Carrie Beauzile-Milinazzo

Plotting in Python

The basic plot routine is called MatPlotLib. It is a "port" or copy of the plotting routines from MatLab, so if you know matlab, then you already know how MatPlotLib works. It's basic, but it works fine

Seaborn is much nice plotting package, it has some really nice visualizations available.

Sources

```
https://matplotlib.org/
```

```
https://seaborn.pydata.org/
```

There is a grammar of graphics based plotting routine in python, called plotnine, that should be more or less equivalent to ggplot in R

```
https://realpython.com/ggplot-python/
```

```
https://plotnine.org/
```

There are galleries available that show numerous cool plotting methods in each of these

Working Graphics vs Presentation Graphics

It is important to understand that there are two very different types of graphics that you will need to work with in Data Science

1.) Working Graphics- These are the graphics and visualizations that we use during the course of an analysis, they are a huge part of how we gain our understanding of the system we are working with. Working graphics need to be:

```
a.) accurate
b.) not misleading
c.) easy to adjust and work with
d.) visually simple, no fancy colors, fonts, etc, etc
e.) you can use specialized visual representations of complex
mathematics or statistics without concern
```

These are visualizations that we create as part of the workflow, and we create a lot of them

The "visual quality" or "visual appeal" of these graphics isn't particularly important. I've looked at hundreds or thousands of regressions, I don't need anything beyond the basic graphics. A simple, plain graph using the default setting works fine, I learn what I need to from it and move on.

So Fast, Accurate, easily adjusted, not high visual appeal or quality.

2.) Presentation graphics- The truth is that a lot of subject matter experts (domain experts, or line of business or business executives) will make judgements on the quality of your work based on the quality of the graphics.
This is unfair, and probably unreasonable, but there you have it. They often don't understand the mathematics of what you are doing, so they make judgements based on other aspects of the work, notably the visual quality of the presentation

Presentation graphics need to be

```
a.) accurate
b.) not misleading
c.) highly visually appealing- use the organizational color scheme,
the desired font, the symbols and their sizes
     all need to be as close to perfect as possible.
d.)  All labels need to be in place and as close to perfect as
possible
e.)  Really complex mathematical representations may be difficult
to explain,   avoid them in presentations
      if you don't really need them.
f.)  Painstakingly prepared with high attention to all details.
```

These two types of presentation are almost diametrically opposed!

I find creating really high quality presentation graphics to be dull chore, that easily frustrates me. Once I have seem the contents of a piecing of "working graphics", I know what the graph is telling me and I want to move on to the next question. It can take hours to get the graphics "perfect".

Fortunately, there are people who just love making high quality graphics. I am thrilled when I can find someone like this to be on my team, because while creating this grade of graphics is difficult and boring for me, I understand the value.

So here some examples of using these packages. They will all get the job done, matplotlib is easy for working graphics, but not idea for presentation.

Seaborn and plotnine will probably due better at easily producing production grade graphics.

matplot lib

We will look at three basic graphs in matplotlib

```
-boxplot
-histogram
-biplot
```

for many examples see

https://matplotlib.org/2.0.2/gallery.html

In [1]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import os
```

In [2]:
```
# chack on the current working directory

os.getcwd()

os.chdir('C:\\Users\\cmilinazzo\\Documents\\USB Drive\\R and Python Programming\\Mo
```

In [3]:
```
#get the dataframe from the file sales.csv
# set the file path to the location you saved the file or make sure it is in your c

df = pd.read_csv("sales-3.csv")

df.head()
```

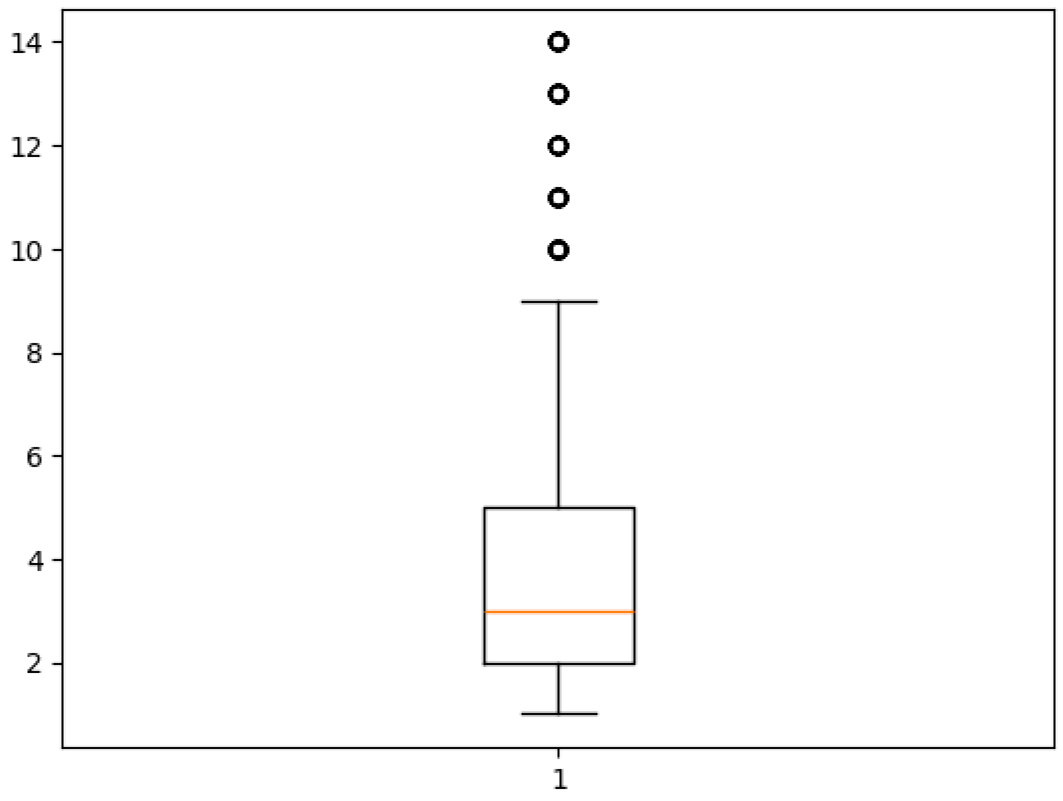
Out[3]:

	Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment	Countr
0	1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	Unite Stat
1	2	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer	Unite Stat
2	3	CA-2016-138688	6/12/2016	6/16/2016	Second Class	DV-13045	Darrin Van Huff	Corporate	Unite Stat
3	4	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	Unite Stat
4	5	US-2015-108966	10/11/2015	10/18/2015	Standard Class	SO-20335	Sean O'Donnell	Consumer	Unite Stat

5 rows × 21 columns

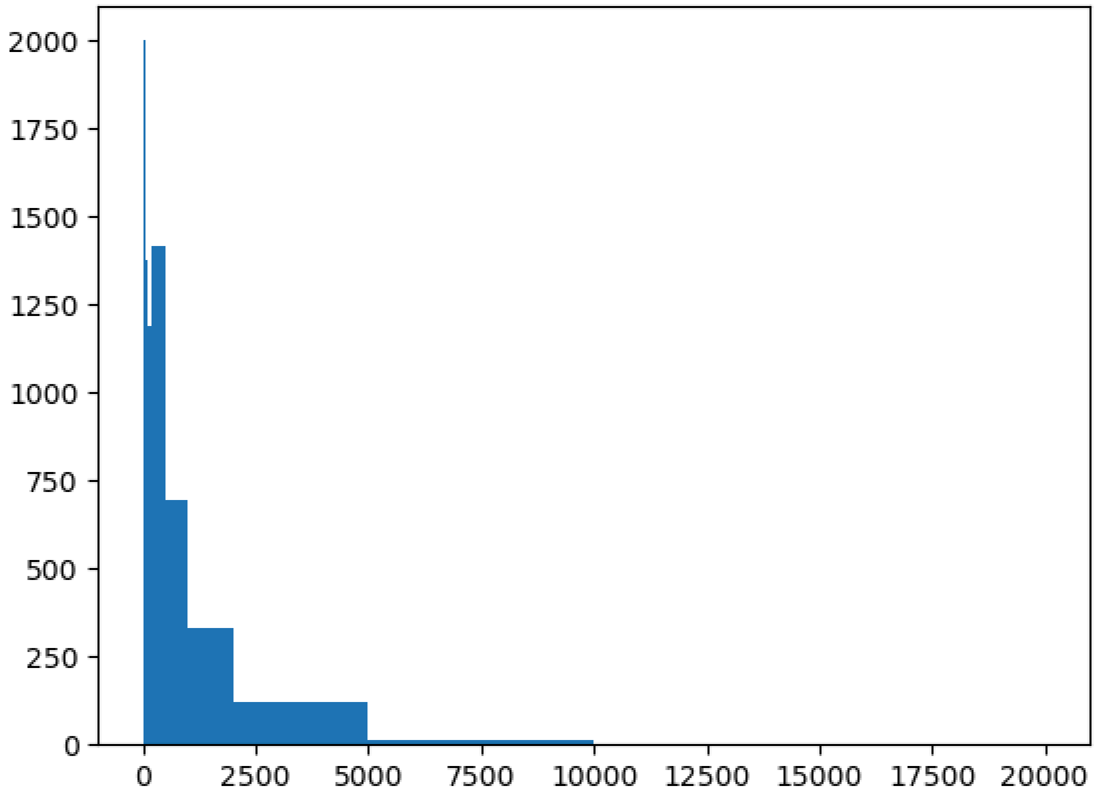
In [4]:
```
#boxplot using matplotlib



plt.boxplot(df.Quantity)
```

Out[4]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x2b117afcb30>,
  <matplotlib.lines.Line2D at 0x2b117afce90>],
 'caps': [<matplotlib.lines.Line2D at 0x2b117afd1c0>,
  <matplotlib.lines.Line2D at 0x2b117afd4f0>],
 'boxes': [<matplotlib.lines.Line2D at 0x2b116f7f140>],
 'medians': [<matplotlib.lines.Line2D at 0x2b117afd7c0>],
 'fliers': [<matplotlib.lines.Line2D at 0x2b117afdaf0>],
 'means': []}
```



