Week 4 Notes

04-01: Plotting with Pandas

Pandas uses matplotlib under the hood, and provides some convenient functions for visualising data. Before we dive into visualisation in pandas, let's take a look at the matplotlib's style package. Matplotlib comes with a number of predefined styles, which we can choose from to change the default look of our plots.

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

%matplotlib notebook
```

See Styles with Pandas

In [2]:
```
# see the pre-defined styles provided.
plt.style.available
```

Out[2]:
```
['Solarize_Light2',
 '_classic_test_patch',
 'bmh',
 'classic',
 'dark_background',
 'fast',
 'fivethirtyeight',
 'ggplot',
 'grayscale',
 'seaborn',
 'seaborn-bright',
 'seaborn-colorblind',
 'seaborn-dark',
 'seaborn-dark-palette',
 'seaborn-darkgrid',
 'seaborn-deep',
 'seaborn-muted',
 'seaborn-notebook',
 'seaborn-paper',
 'seaborn-pastel',
 'seaborn-poster',
 'seaborn-talk',
 'seaborn-ticks',
 'seaborn-white',
 'seaborn-whitegrid',
 'tableau-colorblind10']
```

We'll use the `'seaborn-colorblind'` style so that our plots use a color palette that is more color vision deficiency friendly.

In [3]:
```
# use the 'seaborn-colorblind' style
plt.style.use('seaborn-colorblind')
```

Visualising Panda Series & Dataframes using `df.plot()`

Now, where the built-in visualisation of pandas really shines is in helping with fast and easy plotting of series and dataframes that can help us explore the data. We can call the `plot()` method on series and dataframes and this will call `matplotlib.pyplot.plot()` under the hood, and weget a line graph of all the columns in the dataframe with labels.

In [4]:
```
np.random.seed(123)

df = pd.DataFrame({'A': np.random.randn(365).cumsum(0),
                   'B': np.random.randn(365).cumsum(0) + 20,
                   'C': np.random.randn(365).cumsum(0) - 20},
                  index=pd.date_range('1/1/2017', periods=365))
df.head()
```

Out[4]:

	A	B	C
2017-01-01	-1.085631	20.059291	-20.230904
2017-01-02	-0.088285	21.803332	-16.659325
2017-01-03	0.194693	20.835588	-17.055481
2017-01-04	-1.311601	21.255156	-17.093802
2017-01-05	-1.890202	21.462083	-19.518638

In [5]:
```
df.plot(); # add a semi-colon to the end of the plotting call to suppress unw
```

Showing Different Kinds of plots using `df.plot(... kind = '')`

We can select which plot we want to use by passing it into the 'kind' parameter. You can also choose the plot kind by using the `DataFrame.plot.kind` methods instead of providing the `kind` keyword argument.

`kind` :

- `'line'` : line plot (default)
- `'bar'` : vertical bar plot
- `'barh'` : horizontal bar plot
- `'hist'` : histogram
- `'box'` : boxplot
- `'kde'` : Kernel Density Estimation plot
- `'density'` : same as 'kde'
- `'area'` : area plot
- `'pie'` : pie plot
- `'scatter'` : scatter plot
- `'hexbin'` : hexbin plot

Scatter plot

In [6]:
```
df.plot('A','B', kind = 'scatter');
```

More Complex Plots

This time, we want to make a scatterplot with points varying in color and size. We'll use df.plot.scatter, pass in columns A and C. We set the color `c` and size `s` to change

based on the value of column B. Finally, we can choose the color palette used by passing a string into the parameter `colormap`.

In [7]:
```
# create a scatter plot of columns 'A' and 'C', with changing color (c) and s
df.plot.scatter('A', 'C', c='B', s=df['B'], colormap='viridis');
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
pandas/plotting/_matplotlib/tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```

Modifications to Plots

Here, we can see the A and C columns plotted against one another with size and color changing based on the values of the B column. Because `df.plot.scatter` returns a `matplotlib.axes._subplot`, we can perform modifications on this object just like objects returned by matplotlib plots.

