Lesson 7 Plotting with Seaborn   Last update Feb 14, 2021

Seaborn offers much more plotting capabilities then **pyplot**. Seaborn actually uses pyplot as a base, but provides much more enhancing plotting capabilities.

To use pyplot and seaborn you import them as follows:

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

You need to have seaborn version 11.0 or greater installed

To install on Jupiter notebook or on Spyder:

```
conda install -c anaconda seaborn==0.11.0
```

To install on a regular python:

```
pip install seaborn==0.11.0
```

Types of Plots available with Seaborn:

relation plots

Relation plots makes use of two other axes functions for visualizing statistical Relationships which are:

- scatterplots
- lineplots

lineplots

A line plot is used to plot lines between 2 variables. As an example we will use a data frame filled with the amount of money spent per day for groceries as our data.

```
import pandas as pd
data=pd.DataFrame({'Day':[1,2,3,4,5,6,7],'Grocery':[30,80,45,23,51,46,76]})
print(data)
```

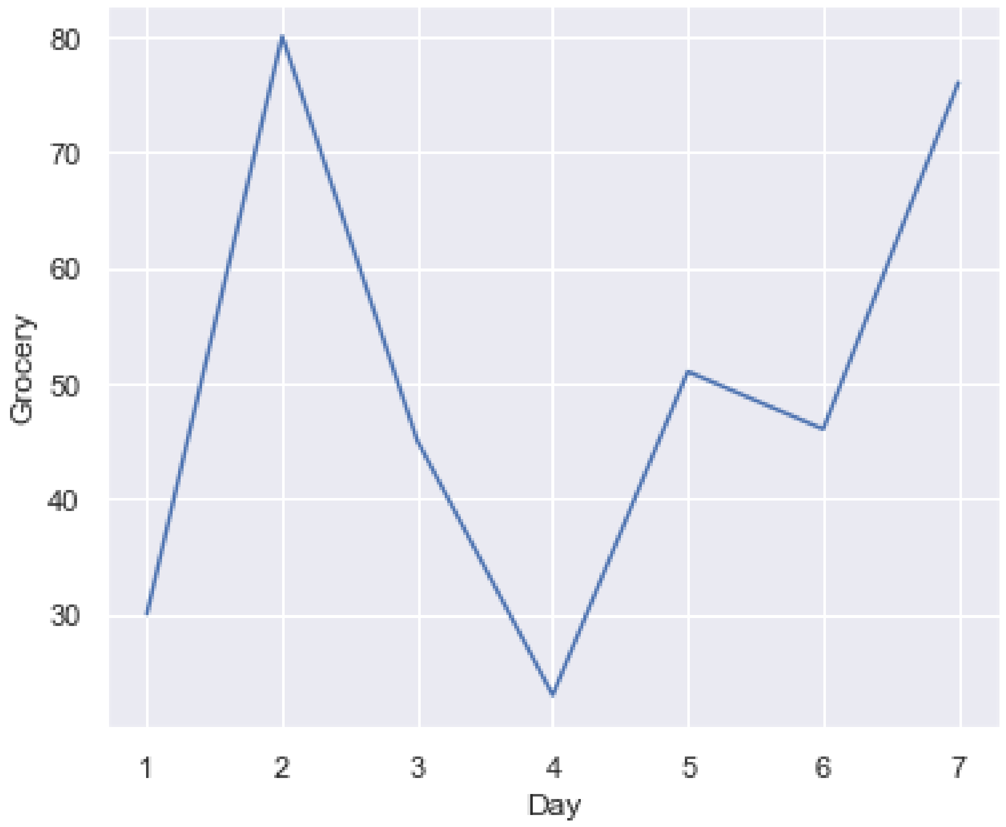
```
   Day  Grocery
0    1       30
1    2       80
2    3       45
3    4       23
4    5       51
5    6       46
6    7       76
```

We can use the pyplot **plt.figure** function to set our plot size to 6 by 5 inches

```
plt.figure(figsize=(6,5))
```

We then plot the data with the seaborn lineplot function

```
sns.lineplot(x="Day", y="Grocery",  data=data)
plt.show()
```



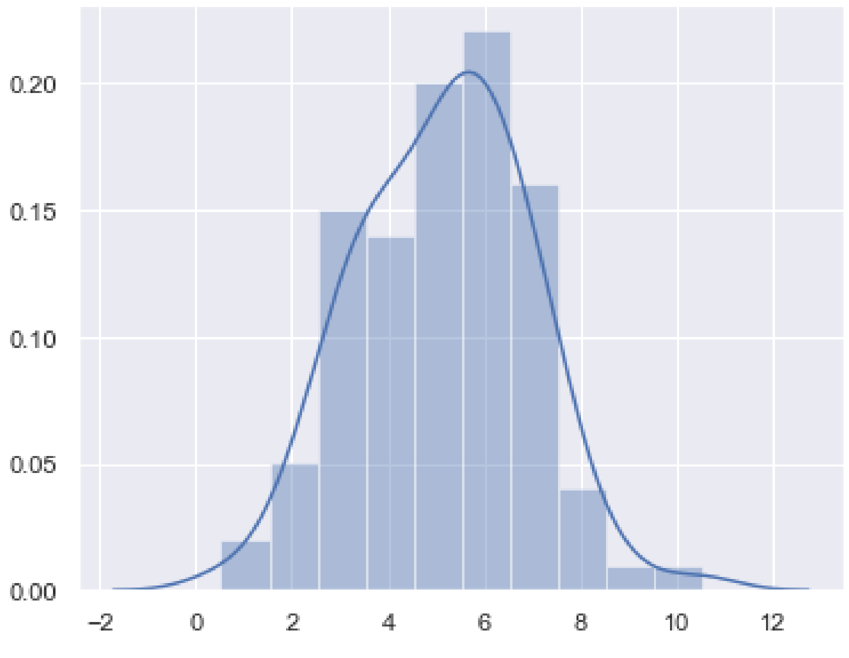
Plotting Univariate Distributions:

Univariate means "one variable" (one type of data like temperature). If you have two sets of data, such as population sales vs area, then it is called "**Bivariate Data"**

To plot **univariate distributions** you use the **displot()** function as follows:

We can plot the normal distribution curve of 100 points centered at 5 with deviation of 2 as follows using the numpy **random.normal** function. We have set **kde** equal to True so we can also plot the kernel density estimation (kde). The kernel density estimation is used create a smooth curve around the normal distribution of data..

```
import numpy as np
data = np.random.normal(loc=5,size=100,scale=2)
sns.displot(data,kde=True);
plt.show()
```



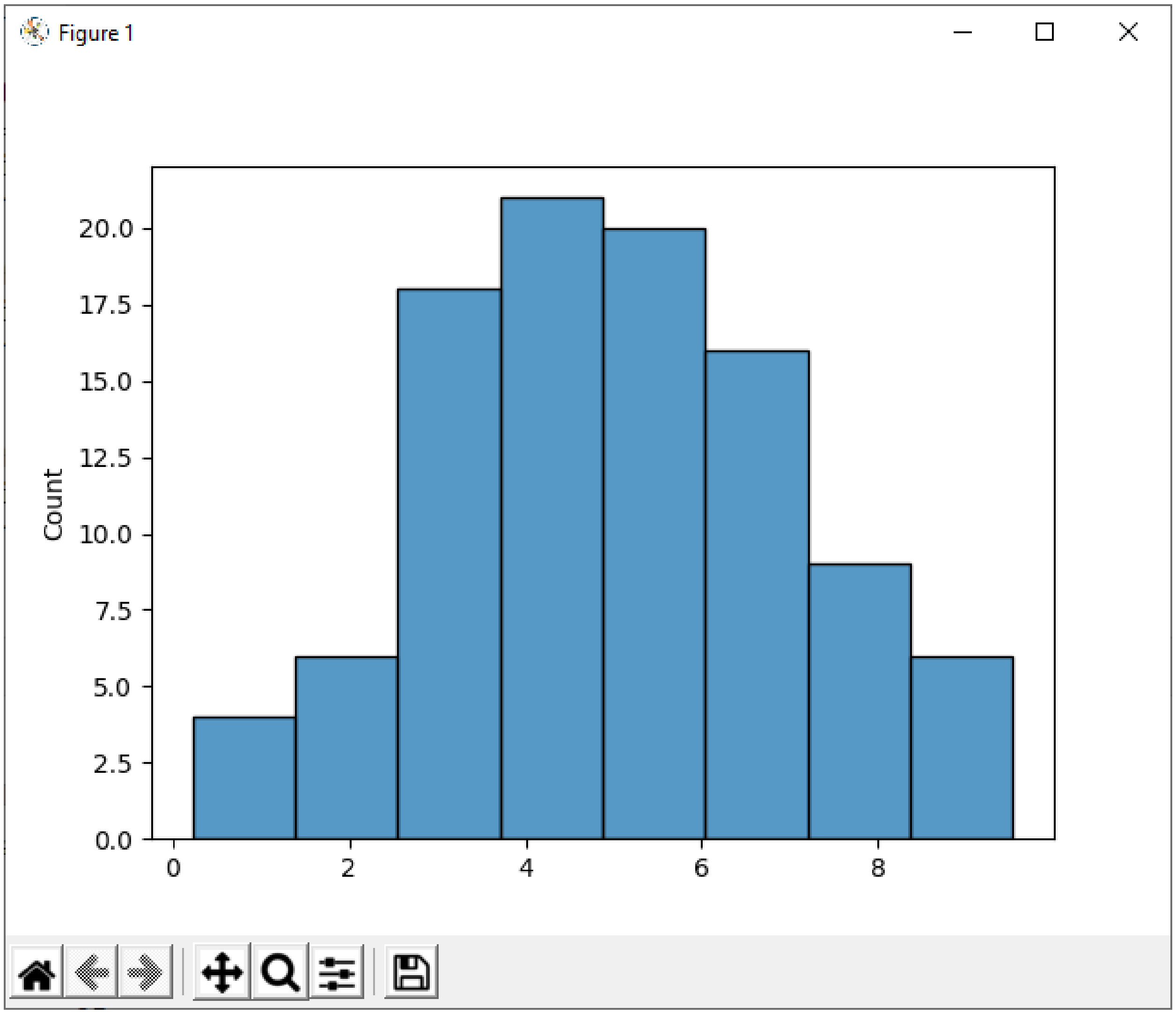
We have plotted a graph using displot for normal distribution curve of 100 points centered at 5 with deviation of 2.