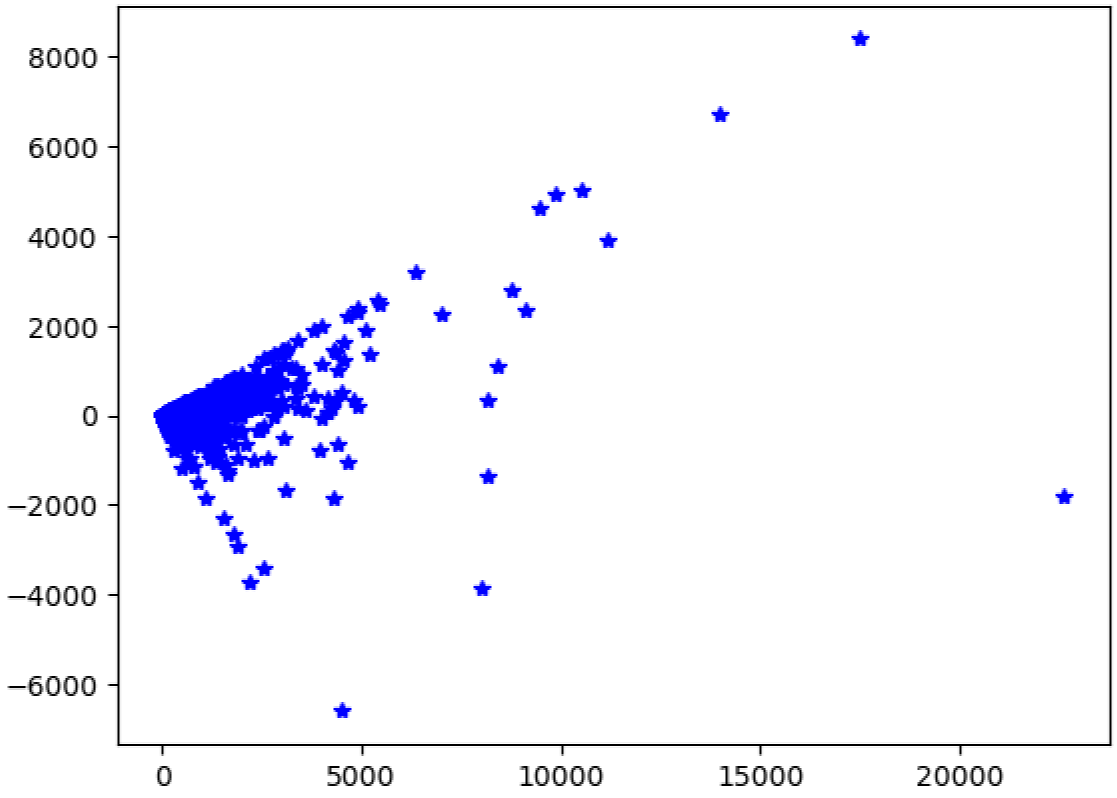
In [5]:
```
#histogram using Matplotlib

#note the values in "bins" are the boundaries of the boxes in the histogram

plt.hist(df.Sales,bins=[0,5,10,20,50,100,200,500,1000,2000,5000,10000,20000])
```

Out[5]:
```
(array([ 507.,  855., 1490., 1997., 1377., 1189., 1417.,  694.,  328.,
         121.,   14.,    4.]),
 array([0.e+00, 5.e+00, 1.e+01, 2.e+01, 5.e+01, 1.e+02, 2.e+02, 5.e+02,
        1.e+03, 2.e+03, 5.e+03, 1.e+04, 2.e+04]),
 <BarContainer object of 12 artists>)
```

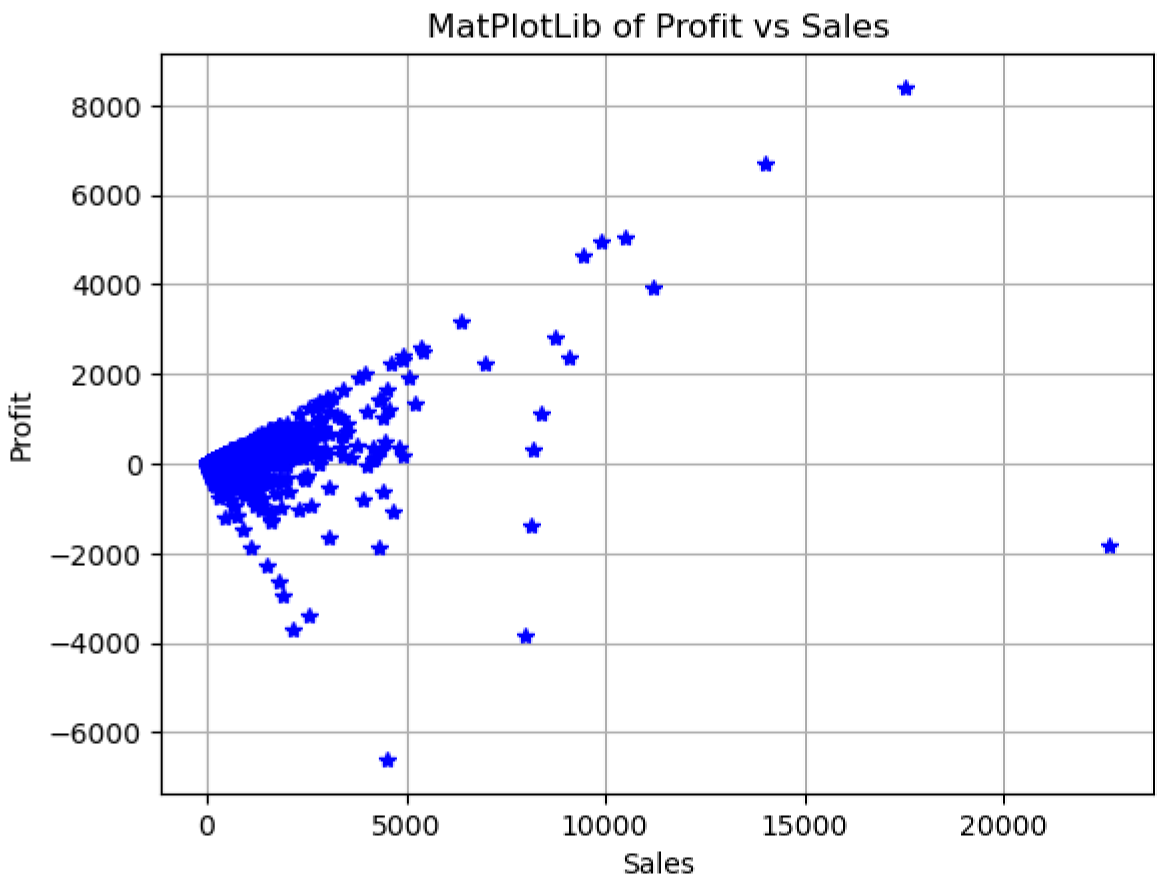


In [6]:
```
#biplot using matplotlib

#the "b*" means to plot using a blue star marker

plt.plot(df.Sales, df.Profit,"b*")
```

Out[6]:
```
[<matplotlib.lines.Line2D at 0x2b11819caa0>]
```



In [7]:
```
# we can add titles and labels
#to do this we create the figure and axes (fig, ax

fig,ax=plt.subplots()

ax.plot(df.Sales, df.Profit,"b*")
ax.set(ylabel="Profit",xlabel="Sales",title="MatPlotLib of Profit vs Sales")

#add a grid
ax.grid()

#show the plot on the axis

plt.show()
```

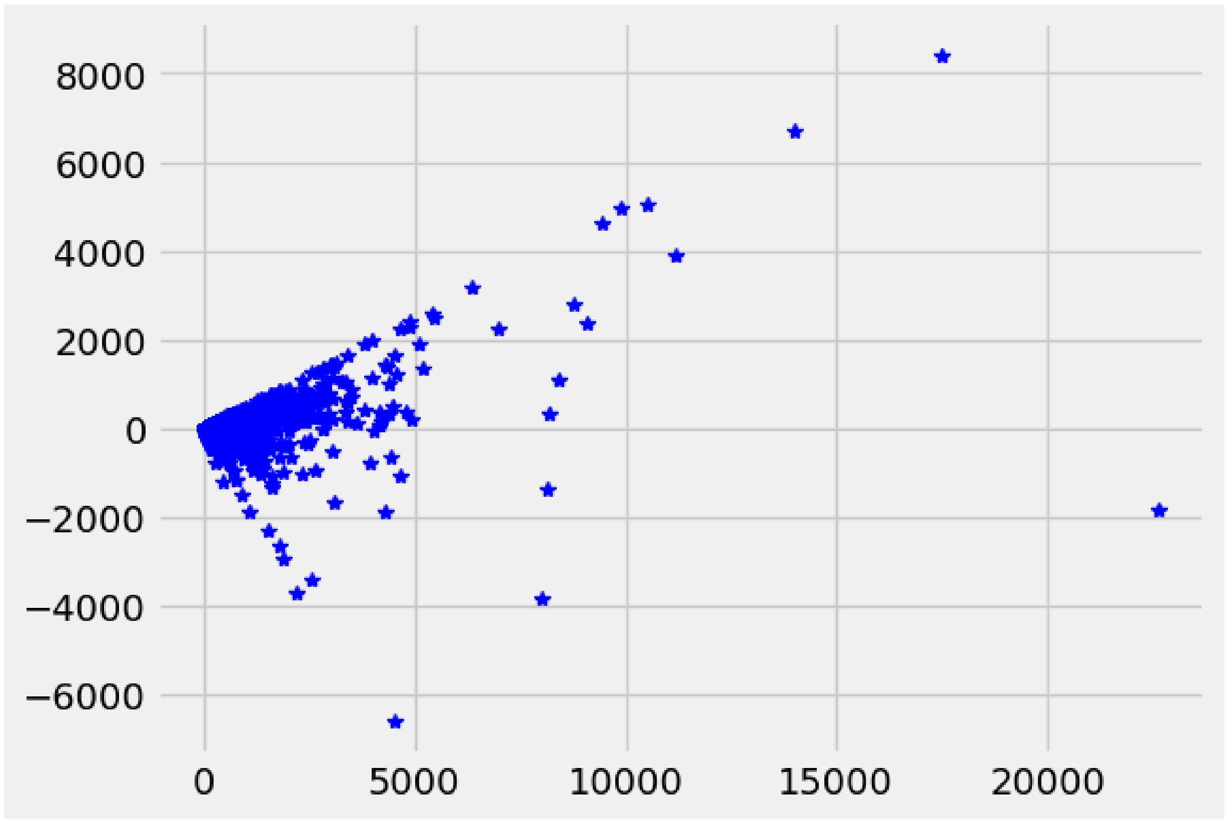


```
In [8]: print(plt.style.available)
```

```
['Solarize_Light2', '_classic_test_patch', '_mpl-gallery', '_mpl-gallery-nogrid', 'b
mh', 'classic', 'dark_background', 'fast', 'fivethirtyeight', 'ggplot', 'grayscale',
'petroff10', 'seaborn-v0_8', 'seaborn-v0_8-bright', 'seaborn-v0_8-colorblind', 'seab
orn-v0_8-dark', 'seaborn-v0_8-dark-palette', 'seaborn-v0_8-darkgrid', 'seaborn-v0_8-
deep', 'seaborn-v0_8-muted', 'seaborn-v0_8-notebook', 'seaborn-v0_8-paper', 'seaborn
-v0_8-pastel', 'seaborn-v0_8-poster', 'seaborn-v0_8-talk', 'seaborn-v0_8-ticks', 'se
aborn-v0_8-white', 'seaborn-v0_8-whitegrid', 'tableau-colorblind10']
```

```
In [9]: plt.style.use('fivethirtyeight')
        plt.plot(df.Sales, df.Profit,"b*")
```

```
Out[9]: [<matplotlib.lines.Line2D at 0x2b117711c40>]
```