In [8]:
```
ax = df.plot.scatter('A', 'C', c='B', s=df['B'], colormap='viridis')
ax.set_aspect('equal')
```

Other Possible Plots

Box plots

In [9]:
```
df.plot.box();
```

Histograms

In [10]:
```
df.plot.hist(alpha=0.7);
```

Kernel Density Estimation Plots

These visualise an estimate of a variables probability density function. Kernel density esitmation plots come in handy in data science applications, where you want to derive a **smooth continuous function** from a given sample.

You can also choose the plot kind by using the `DataFrame.plot.kind` methods instead of providing the `kind` keyword argument.

`kind` :

- `'line'` : line plot (default)
- `'bar'` : vertical bar plot
- `'barh'` : horizontal bar plot
- `'hist'` : histogram
- `'box'` : boxplot
- `'kde'` : Kernel Density Estimation plot
- `'density'` : same as 'kde'
- `'area'` : area plot
- `'pie'` : pie plot
- `'scatter'` : scatter plot
- `'hexbin'` : hexbin plot

In [11]:
```

```

```
df.plot.kde();
```

Visualising Multivariate Data using scatter matrix and parallel coordinates

Scatter Matrix using `pd.plotting.scatter_matrix()`

Pandas has plotting tools that help with visualising large amounts of data or high dimensional data.

The iris data set is a classic multivariate data set, which includes the sepal length, sepal width, petal length, and petal width, for hundreds of samples of three species of the iris flower. Pandas has a plotting tool that allows us to create a **scatter matrix** from a Dataframe.

A scatter matrix is a way of comparing each column in a DataFrame to every other column in a pairwise fashion.

In [12]:
```
iris = pd.read_csv('iris.csv')
iris.head()
```

Out[12]:

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa

	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
4	5.0	3.6	1.4	0.2	Iris-setosa

In [13]:
```
pd.plotting.scatter_matrix(iris, alpha = 0.7); #Refuses to work
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
pandas/plotting/_matplotlib/tools.py:400: MatplotlibDeprecationWarning:
The is_first_col function was deprecated in Matplotlib 3.4 and will be removed
two minor releases later. Use ax.get_subplotspec().is_first_col() instead.
  if ax.is_first_col():
```

Parallel Coordinates using `pd.plotting.parallel_coordinates`

Parallel coordinat plots are a common way of visualising high dimensional multivariate data. Each variable in the data set corresponds to an equally spaced parallel vertical line. The values of each variable are then connected by lines between for each individual observation. Coloring the lines according to species of flower allows the viewer to more easily see any patterns or clustering.

In [14]:
```
plt.figure()
pd.plotting.parallel_coordinates(iris, 'Name'); # Refuses to work.
```

04-02: Seaborn

Seaborn is a Python visualisation library based on matplotlib. Seaborn is really just a **wraparound** matplotlib. It adds styles to make default visualisations much more visually appealing and makes **creation of specific types of complicated plots** much simpler.

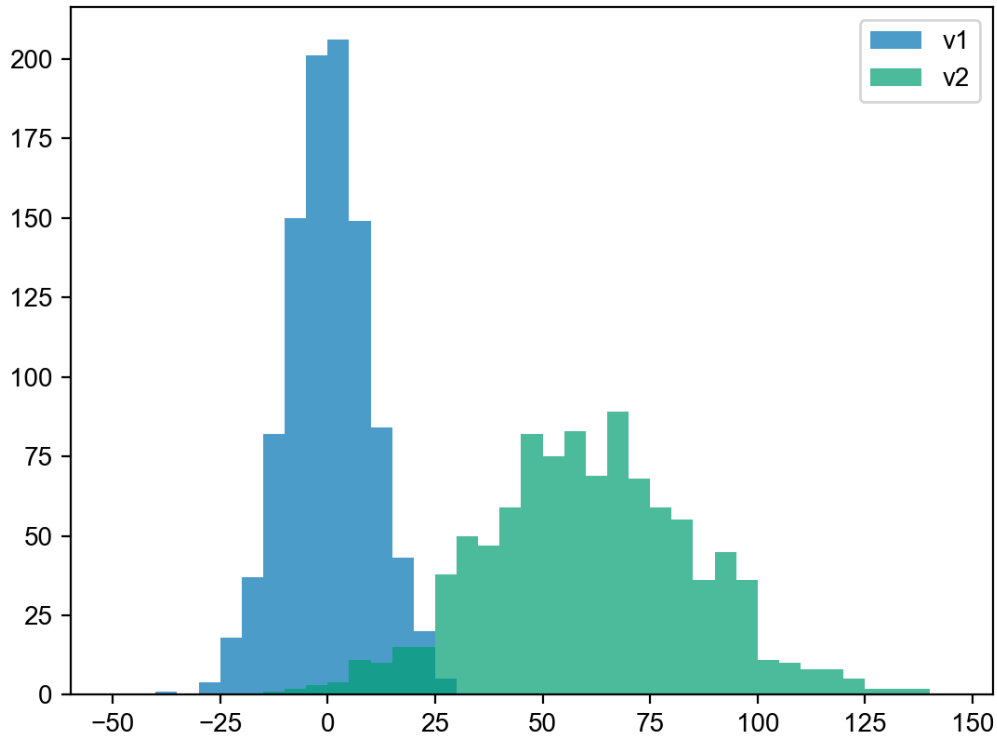
In [15]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib notebook
```

In [16]:
```
np.random.seed(1234)