Plotting histogram

A histogram represents the distribution of data by forming bins along the range of the data and then bars are drawn to show the number of observations (frequency) that fall within in each bin.

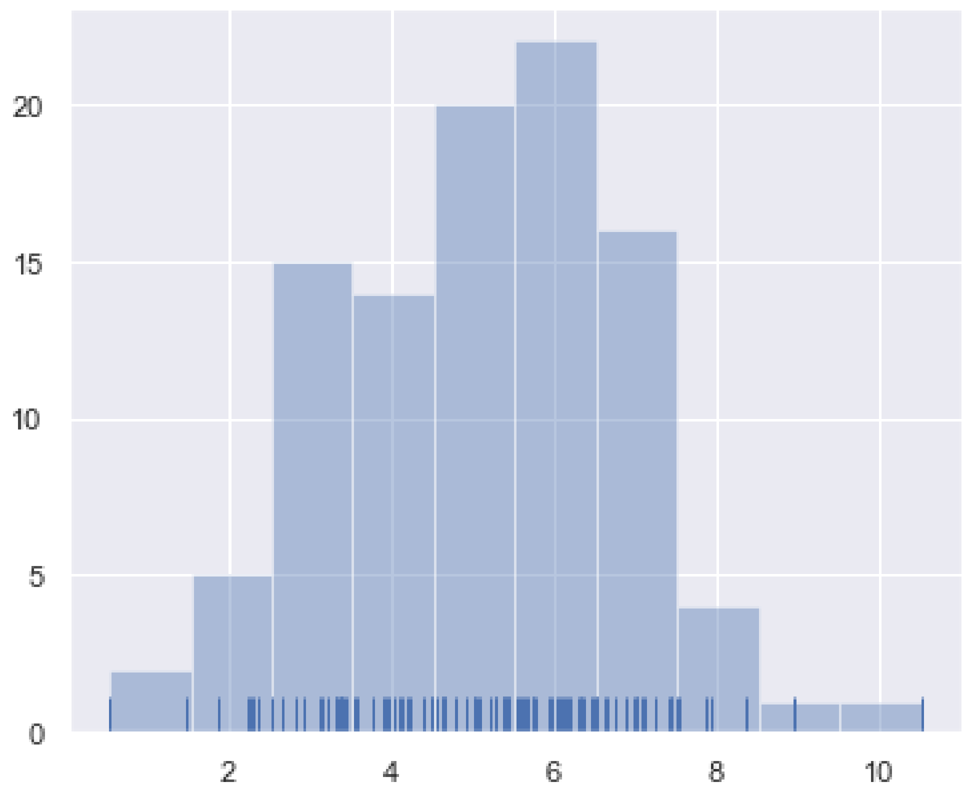
We can use the seaborn **histplot** function to print a histogram.

```
sns.histplot(data)
plt.show()
```



We can also use the displot function to plot a histogram by removing the kernel density estimation (kde) curve by setting **kde=False** (not including kde). We have added an additional **rug** plot instead, which draws a small vertical tick for each observation at the bottom of the plot.

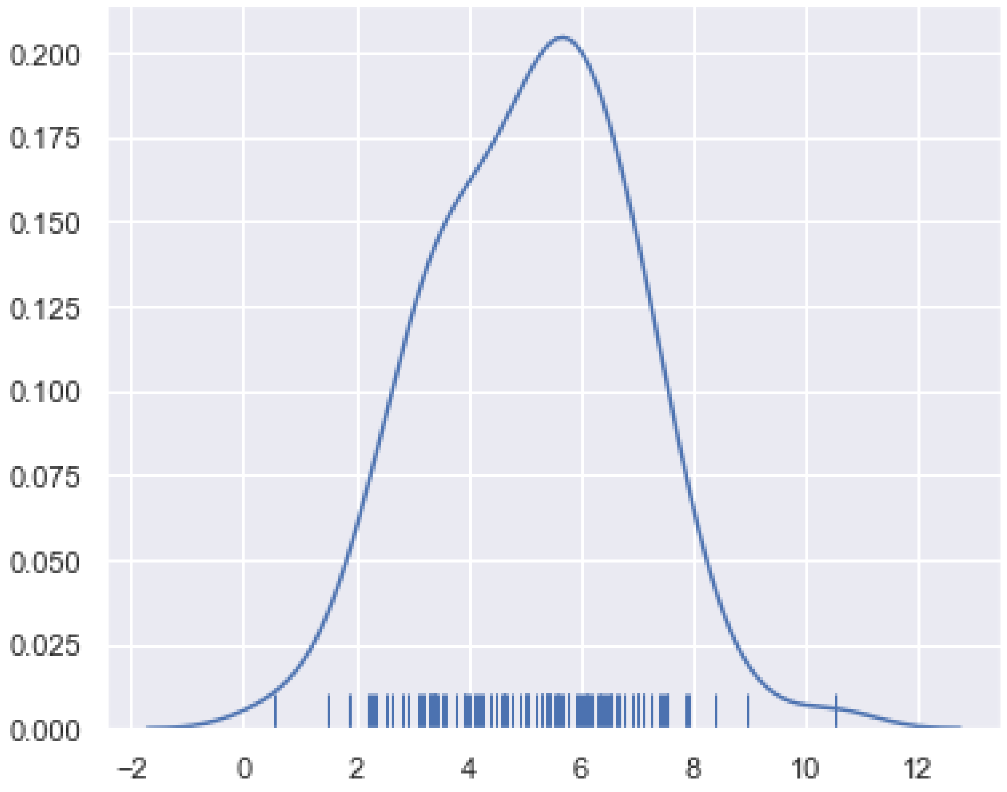
```
data = np.random.normal(loc=5,size=100,scale=2)
sns.displot(data, kde=False, rug=True);
plt.show()
```



Plotting Kernel density estimation KDE

The **kernel density estimate KDE** is used to plot the shape of a distribution. The KDE plot encodes the density of observations on one axis with height along the other axis. We use the seaborn **distplot** function with **kind="kde"** to plot the kde.

```
# plotting Kernel density estimation KDE
data = np.random.normal(loc=5,size=100,scale=2)
sns.displot(data, kind="kde", rug=True);
plt.show()
```

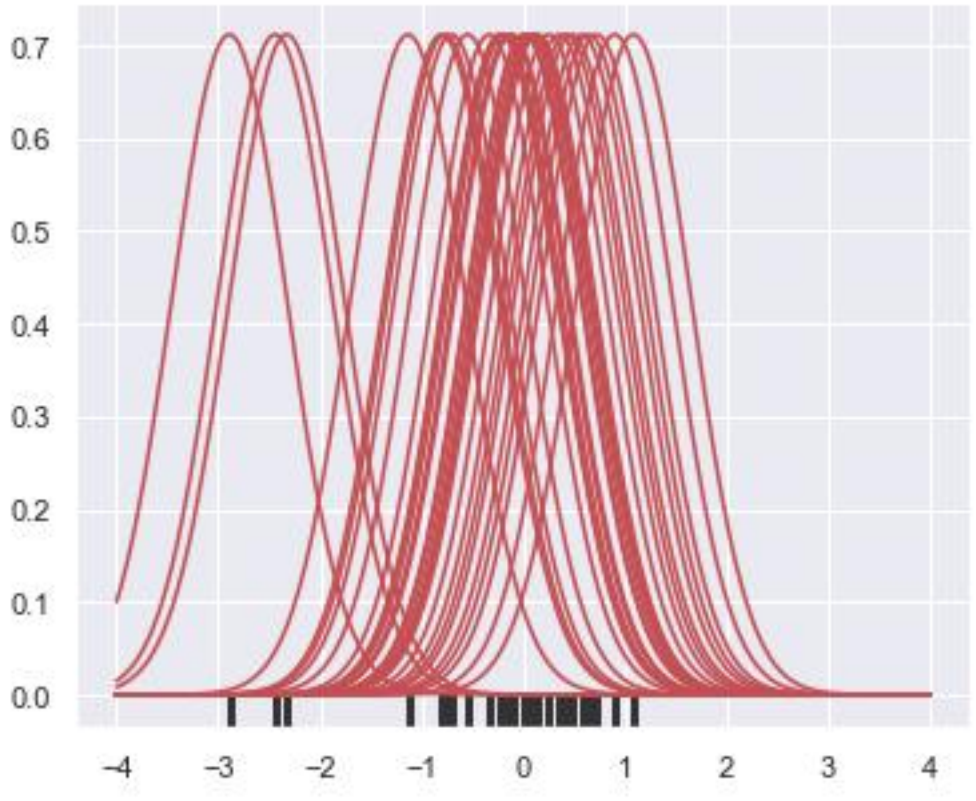


Drawing a KDE

In drawing a KDE each observation is first replaced with a normal (Gaussian) curve centered at that value.