In [10]:
```
#turn off style sheet

plt.rcdefaults()
```

Creating your own style

you can create your own stylesheet and completely customize how matplotlib graphics display you could do this for your organization and share it...

https://matplotlib.org/stable/users/explain/customizing.html#customizing

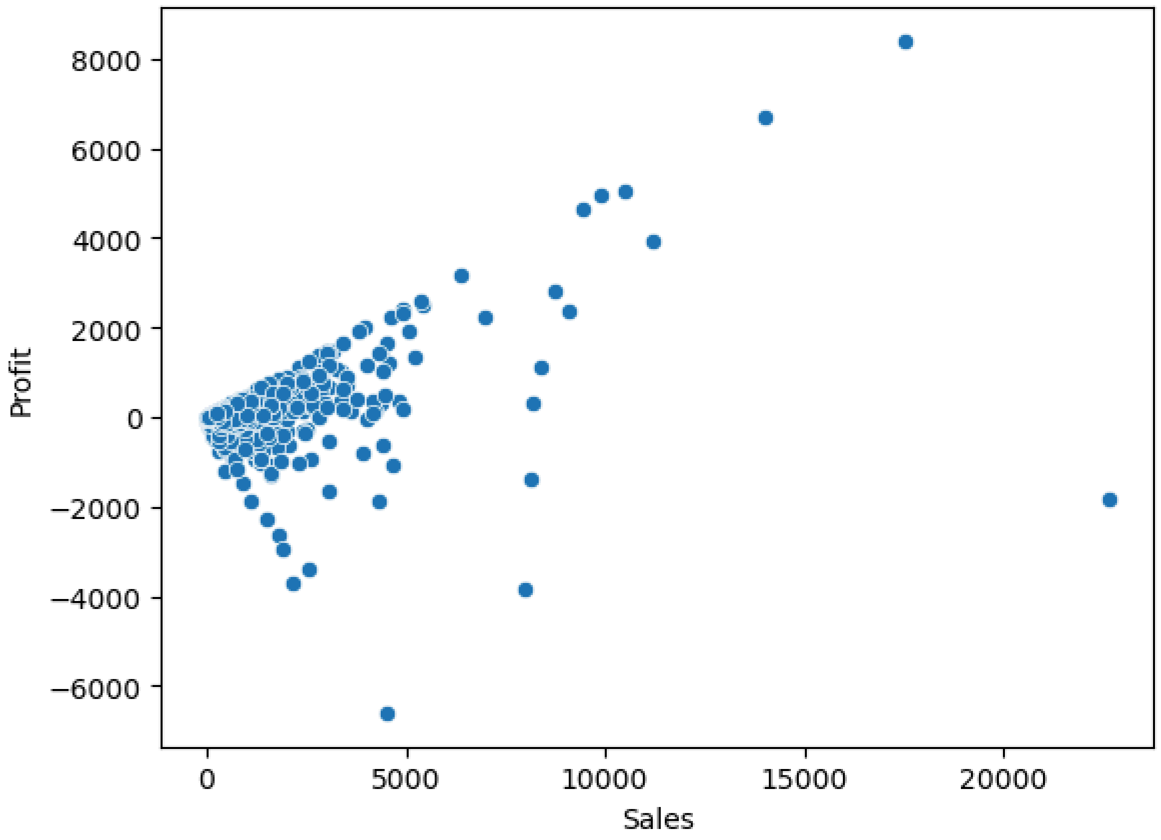
Since Seaborn and plotnine are built on top of matplotlib, I think the stylesheets will work in them as well

Seaborn

In [11]:
```
import seaborn as sns
```

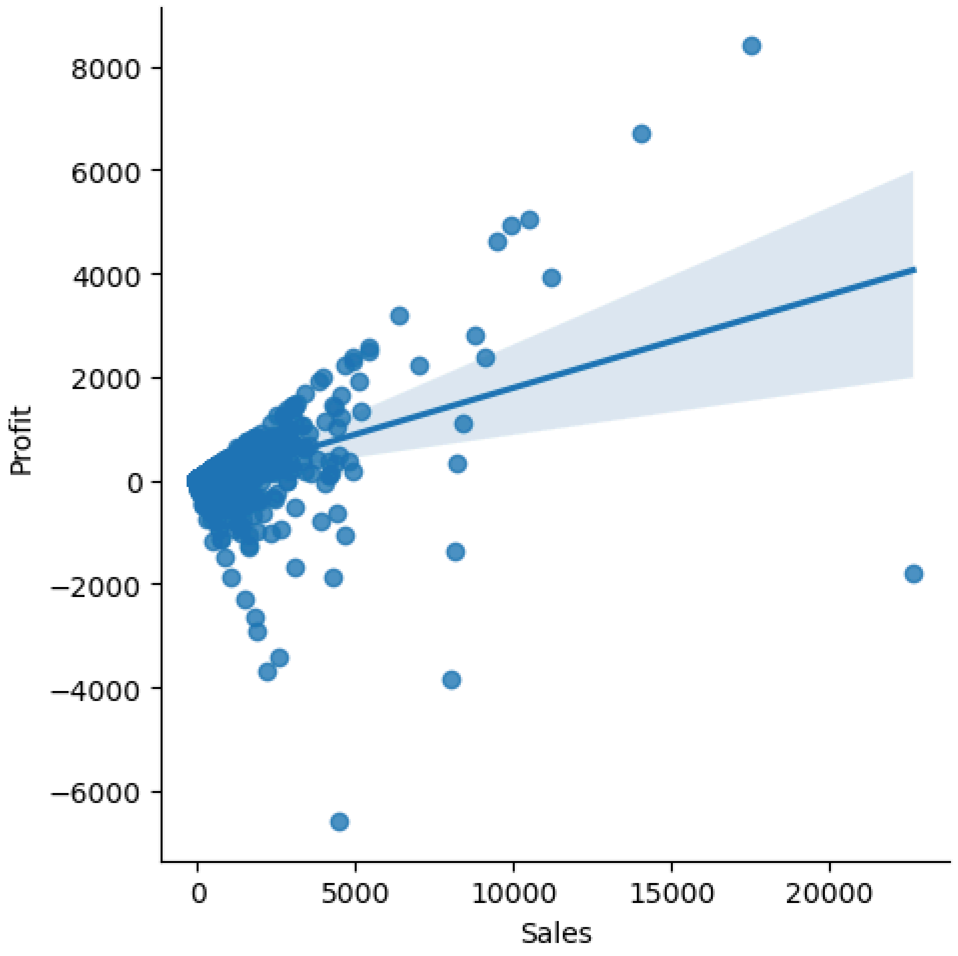
In [12]:
```
#here is a basic scatterplot using seaborn
# we pass in the data frame and specify the variables along the x and y axis

sns.scatterplot(df,x="Sales", y="Profit")
```

Out[12]:
```
<Axes: xlabel='Sales', ylabel='Profit'>
```



In [13]:
```
#this is a seaborn plot of a linear model of Profit vs Sales
# it adds the linear regression line and a confidence interval on the regression li

sns.lmplot(df,x="Sales", y="Profit")
```

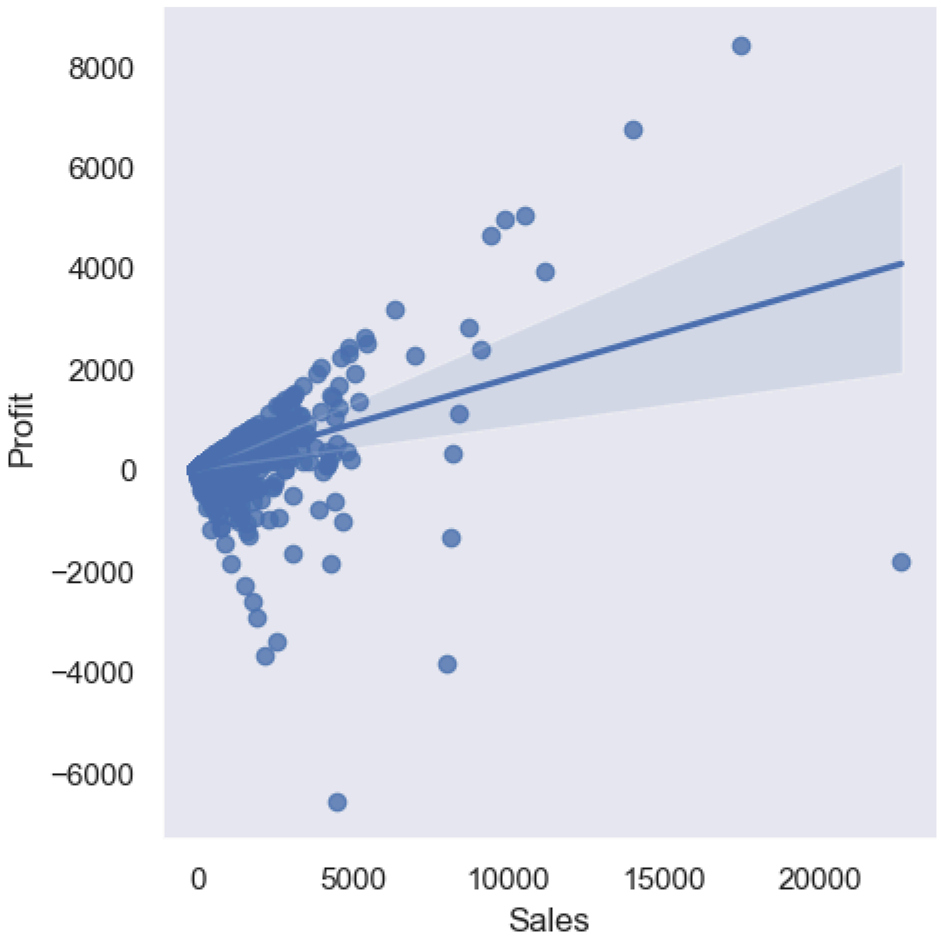
Out[13]:
```
<seaborn.axisgrid.FacetGrid at 0x2b11a423f20>
```



