price'], onlyTrue['Sales'], alpha=0.3, marker='+', s=5, color='green');
b.fig.suptitle("Price vs Monthly Sales (Q)", weight='bold')

plt.show()
```

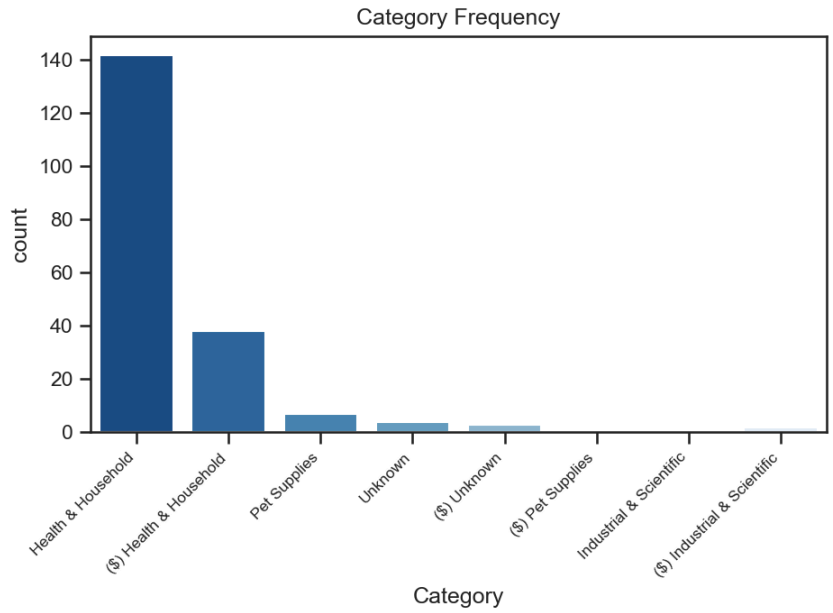

```
x=sns.jointplot(onlyTrue['Price'], onlyTrue['Revenue'], alpha=0.4);
x.fig.suptitle("Price vs Monthly Revenue (Q)", weight='bold')
plt.show()
```




```
a=onlyTrue.groupby('Brand')['Revenue'].sum().sort_values(ascending=False)
a_df=pd.DataFrame({'Brand':a.index, 'Revenue':a.values})
a_df.head(10)
```

	Brand	Revenue
0	CONTOUR-NEXT	608449.66
1	Care Touch	399523.08
2	($) KETO MOJO	341048.11
3	($) Dario	315893.70
4	TRUE METRIX	274566.00
5	AUVON	197478.83
6	($) Metene	174301.42
7	Caretouch	149150.16
8	Zoetis	140081.04
9	Accu-Chek	125880.73