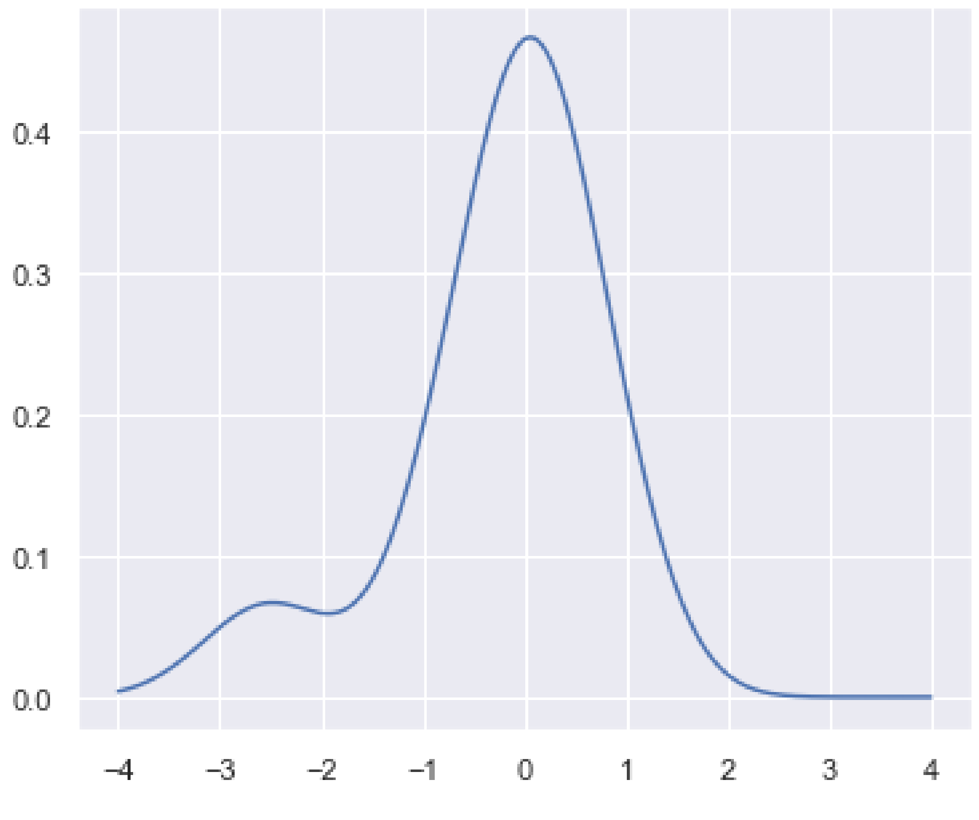
```
# drawing kde
data = np.random.normal(0, 1, size=30)
bandwidth = 1.06 * data.std() * data.size ** (-1 / 5.)
support = np.linspace(-4, 4, 200)

# make kernels
import scipy
kernels = []
```

```
for x in data:
   kernel = scipy.stats.norm(x, bandwidth).pdf(support)
   kernels.append(kernel)

# plot kernels
for kernel in kernels:
   plt.plot(support, kernel, color="r")

sns.rugplot(data, color=".2", linewidth=3);
plt.show()
```



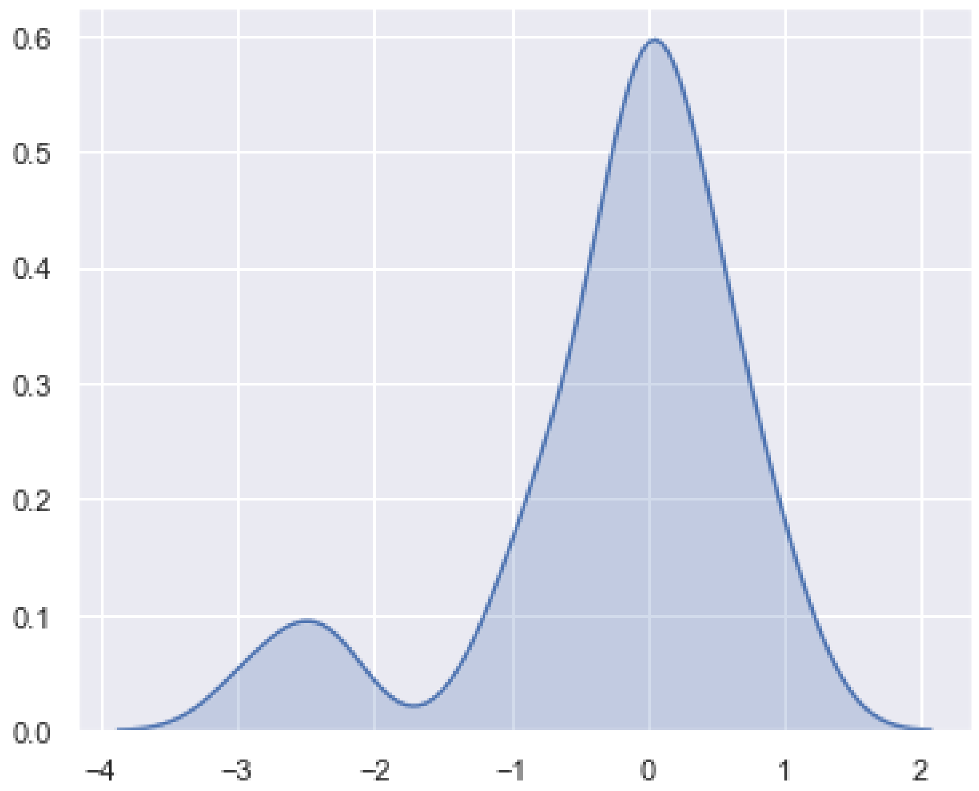
Next, these curves are summed to compute the value of the **density** at each point in the support grid using the numpy **sum** function. The resulting curve is then normalized so that the area under it is equal to 1 by dividing the density by the area under the curve. The area under the curve is calculated using the scipy **integrate.trapz** function to Integrate along the x axis using the composite trapezoidal rule.

```
# plot density from kde sum
plt.figure(figsize=(6,5))

density = np.sum(kernels, axis=0)
density /= scipy.integrate.trapz(density, support)
plt.plot(support, density);

plt.show()
```



We can also plot the **kde** using the seaborn **kdeplot** function. We set shade to True so that he area under the curve is shaded.

```
# plot kde using seaborn kdeplot
sns.kdeplot(data, shade=True);
plt.show()
```



Notice: the seaborn **kdeplot** function results in the same curve.

Fitting parametric distributions

A **displot** is used to fit a parametric distribution to a dataset and visually evaluate how closely it corresponds to the observed data. A parametric distribution is used in statistics when an assumption is made of the way the underlying data is distributed. We use the numpy **random.gamma** function to obtain random gamma data distribution.