Seaborn has axis themes to customize the style of the graph

```
In [14]: sns.set_theme(style="dark")
         sns.lmplot(df,x="Sales", y="Profit")
```

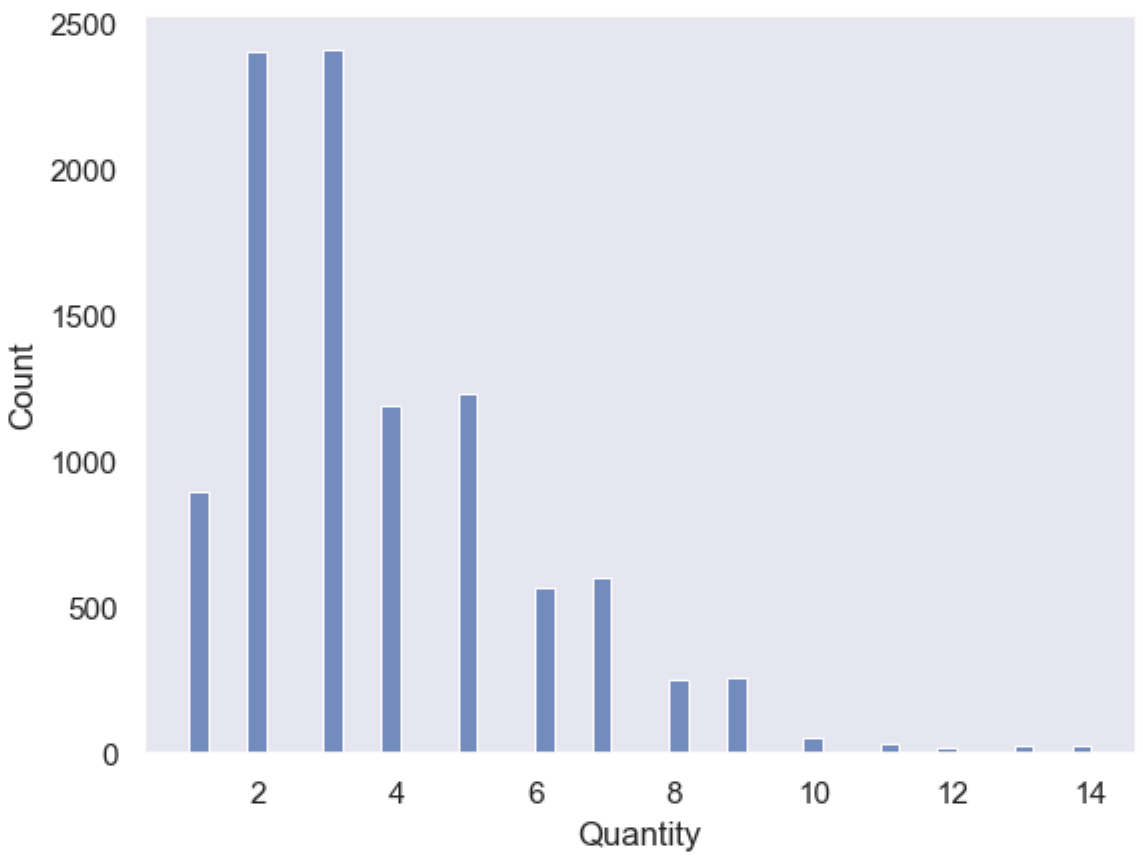
```
Out[14]: <seaborn.axisgrid.FacetGrid at 0x2b11a6cc8c0>
```



In [15]:
```
# Histogram in Seaborn
```

In [16]:
```
sns.histplot(df,x="Quantity")
```

Out[16]:
```
<Axes: xlabel='Quantity', ylabel='Count'>
```



Seaborn can produce a cumulative distribution plot, this shows the number of rows with the Quantity value at or below a specific value, this is a cumulative sum of the values in the histogram

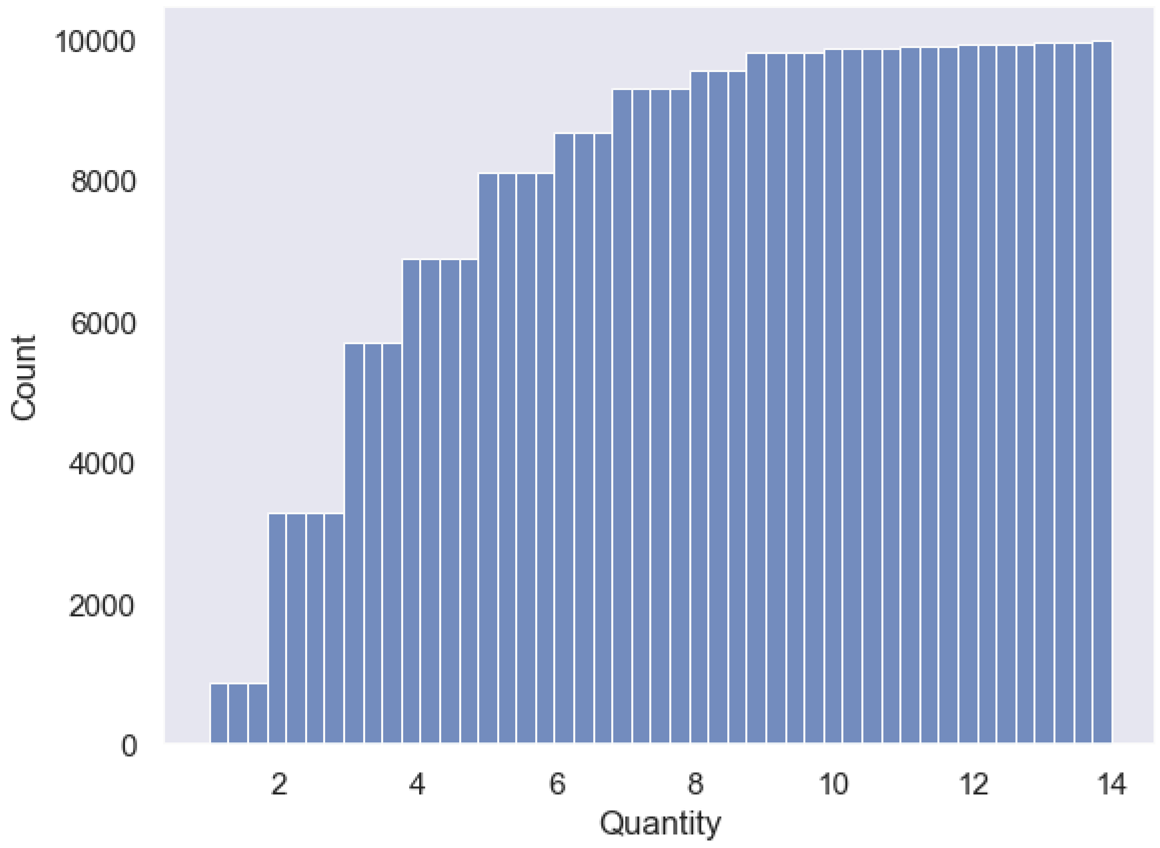
Statistically, this is a cumulative distribution function- literally the sum or integral of a distribution function

The histogram is showing a distribution function itself

Key idea here is that Seaborn easily creates the cummulative distribution plot for us

```
In [17]: sns.histplot(df,x="Quantity",cumulative=True)

         # Visually represents the probability that a random variable takes on a value less
```

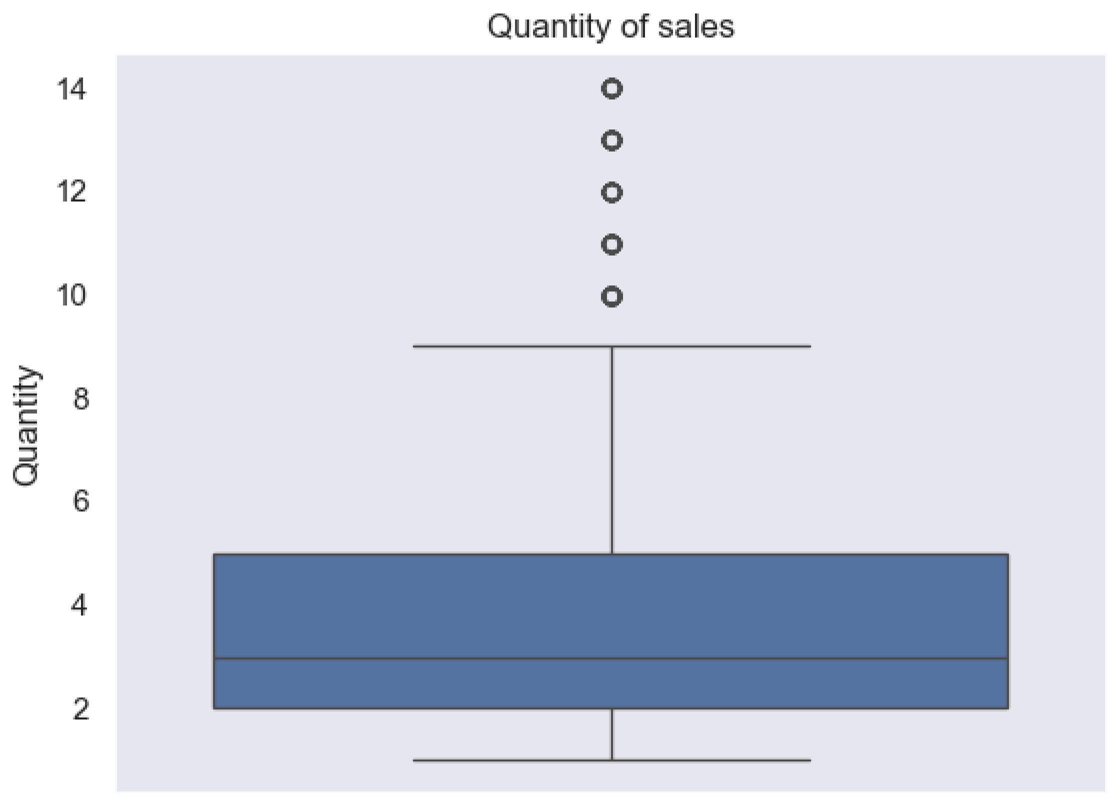
```
Out[17]: <Axes: xlabel='Quantity', ylabel='Count'>
```