v1 = pd.Series(np.random.normal(0,10,1000), name='v1')
v2 = pd.Series(2*v1 + np.random.normal(60,15,1000), name='v2')
```

In [17]:
```
plt.figure()
plt.hist(v1, alpha=0.7, bins=np.arange(-50,150,5), label='v1');
plt.hist(v2, alpha=0.7, bins=np.arange(-50,150,5), label='v2');
plt.legend();
```

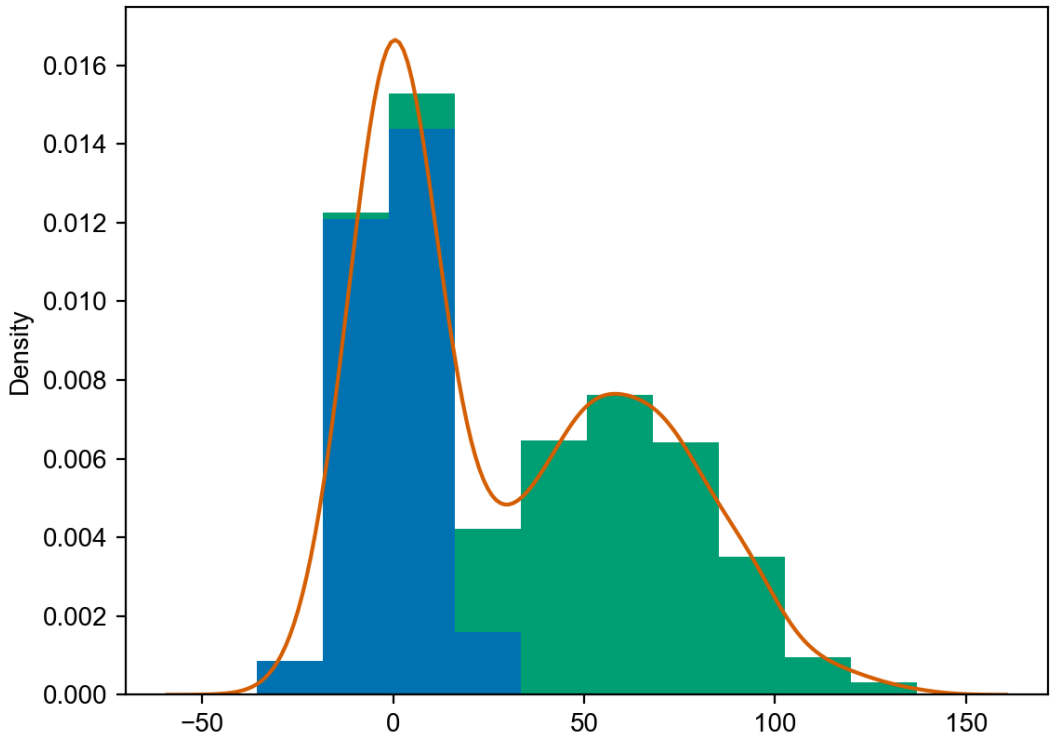


Plotting a KDE using `sns.kdeplot()`