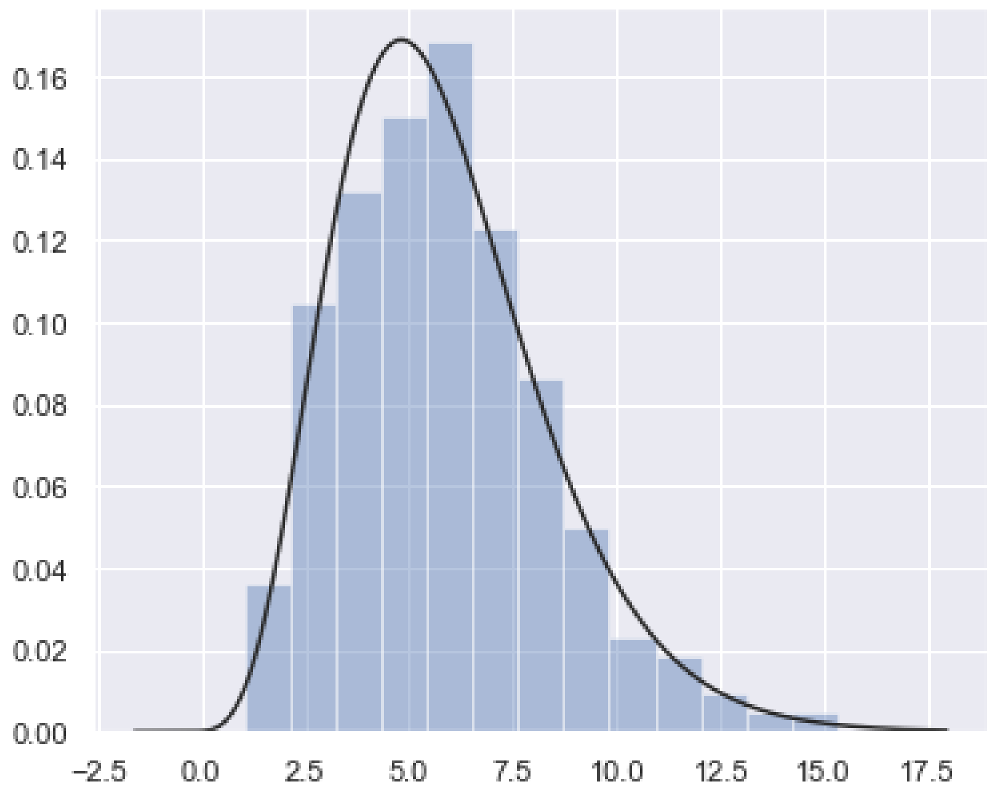
The probability density for the Gamma distribution is

$$p(x) = x^{k-1}\frac{e^{-x/\theta}}{\theta^k\Gamma(k)},$$

where k is the shape and θ the scale, and Γ is the Gamma function.

The **Gamma distribution** is often used to model the times to failure of electronic components, and arises naturally in processes for which the waiting times between Poisson distributed events are relevant

```
data = np.random.gamma(6, size=200)
sns.displot(data,kde=True)
plt.show()
```



Plotting bivariate distributions using Scatter plot

Bi variate data uses two variables like population and area. The seaborn **jointplot** function is used for plotting **bivariate distributions.**

The seaborn **jointplot** function, creates a multi-panel figure that shows both the bivariate (or joint) relationship between two variables along with the univariate (or marginal) distribution of each on separate axes. **Correlation** is a measure used to represent how strongly two random variables are related known . The value of **correlation** takes place between -1 and +1.

Covariance is nothing but a measure of **correlation**. Correlation is when the change in one item may result in the change in another item. Covariance is when two items vary together. **Positive covariance**: Indicates that two variables tend to move in the same direction. **Negative covariance**: Reveals that two variables tend to move in inverse directions.

$$\mathrm{Cov}\,(X, Y) = \frac{\sum (X_i - \bar{X})(Y_j - \bar{Y})}{n}$$

Standard deviation SD of a dataset is a measure of the magnitude of deviations between the values of the observations contained in the dataset. Where variance is the square of standard deviation.

$$SD = \sqrt{\frac{\sum (r_i - r_{avg})^2}{n - 1}}$$

The multivariate normal, multi-normal or Gaussian distribution is a generalization of the one-dimensional normal distribution to higher dimensions. Such a distribution is specified by its **mean** and **covariance** matrix. These parameters are analogous to the mean (average or "center") and variance (standard deviation, or "width," squared) of the one-dimensional normal distribution.

The probability density function for multivariate_normal is

$$f(x) = \frac{1}{\sqrt{(2\pi)^k \det \Sigma}} \exp\left(-\frac{1}{2} (x - \mu)^T \Sigma^{-1} (x - \mu) \right),$$

where μ is the mean, Σ the covariance matrix, and k is the dimension of the space where x takes values.

Where the multivariate normal distribution is a multidimensional generalization of the one-dimensional normal distribution . It represents the distribution of a multivariate random variable that is made up of multiple random variables that can be correlated with each other. Multivarate data has x and y values.

Example of Multivariate data:

```
[[ 0.92632     2.38255891]
 [ 0.92301184  1.04130204]
 [-1.22132762 -0.27817411]
 [ 1.30463088  1.70426224]
 [-0.33744608  1.48893676]
 [-0.82018973 -0.89850038]
 [-0.93802065  1.45596415]
 [-0.8825863   1.66154621]
 [ 0.58593356  2.51111507]]
```

We use the numpy **random.multivariate_normal** function to draw random samples from a multivariate normal distribution. It receives a mean and covariance matrix and data size.

```
mean = [0, 1]
cov =  [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
```

The mean is a vector mean of each of the x and y points mean = [x,y]

mean x	mean y

0	1

The covariance matrix is a square matrix that contains the variances and covariance's associated with x and y. The diagonal elements of the covariance matrix contain the variances of the variables and the off-diagonal elements contain the covariance's between all possible pairs of variables.

	x	y
x	variance	covariance
y	covariance	variance

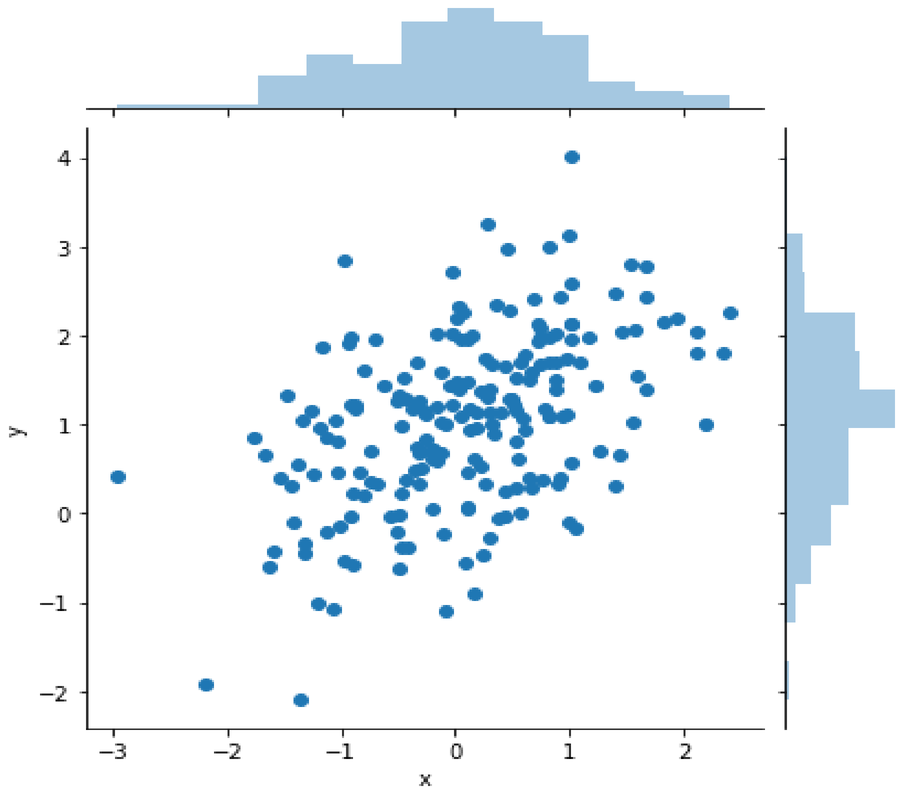
	x	y
x	1	.5
y	.5	1

The covariance matrix is symmetric because the covariance between X and Y is the same as the covariance between Y and X.

The covariance matrix must be positive semi definite. A n xn matrix M is said to be **positive-definite** if the $z^T M z$ scalar is strictly positive for every non-zero column vector z of n real numbers. Where z^T denotes the transpose of z.