Seaborn boxplot

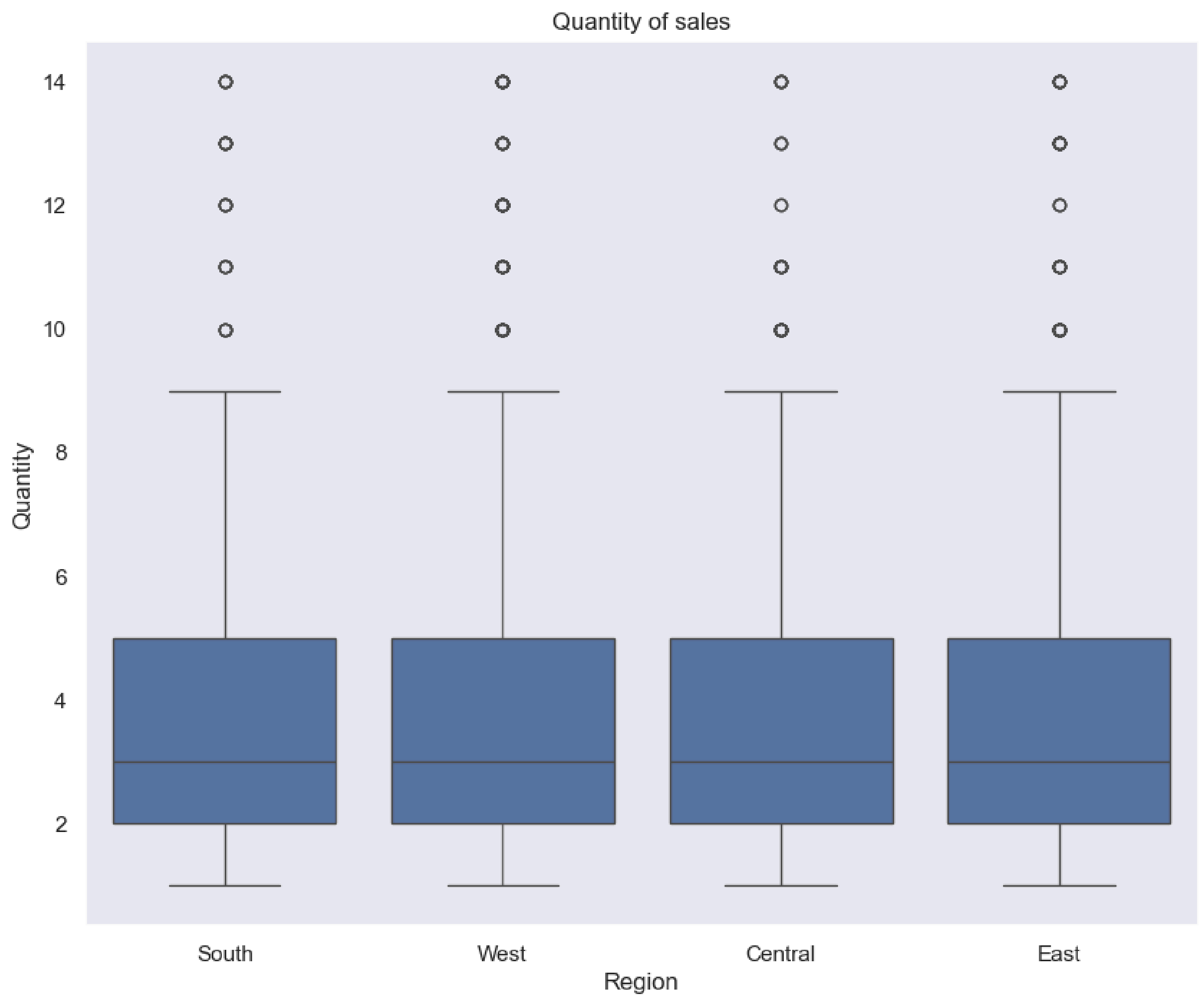
```
In [18]: sns.boxplot(df,y="Quantity", orient="Vertical").set(title="Quantity of sales")
```

```
Out[18]: [Text(0.5, 1.0, 'Quantity of sales')]
```



In [19]:
```
# plotting boxplots by category

# this command allows control of the figure size,  experiment with the parameters
# the first boxplot I tried was very small

plt.figure(figsize=(10, 8))

sns.boxplot(df,x='Region',y="Quantity",  orient="v").set(title="Quantity of sales")
```