```
plt.figure()
sns.set()
sns.set_style("ticks")
chart = sns.countplot(x=onlyTrue['Category'], data=data, palette="Blues_r")
chart.grid(False)
plt.title('Category Frequency')
plt.xticks(
    rotation=45,
    horizontalalignment='right',
    fontweight='light',
    fontsize='x-small'
)
plt.tight_layout()
```

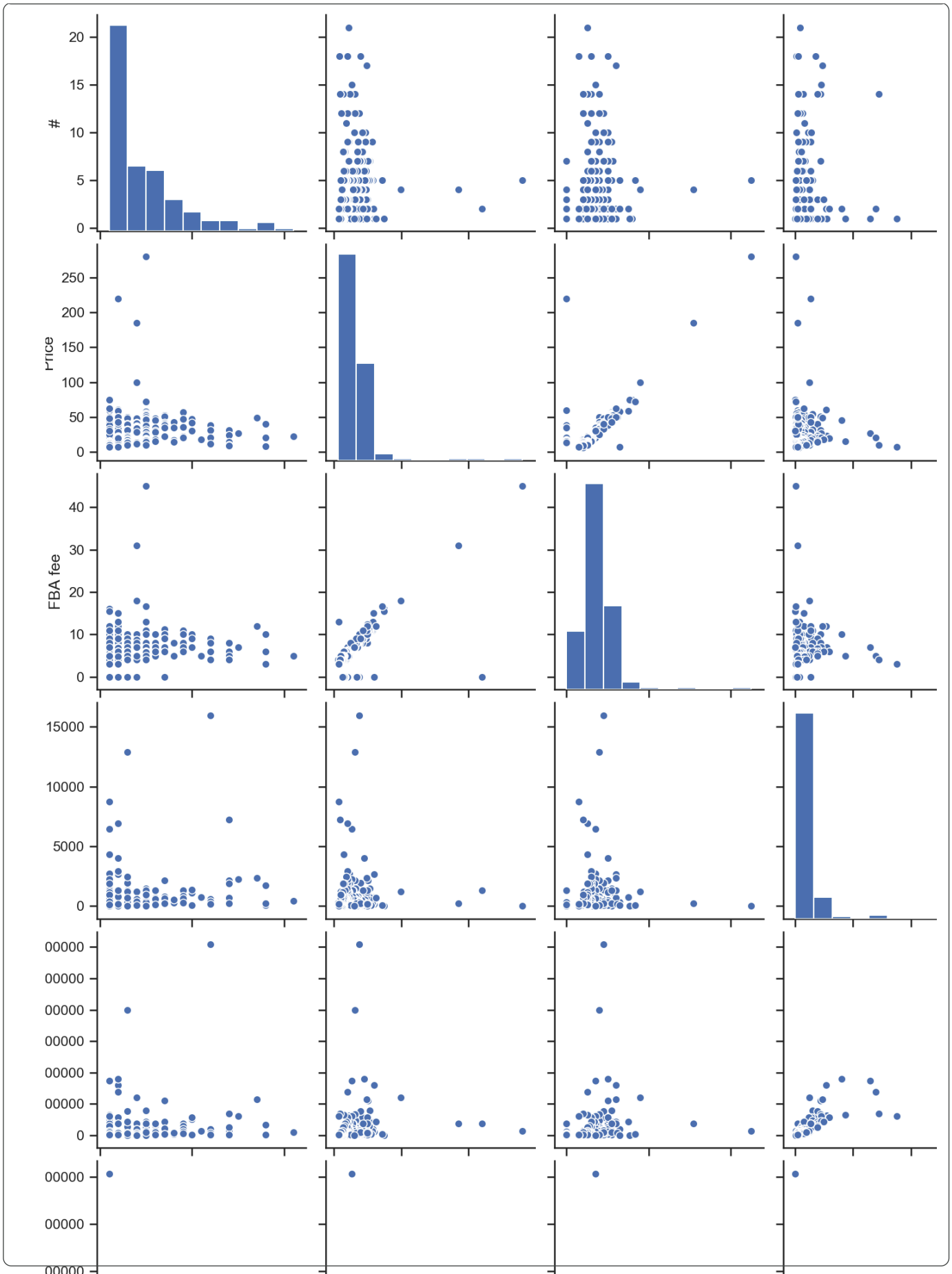



```
dd= onlyTrue['Category'].value_counts()
pd.DataFrame(dd, columns=['Frequency'])
```

	Frequency

```
n=sns.relplot(x='Price', y='Sales', data=onlyTrue, hue='Clasification', size='Revenue',alpha=0.4);
n.fig.suptitle("Sales vs Price vs Revenue", weight='bold')
plt.show()
```


```
sns.pairplot(onlyTrue)
```



KEYWORD ANALYSIS

```
Start coding or generate with AI.
```

```
import re
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import string
import nltk

import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to C:\Users\Jorge
[nltk_data]     Campuzano\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
train2 = data['Title'].str.replace("[^a-zA-Z#]", " ")
```

```
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to C:\Users\Jorge
[nltk_data]     Campuzano\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
stop_words = set(stopwords.words('english'))
```

```
lista=[]
for line in train2:
    filtered_sentence=[]
    try:
        words = word_tokenize(line)
        for w in words:
            w=w.lower()
            if w not in stop_words:
                filtered_sentence.append(w)
        lista.append(filtered_sentence)


    except:
        continue
lista2=pd.Series(lista)
```

```
lf=[]
for item in lista2:

    texto=[]
    for word in item:
        word=word.lower()
        texto.append(word)
    lf.append(texto)
```

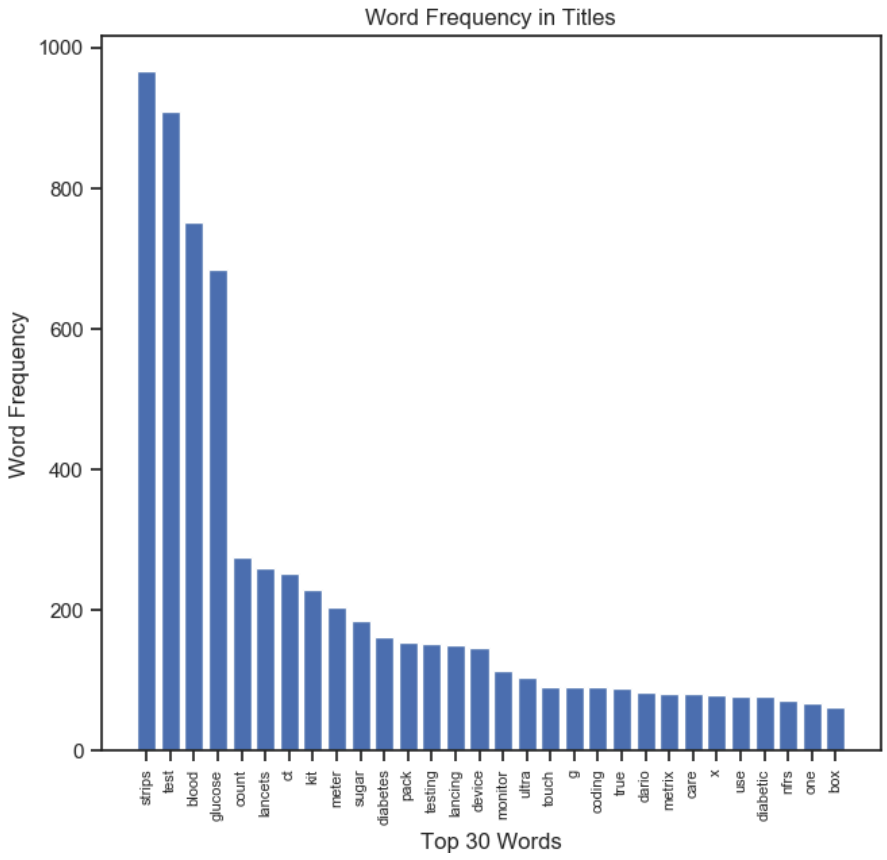
```
l3=pd.Series(lf)
```

```
from nltk.stem.porter import *
stemmer = PorterStemmer()

l4 = l3.apply(lambda x: [stemmer.stem(i) for i in x]) # stemming
```

```
import operator
counts = dict()
for line in l3:
```

```
    for word in line:
        word=word.lower()
        word=word.strip()
        if word in counts:
            counts[word] += 1
        else:
            counts[word] = 1

#print(counts)
countSort=sorted(counts.items(), key=operator.itemgetter(1),reverse=True)
```

```
%matplotlib notebook
plt.figure(figsize=(6,6))
from math import log
testList2 = [(elem1, elem2) for elem1, elem2 in countSort[:30]]
zip(*testList2)
plt.bar(*zip(*testList2))
plt.title("Word Frequency in Titles")
plt.ylabel('Word Frequency')
plt.xlabel('Top 30 Words')
plt.xticks(rotation=90, fontsize=8)

plt.show()
```




```
Start coding or generate with AI.
```

```
salesbrands2=onlyTrue.groupby('Brand')['Price'].mean().sort_values(ascending=False)
salesbrands_df2=pd.DataFrame({'Brand':salesbrands2.index, 'Average Price':salesbrands2.values})
salesbrands_df2.head(10)
```

	Brand	Average Price
0	Abbott	279.990000
1	ABBOTT DIABETES CARE	116.330000
2	McKesson	81.666667
3	($) Housey Healthcare ULC	74.950000
4	Active Forward	62.500000
5	Dario	59.900000
6	Bionime GE100	54.800000
7	A1CNow+	53.830000
8	AlphaTRAK	52.870000
9	($) KETO MOJO	52.490000

```
plt.figure(figsize=(10,10))
a=sns.stripplot(y="Brand", x="Average Price", data=salesbrands_df2.head(20))
a.set_ylabel("Brands",fontsize=15)
a.tick_params(labelsize=8)
plt.title("Average Product Price by Brands")
```