We use the **sns.jointplot** function for plotting where the **scatter** plot is the default.

```
# plotting bivariate distributions
mean = [0, 1]
cov =  [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
sns.jointplot(x="x", y="y", data=df);
plt.show()
```



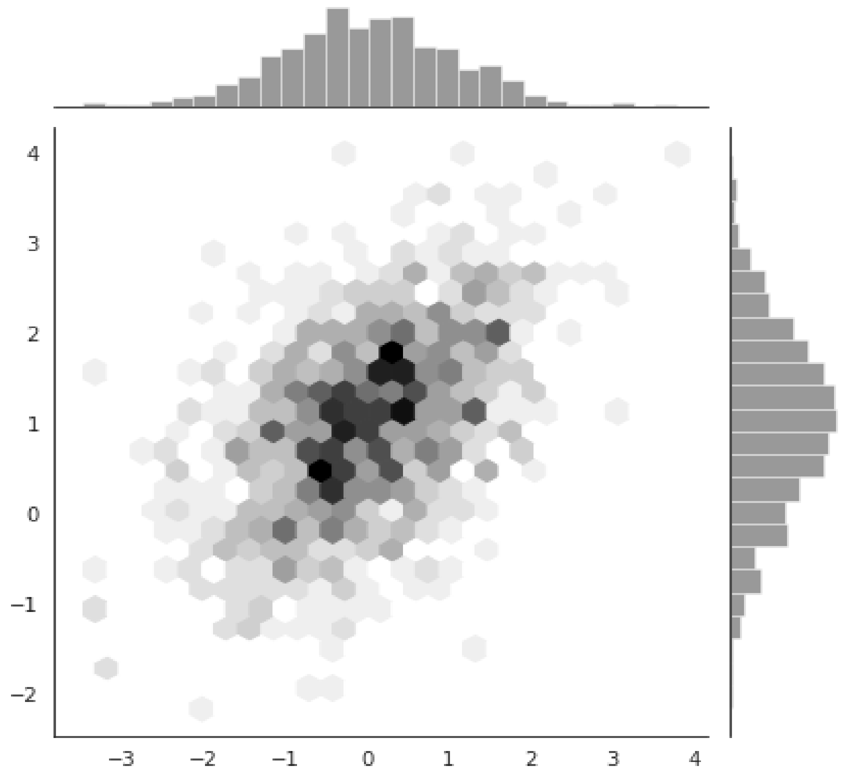
Todo:

Change the mean and covariance matrixes and see what happens.

Hexbin plots

A **"hexbin"** plot is the bivariate analogue of a histogram. It shows the counts of observations that fall within hexagonal bins. This plot works best with relatively large datasets. Again we use the numpy **random.multivariate_normal** function to generate multivariate data sample.

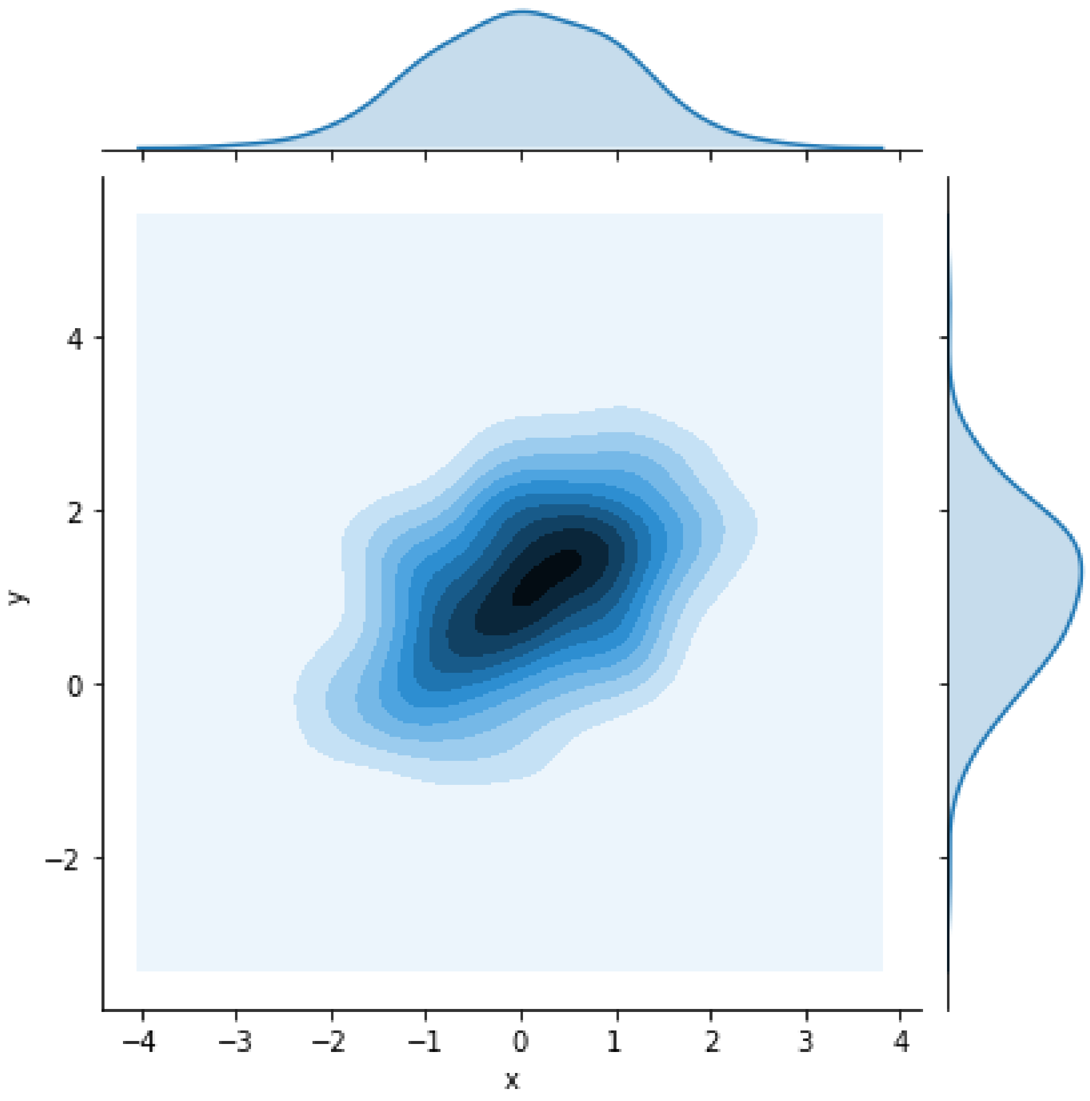
```
# hexbin plot
mean = [0, 1]
cov =  [(1, .5), (.5, 1)]
x, y = np.random.multivariate_normal(mean, cov, 1000).T
with sns.axes_style("white"):
    sns.jointplot(x=x, y=y, kind="hex", color="k");
plt.show()
```



Using Kernel density estimation

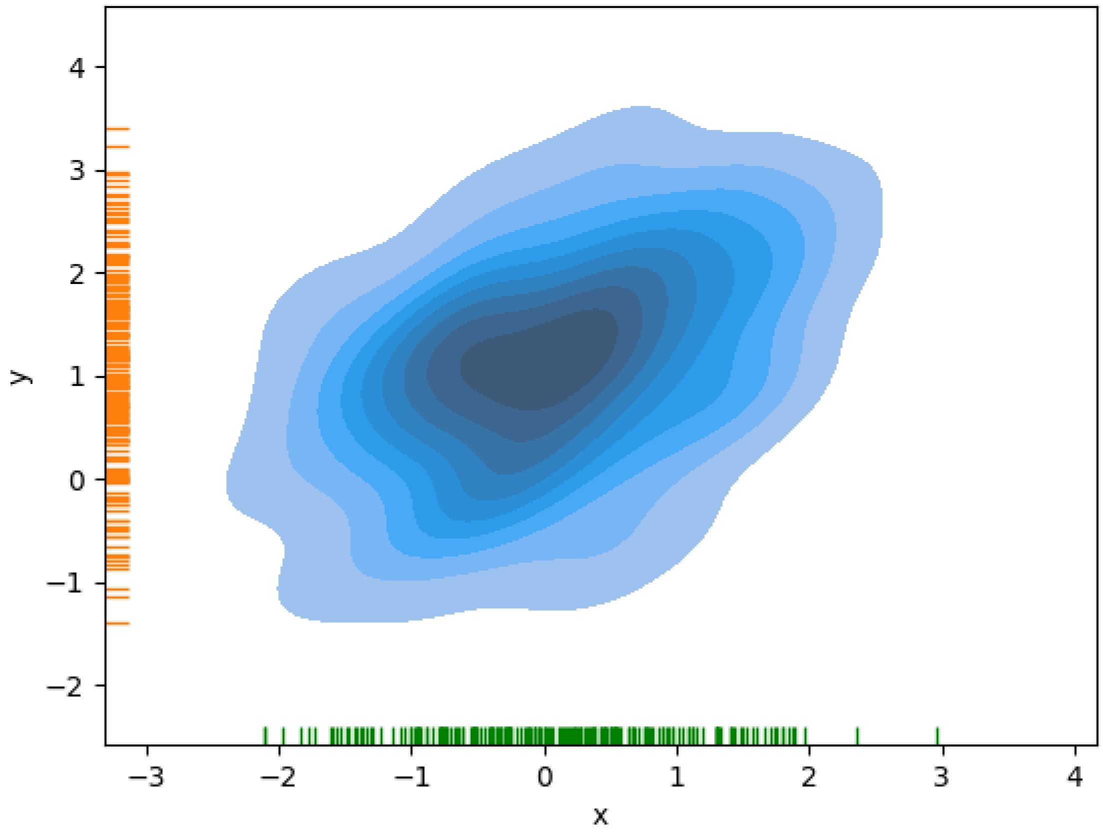
It is also possible to use the kernel density estimation procedure described above to visualize a bivariate distribution. This kind of plot is shown with a contour plot using **jointplot** and **kind** = “kde”.

```
# kde contour plot using jointplot
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
sns.jointplot(x="x", y="y", data=df, kind="kde",shade=True);
plt.show()
```