Out[19]:
```
[Text(0.5, 1.0, 'Quantity of sales')]
```



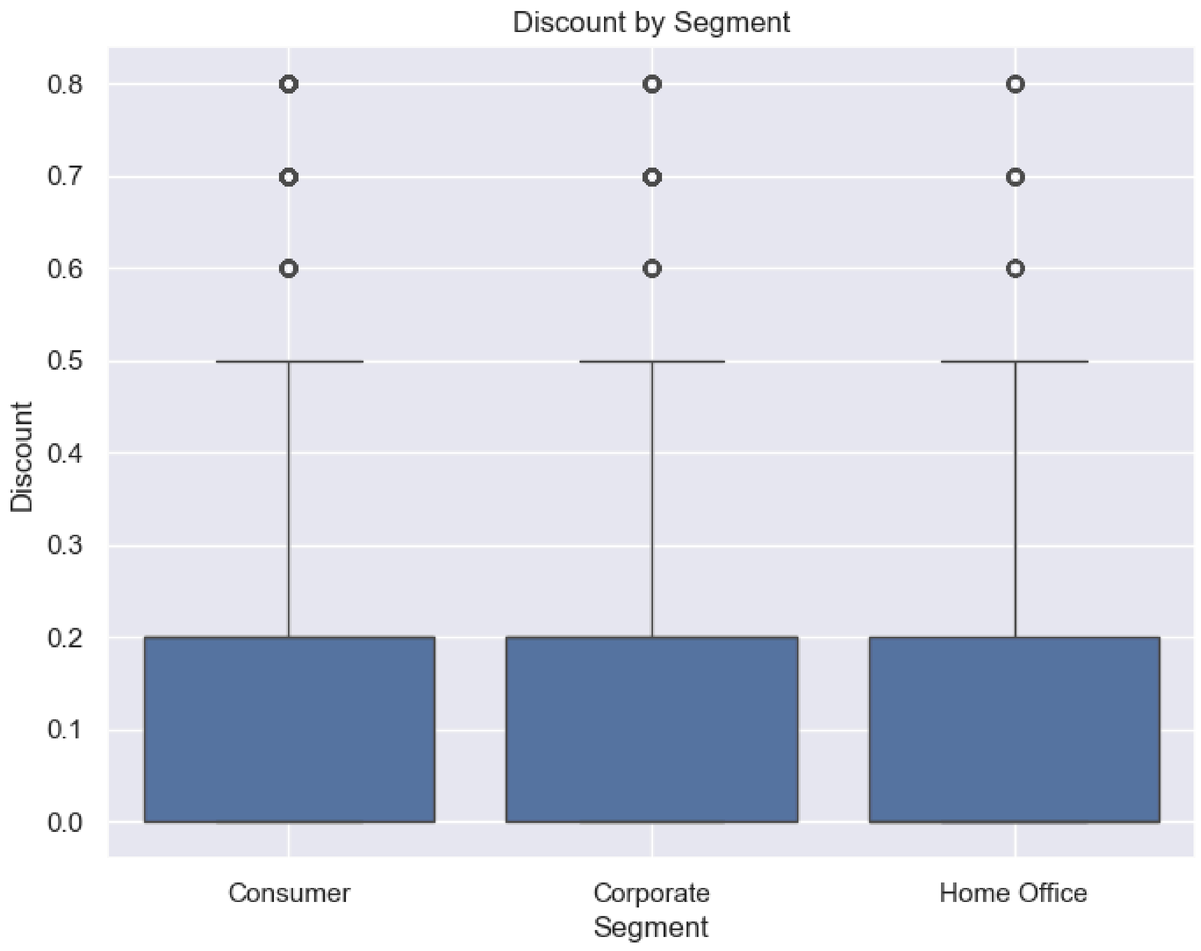
Question/Action

Do a boxplot showing Discount by Segment, change the title

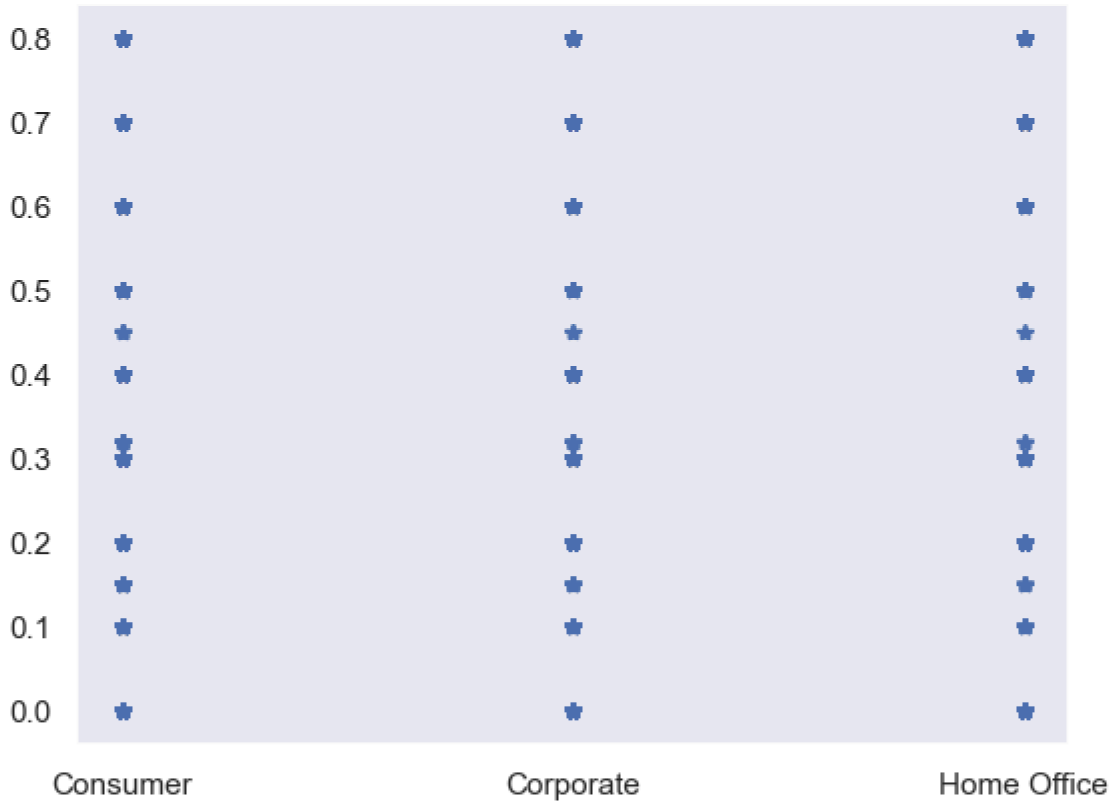
Use

plt.figure(figsize=(8, 6))

In [20]:
```
plt.figure(figsize=(8, 6))
sns.boxplot(data=df, x="Segment", y="Discount").set(title="Discount by Segment")
plt.xlabel("Segment")
plt.ylabel("Discount")
plt.grid(True)
plt.show()

# They're all the same?

plt.plot(df.Segment, df.Discount,"b*")

# Ok weird
```


Discount by Segment
Discount
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Consumer
Corporate
Home Office
Segment

```
Out[20]: [<matplotlib.lines.Line2D at 0x2b11b894230>]
```

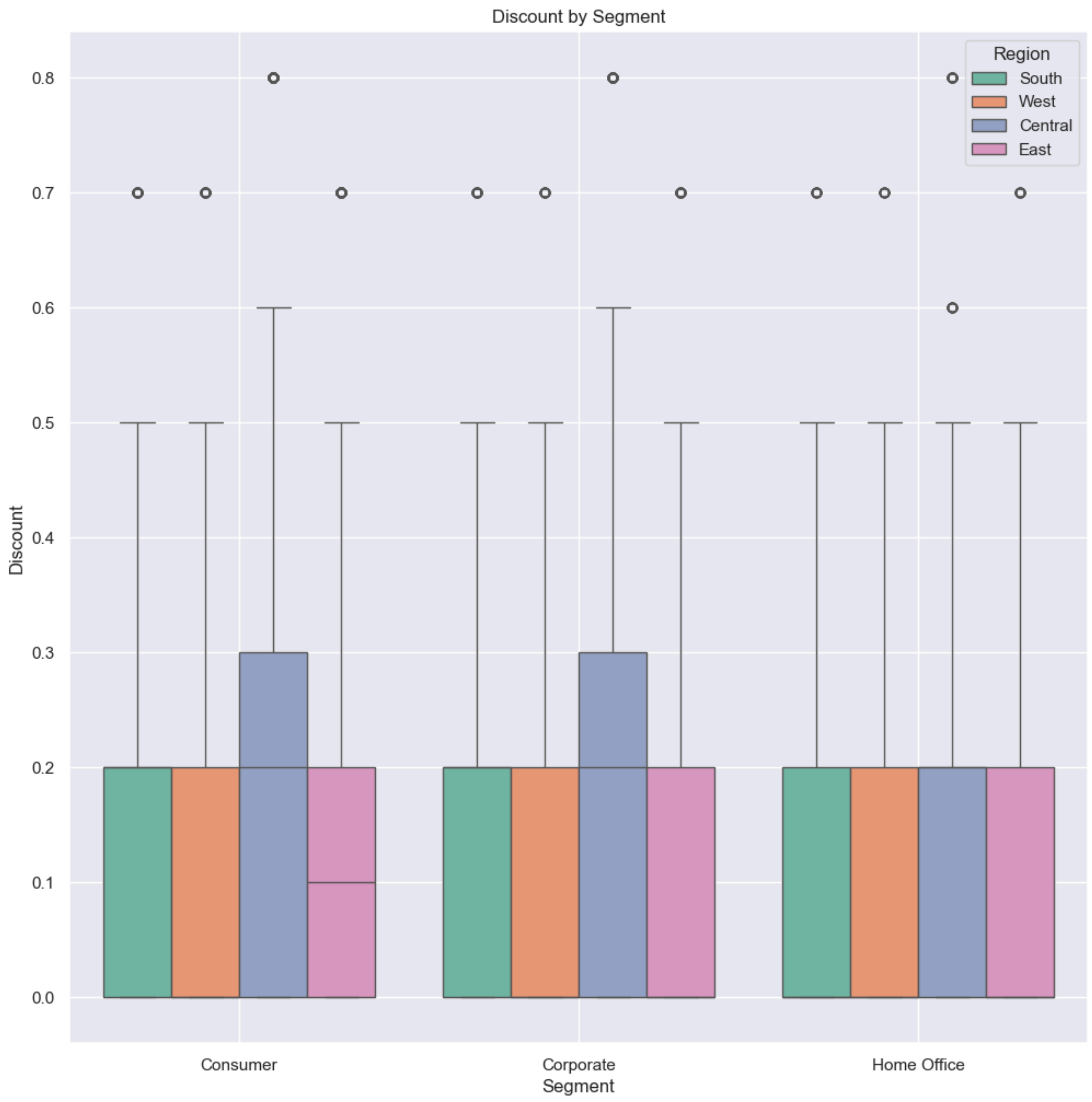

0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
Consumer
Corporate
Home Office

Question/Action

Modify your boxplot above using the hue= command to color code by region

Explain what you see about the nature of discounts based on the plot

In [21]:
```
plt.figure(figsize=(12, 12))
sns.boxplot(data=df, x="Segment", y="Discount", hue="Region", palette="Set2").set(t
plt.xlabel("Segment")
plt.ylabel("Discount")
plt.grid(True)
plt.show()

# It seems that there is more variation in discount in the Central region, but only
# Overall, there are some noticeably higher discounts in the Central region.
# Boxplots might not be the best representation of this data.
# Data with a natural zero often exhibits positive skewness (a long tail to the rig
# Boxplots can visualize skewness, but if a large portion of your data is clustered
# If many of your data points are exactly zero, a boxplot won't explicitly show thi
```



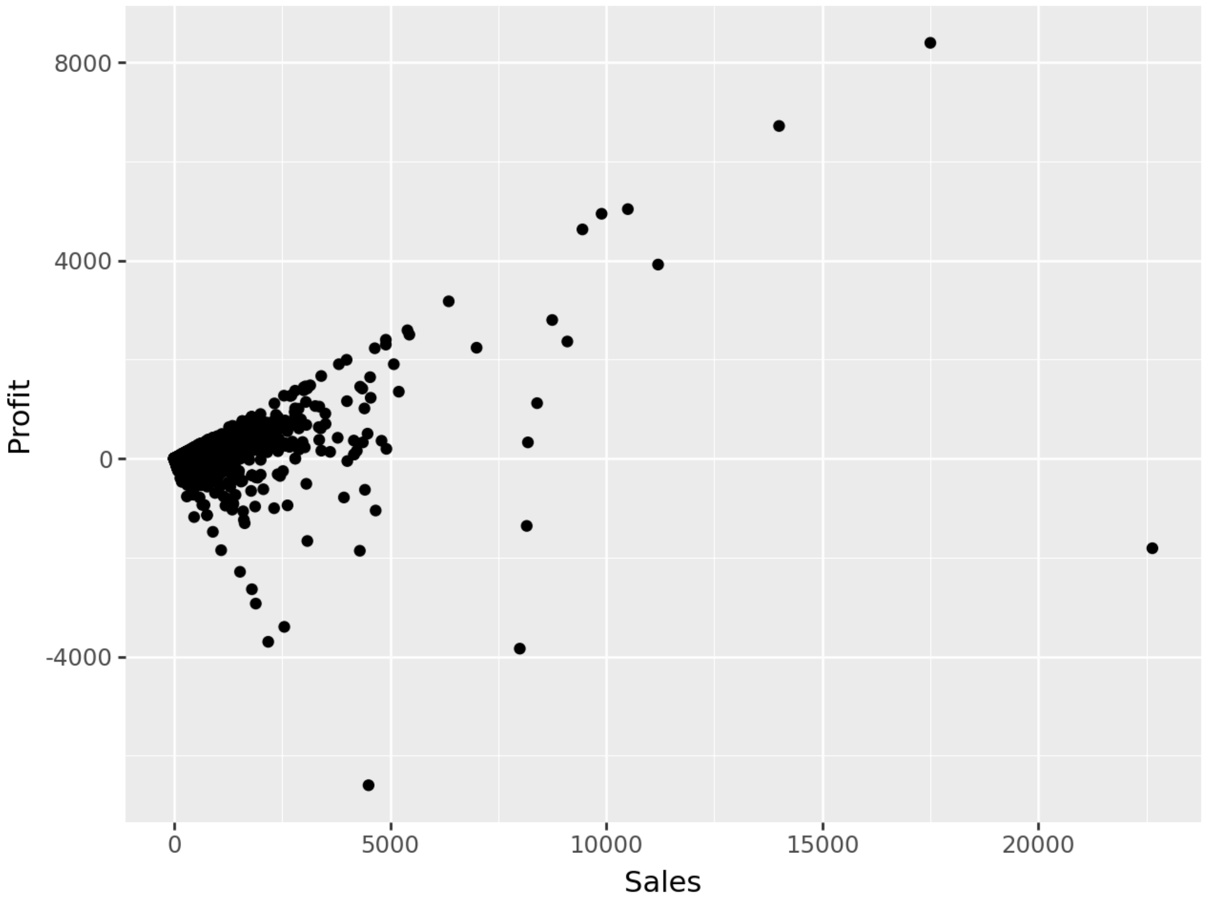
Plotnine

This is a plotting package based on the grammar of graphics, it works pretty much like ggplot in R