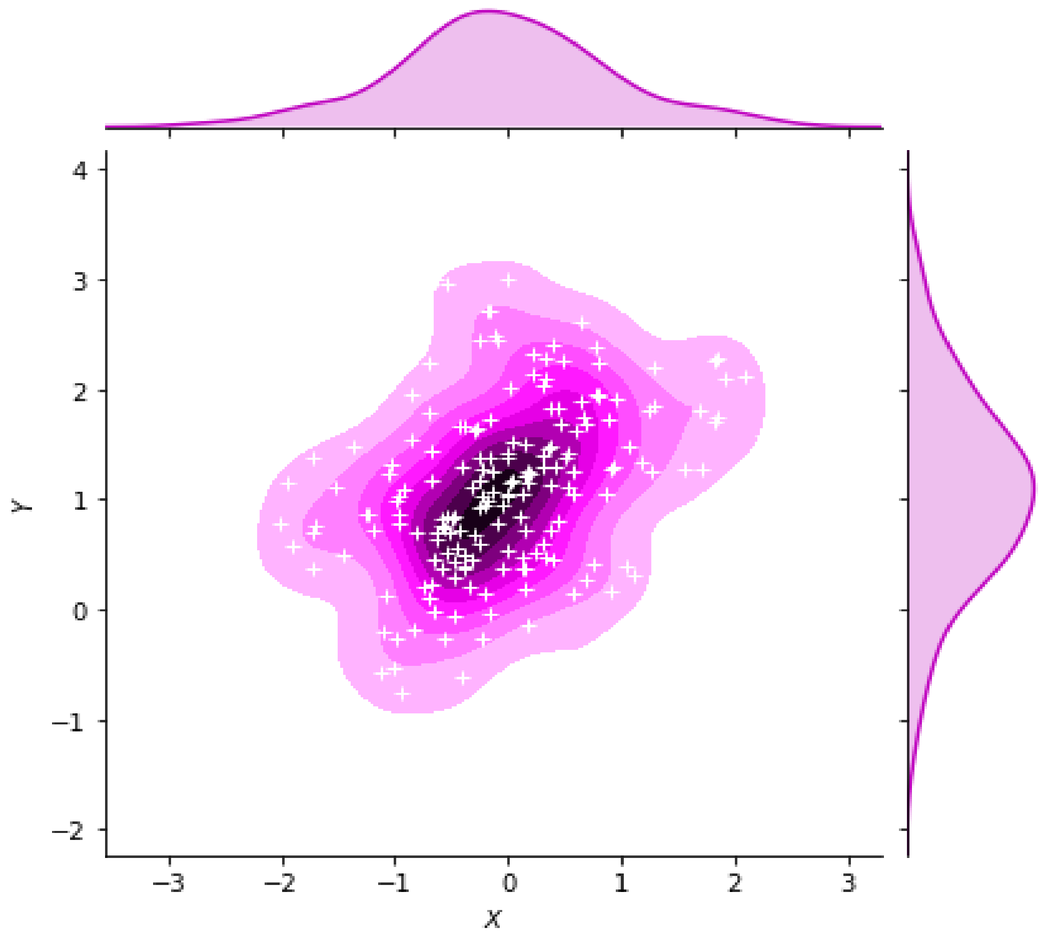
You can also draw a two-dimensional kernel density plot with the **kdeplot** function. This allows you to draw this kind of plot onto a specific (and possibly already existing) matplotlib axes, whereas the **jointplot** function manages its own figure. We also added horizontal and vertical rug plots.

```
#  plotting using kde plot
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
sns.kdeplot(x=df.x, y=df.y, shade=True)
sns.rugplot(df.x, color="g")
sns.rugplot(y=df.y);
plt.show()
```



The **jointplot** function uses a **JointGrid** to manage the figure. For more flexibility, you may want to draw your figure by using a **JointGrid** directly. The **jointplot** function returns a **JointGrid** object after plotting, which you can use to add more layers or to tweak other aspects of the visualization:

```
# using jointGrid
mean, cov = [0, 1], [(1, .5), (.5, 1)]
data = np.random.multivariate_normal(mean, cov, 200)
df = pd.DataFrame(data, columns=["x", "y"])
g = sns.jointplot(x="x", y="y", data=df, kind="kde", color="m", shade=True)
g.plot_joint(plt.scatter, c="w", s=30, linewidth=1, marker="+")
g.ax_joint.collections[0].set_alpha(0)
g.set_axis_labels("$X$", "$Y$");
plt.show()
```



Plotting amounts paid for groceries for the week

We make 2 data frames, one for week1 and the other for week 2 for the amount of groceries paid for each day of the week.

```
x=pd.DataFrame({'Day':[1,2,3,4,5,6,7],'Grocery':[30,80,45,23,51,46,76]})
y=pd.DataFrame({'Day':[8,9,10,11,12,13,14],'Grocery':[65,45,34,23,78,34,56]})
```

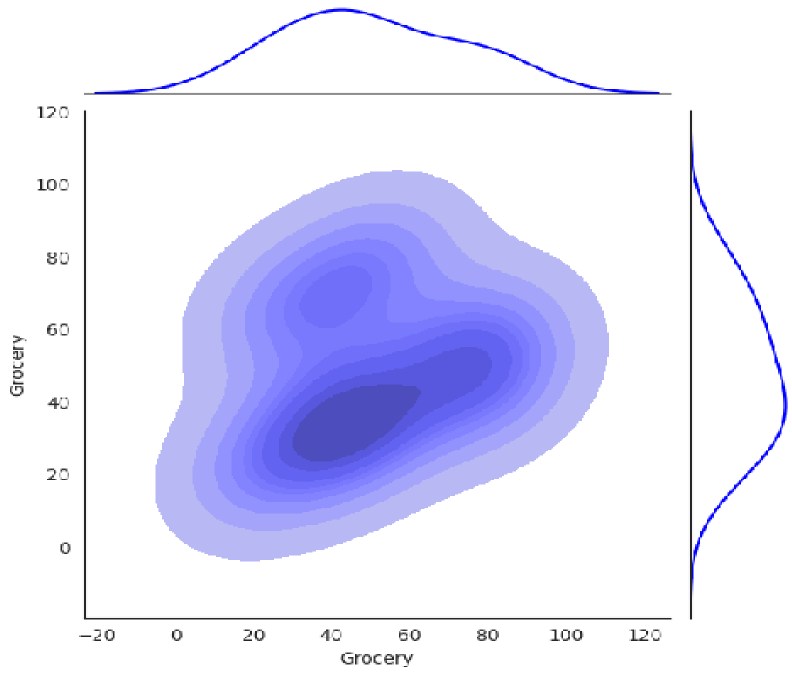
We first plot a contour map by setting kind to "kde" with a white background.

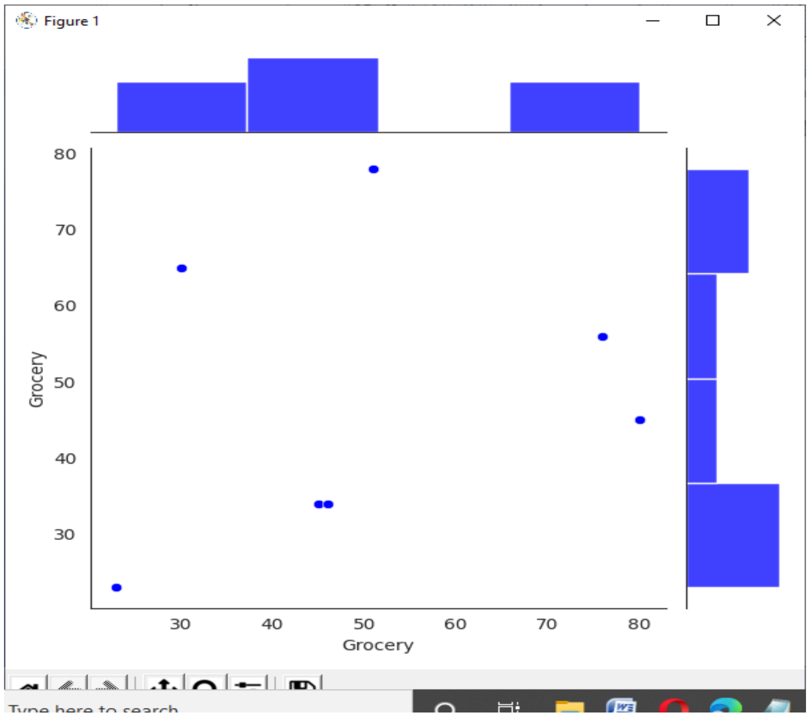
```
with sns.axes_style("white"):
   p=sns.jointplot(x=x['Grocery'], y=y['Grocery'], kind="kde", color="b",shade=True);
   p.fig.suptitle("Grocery Contour Plot")
plt.show()
```

We then plot a scatter plot with a white background.

```
with sns.axes_style("white"):
  p= sns.jointplot(x=x['Grocery'], y=y['Grocery'], color="b");
  p.fig.suptitle("Grocery Scatter Plot")
plt.show()
```

We have plot the titles with the **suptitle** function.





Pair Plot

Pair Plots are used to visualize relationships between each variable in your data. A pair plot produces a matrix of relationships between each variable in your data for an instant visualization of your data. A Pair Plot will take each numerical column of a data frame and put them on both the x and y axes and plot as a scatter plot where they meet. In cases where the same column variables meet, a histogram is drawn that shows the distribution of the variables. An optional regression line may be drawn on the scatter plot.

We will use the above Grocery data for our data. We will also add a third column for the percent of money spent each day.

```
data=pd.DataFrame(
 {'Day':['Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'],
       'Grocery':[30,80,45,23,51,46,76],
       'Percent':[.2,.4,.5,.3,.2,.3,.6]})
print(data)
```

```
         Day  Grocery  Percent
0     Monday       30      0.2
1    Tuesday       80      0.4
2  Wednesday       45      0.5
3   Thursday       23      0.3
4     Friday       51      0.2
5   Saturday       46      0.3
6     Sunday       76      0.6
```

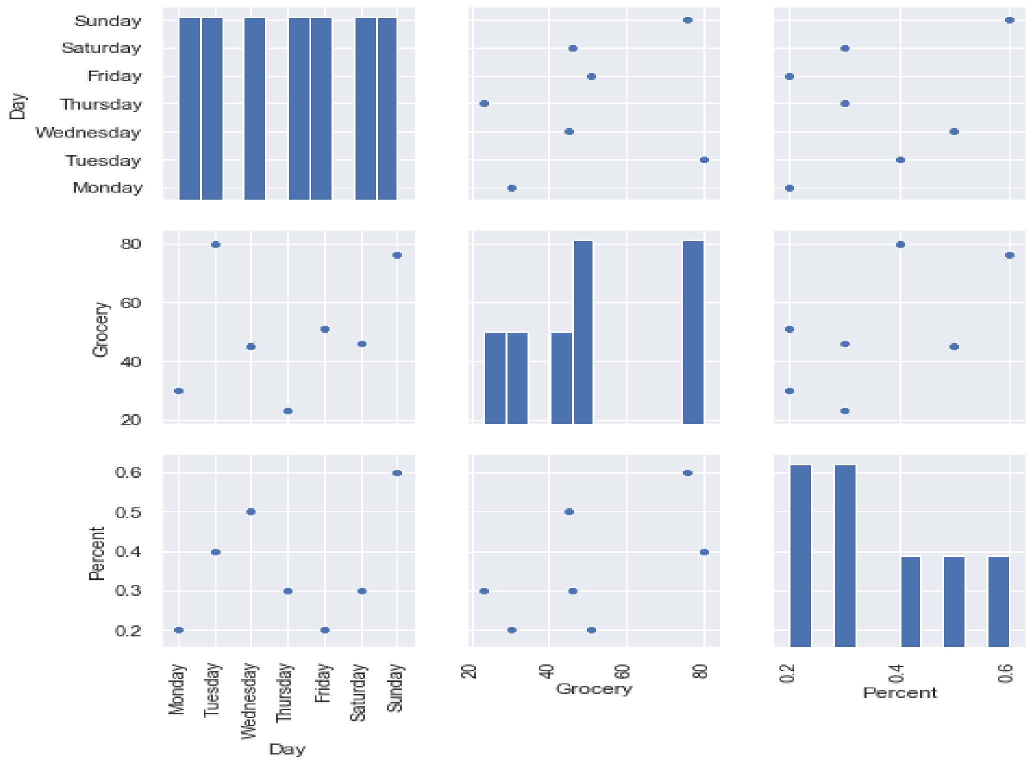
It is quite easy to make a pair plot. When we have string column data we must tell seaborn to include the columns using the string data using the **vars** parameter. The **vars** parameter is used to specify which columns you want to use in your pair plot. If you do not a specify a **vars** parameter the pair plot will plot all columns that only having numeric data.

```
# plot pair plot
g=sns.pairplot(data,vars=['Day','Grocery','Percent'])
```

We can specify to have the columns with numeric labels to have the labels rotated vertically rather than horizontally, using the following rotation code.

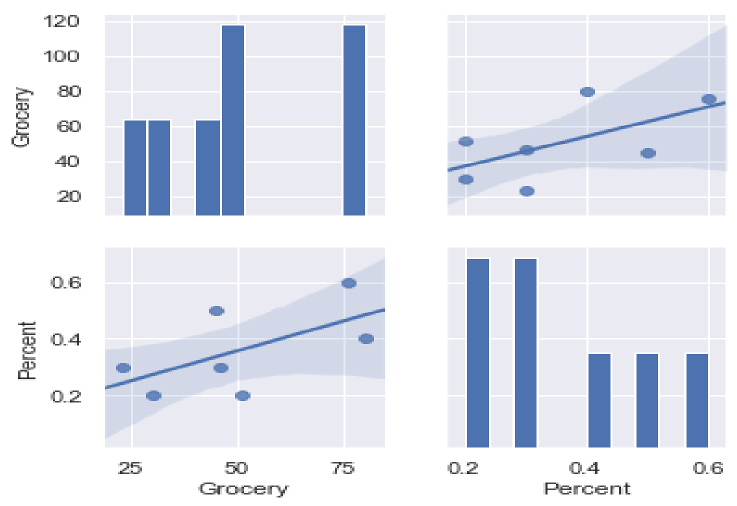
```
# rotate labels vertically rather than horizontally
import matplotlib.ticker as mticker
g.fig.draw(g.fig.canvas.get_renderer())
for ax in g.axes.flat:
    ticks_loc = ax.get_xticks()
    ax.xaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
   ax.set_xticklabels(ax.get_xticklabels(), rotation=90)

g.fig.suptitle("Weekly Groceries")
plt.show()
```

Our pair plot as follows :



We can also plot the regression line using the **kind** parameter. You cannot plot a regression line for non numeric column data. In this situation we do not need to specify the columns to plot or rotate the x axis labels.

```
#regression line pair plot
sns.pairplot(data,kind="reg")
plt.show()
```