In [22]:
```
# if this cell does not run, use the conda install process in the next cell to inst
import plotnine as p9
```

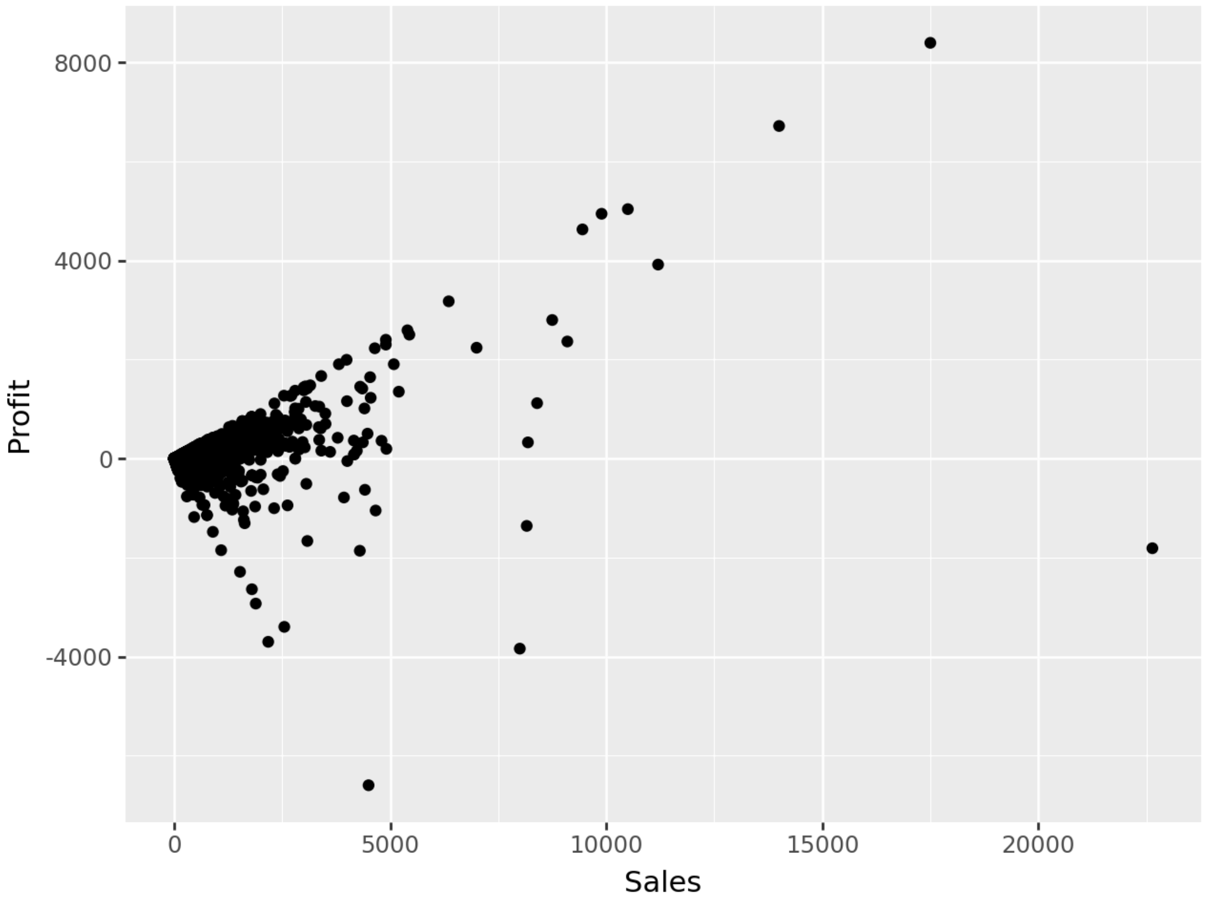
In [23]:
```
#run this to install plotnine- only run it once on a machine

#it is probably easiest to cut and paste the line below into an Anaconda command wi

#       conda install conda-forge::plotnine
```

In [24]:
```
p9.ggplot(df,p9.aes(x="Sales",y="Profit"))+p9.geom_point()
```

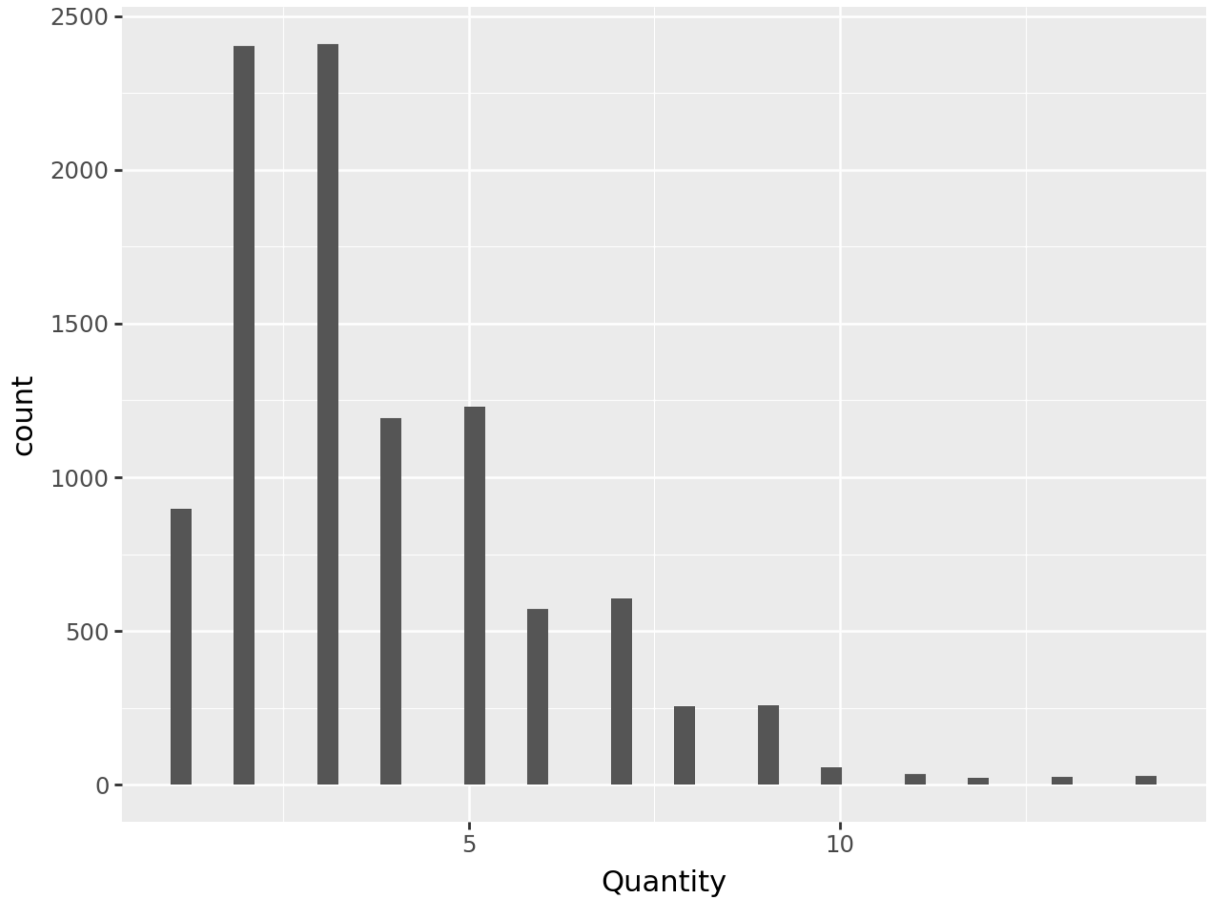


```
In [25]: # simplifyihg the call, import the functions separately, making typing easier

         from plotnine import ggplot,aes,geom_point, geom_boxplot, geom_histogram
```

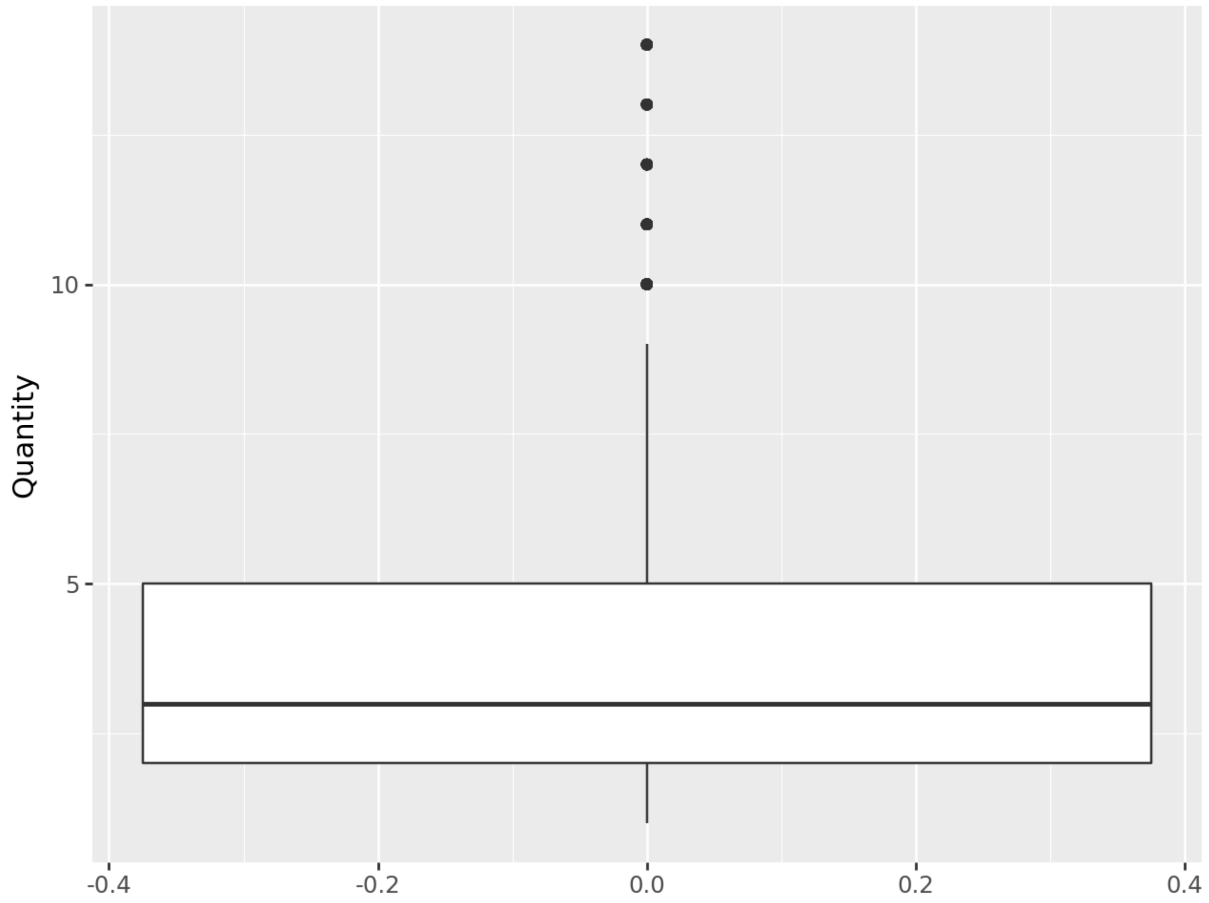
```
In [26]: ggplot(df,aes(x="Sales",y="Profit"))+geom_point()
```



In [27]:
```
ggplot(df,aes(x="Quantity"))+geom_histogram(bins=47)
```



```
In [28]: ggplot(df,aes(y="Quantity"))+geom_boxplot()
```

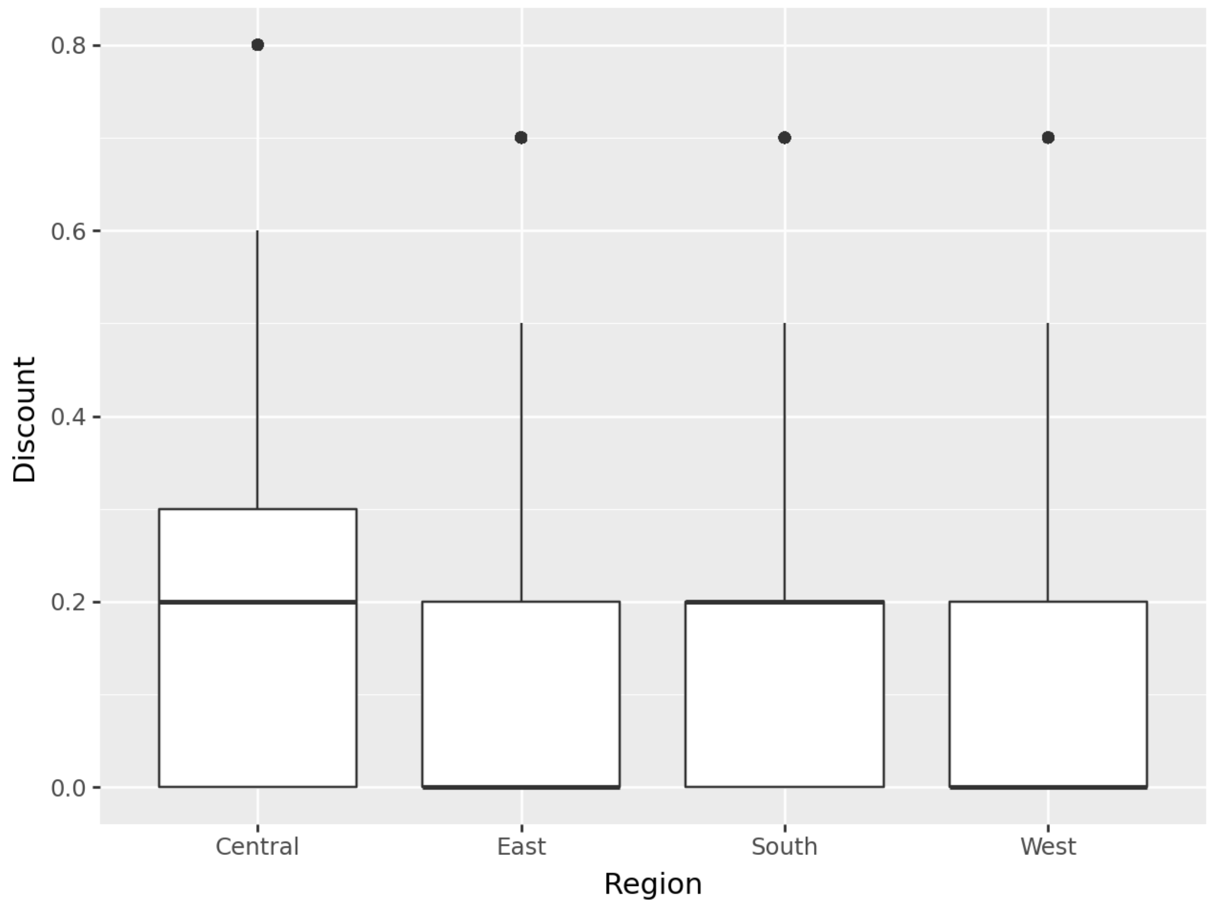


Question/Action

use p9/ggplot to plot Discount vs Region

in the aes() just set y= and x=, the same way it works in r

```
In [29]: ggplot(df,aes(y="Discount",x="Region"))+geom_boxplot()
```

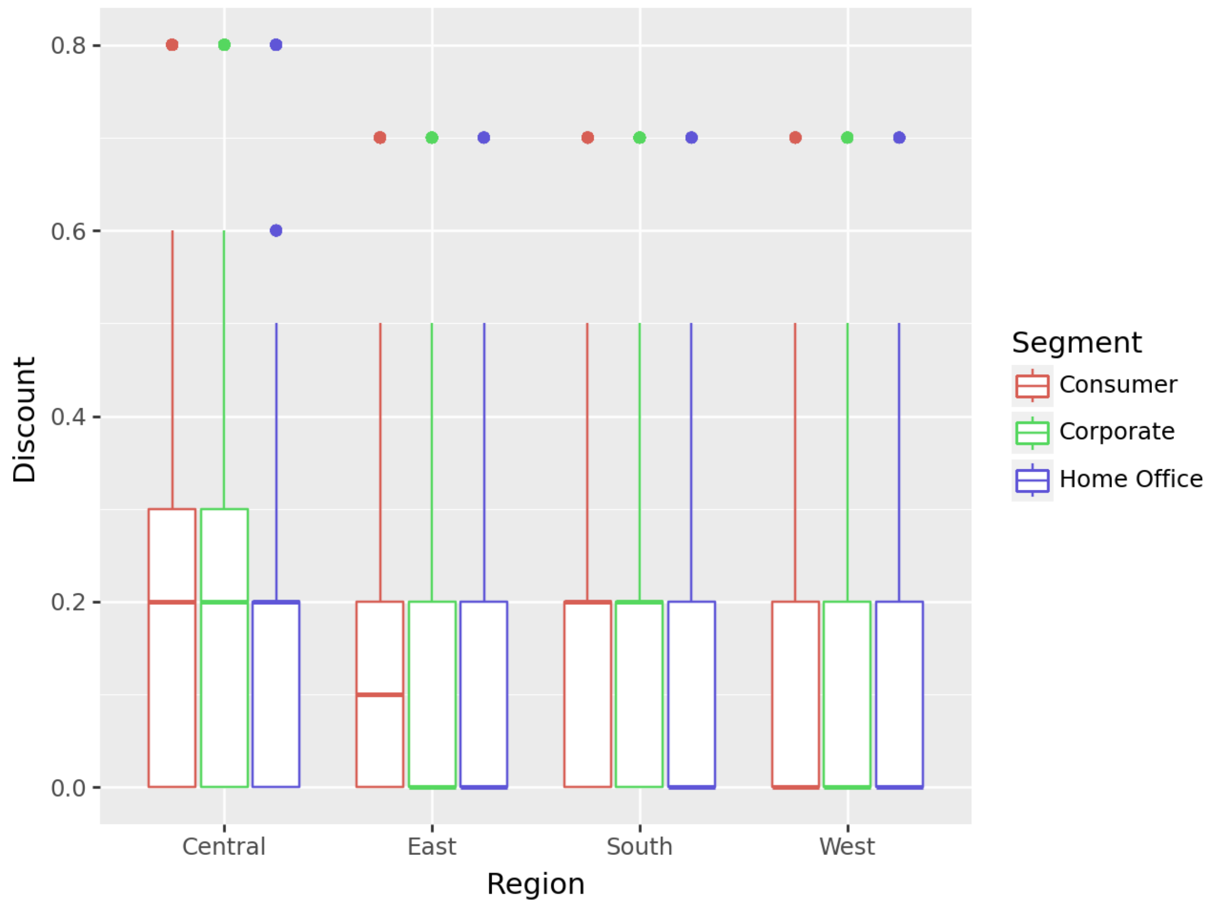


Question/Action

use p9/ggplot to plot Discount vs Region, but color code by segment

It works just like ggplot in R, so just cut and paste, then modify the code to add color= in the aes()

```
In [31]: ggplot(df,aes(y="Discount",x="Region",color="Segment"))+geom_boxplot()
```



Working with graphics in Python

1.) I've just shown you the basics, you will need to look up how to do different plots

2.) The galleries for these tools are nice and show you how to produce different types of plots

https://matplotlib.org/stable/plot_types/index.html

https://seaborn.pydata.org/examples/index.html

https://r-graph-gallery.com/all-graphs.html

3.) You will take a course in your Master's on visualization, so you will see this again

4.) Which package to use? I like ggplot, but plotnine looks a little cumbersome in some ways. Seaborn produces cool images quickly, but then I know matplotlib pretty well....

```
See what packages others in your organization are using

You can always load data into Tableau, PowerBI or even Excel to
create presentation grade graphics-
lots of options
```

5.) You can always write the results from a complex analysis to a csv file, open it in R and use the graphics in R