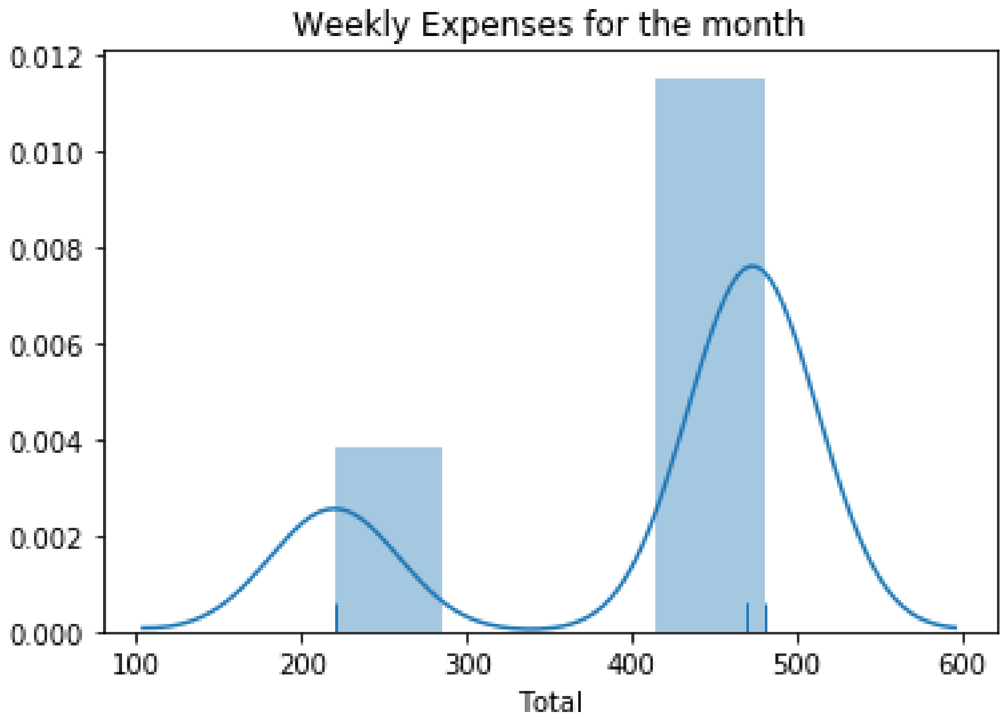


Seaborn Homework Question 1

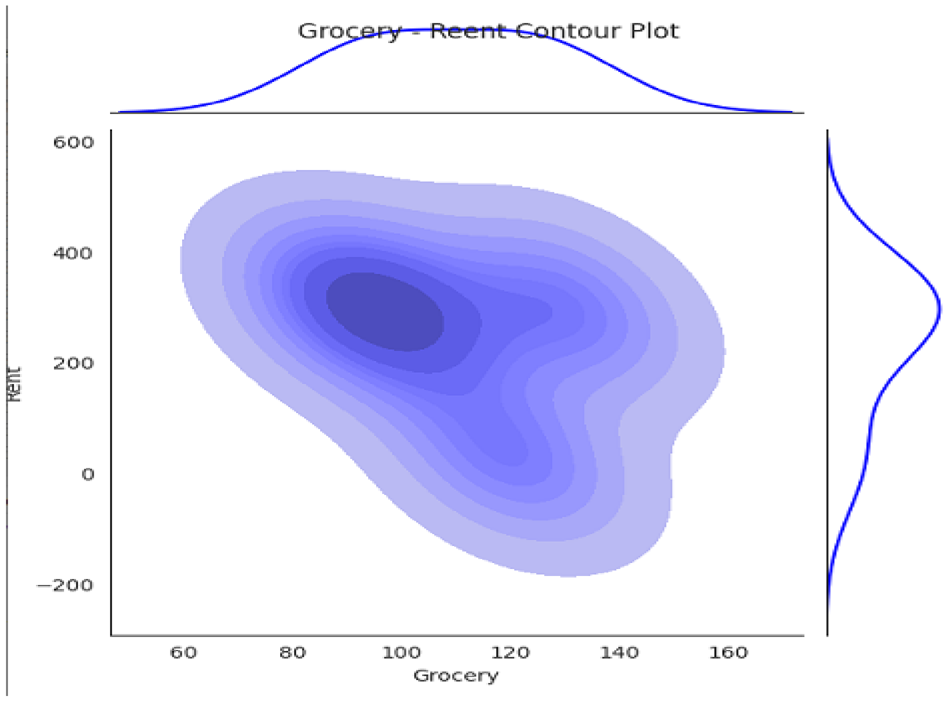
Make a data frame for your monthly expenses for the month. Use labels for the weeks rather than numbers: week1, week2, week 3 and week4. You may have columns for rent, food, entrainment, utilities etc. If you pay some things monthly then divide the monthly total by 4. Plot a distribution plot with a KDE of total weekly expenses. You would also need to make a "Total" column in your data frame. Print out the data frame. Choose two different expense columns and plot a joint plot. Next make a pair plot using the individual weekly expenses. Use labels for the weeks rather than numbers. Week1, week2, week 3 and week4, rotate the labels vertically. Put titles on your plots. Put everything in a python py file called seaborn_homework.py. You should get something like this:

```
    Week  Grocery  Rent  Utilities  Entertainment  Total
0  week1      100   300         50             20    470
1  week2      120    30         40             30    220
2  week3      130   300         30             20    480
3  week4       90   300         40             40    470
```

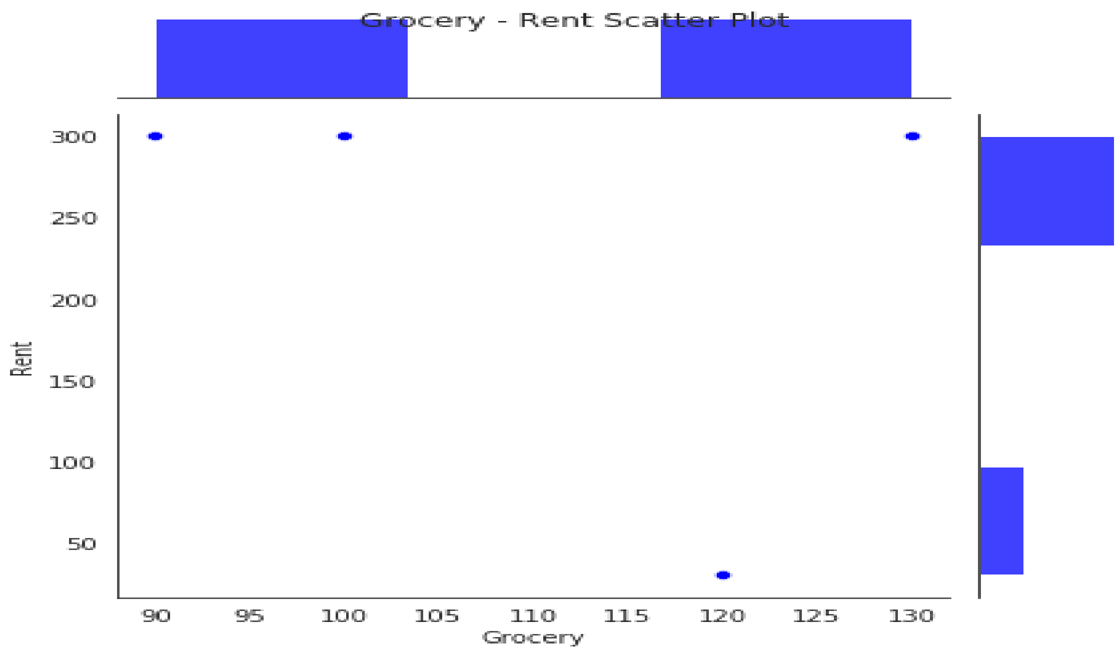
Distribution Plot:



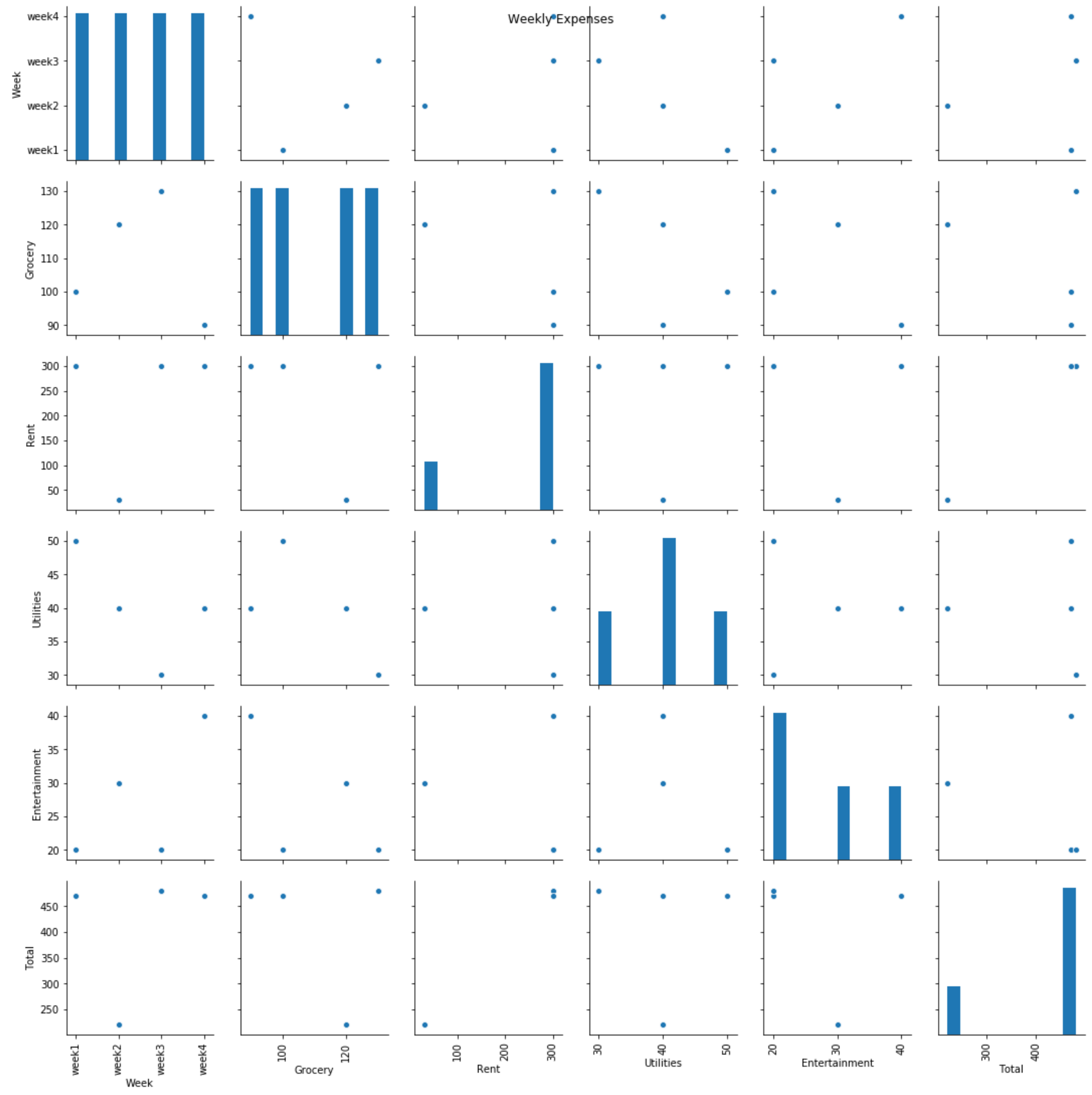
JointPlot:



Scatter Plot:



Pair Plot:



END