We'll use `v3`, which is the concatenation of v1 and v2 to plot the kernel density function of the variable v3.

In [18]:

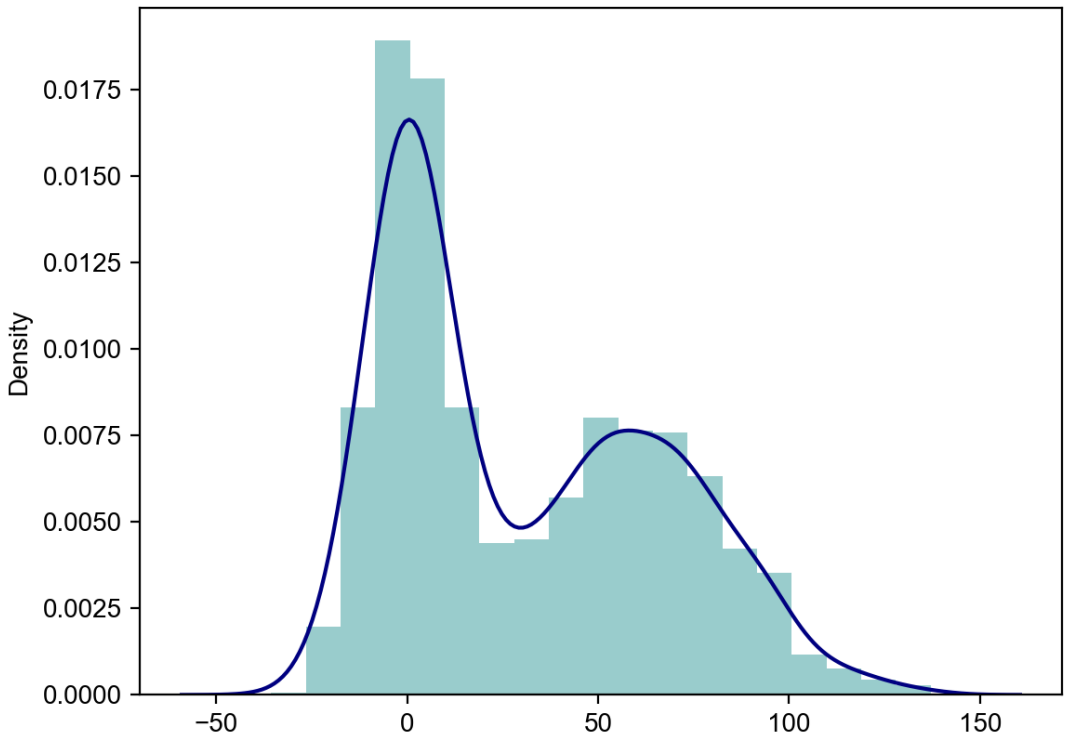
```
# plot a kernel density estimation over a stacked barchart
plt.figure()
plt.hist([v1, v2], histtype='barstacked', density=True);
v3 = np.concatenate((v1,v2))
sns.kdeplot(v3);
```



Using sns.displot()

In [19]:

```
plt.figure()
# we can pass keyword arguments for each individual component of the plot
sns.distplot(v3, hist_kws={'color': 'Teal'}, kde_kws={'color': 'Navy'});
```



```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
seaborn/distributions.py:2557: FutureWarning: `distplot` is a deprecated funct
ion and will be removed in a future version. Please adapt your code to use eit
her `displot` (a figure-level function with similar flexibility) or `histplot`
(an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

More Complex plots: Joint Plots using `sns.jointplot()`

The jointplot creates a scatterplot along the histograms for each individual variable on each axis. You've actually seen jointplots in module 2 and created them manually yourself. To create a jointplot, we just type in `sns.jointplot` and pass in the two series, v1 and v2.

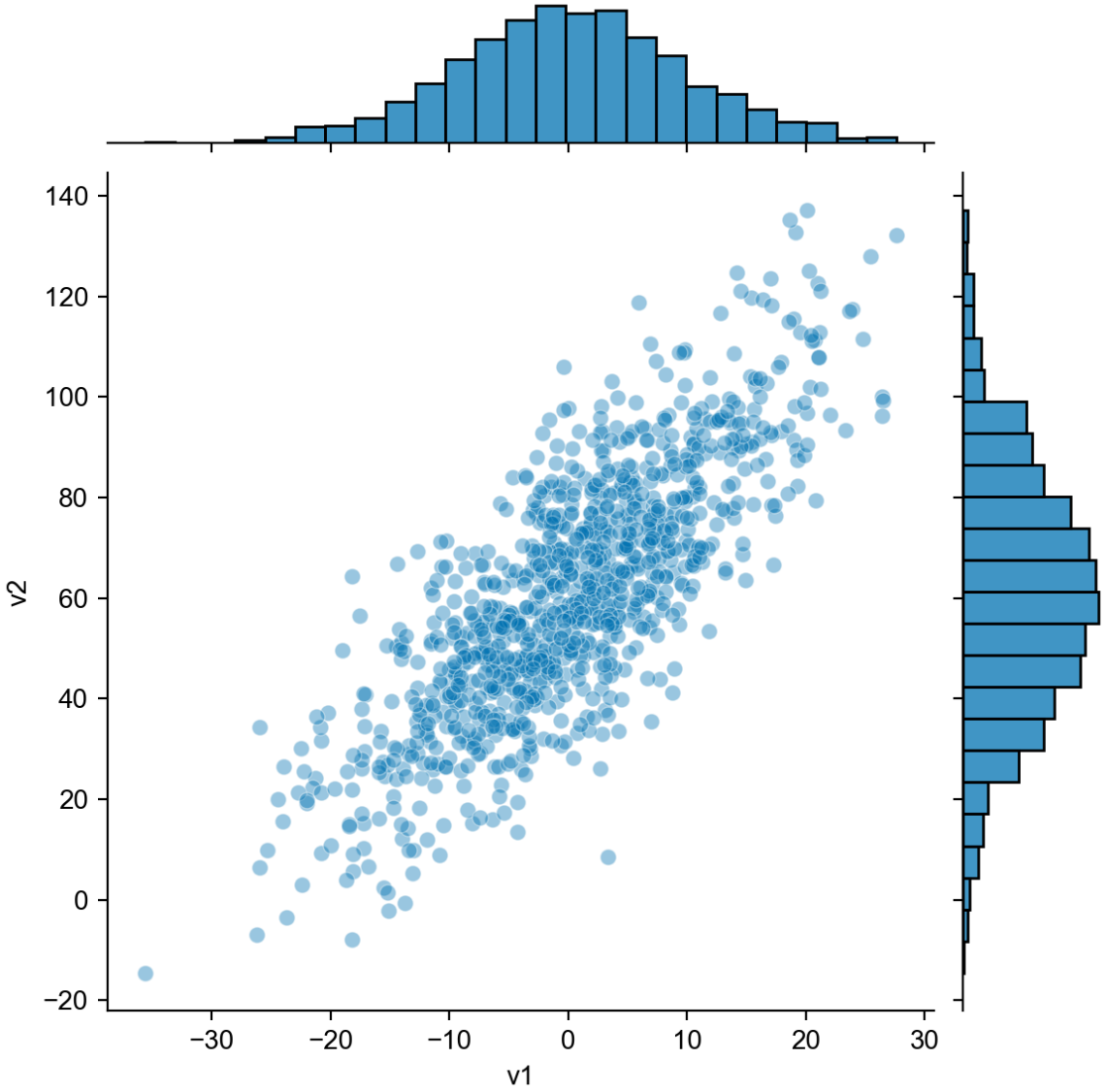
Using jointplot we can see that v1 and v2 appear to be normally distributed variables that are *positively correlated*. Because Seaborn uses matplotlib we can tweeak the plots using Matplotlib's tools.

Return Values

Some of the plotting functions in Seaborn return a `matplotlib.axes` object, while others operate on an entire figure and produce plots with several panels, returning a `Seaborn.grid` object.

In [20]:
```
sns.jointplot(v1, v2, alpha=0.4);
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyw
ord args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in
an error or misinterpretation.
  warnings.warn(
```



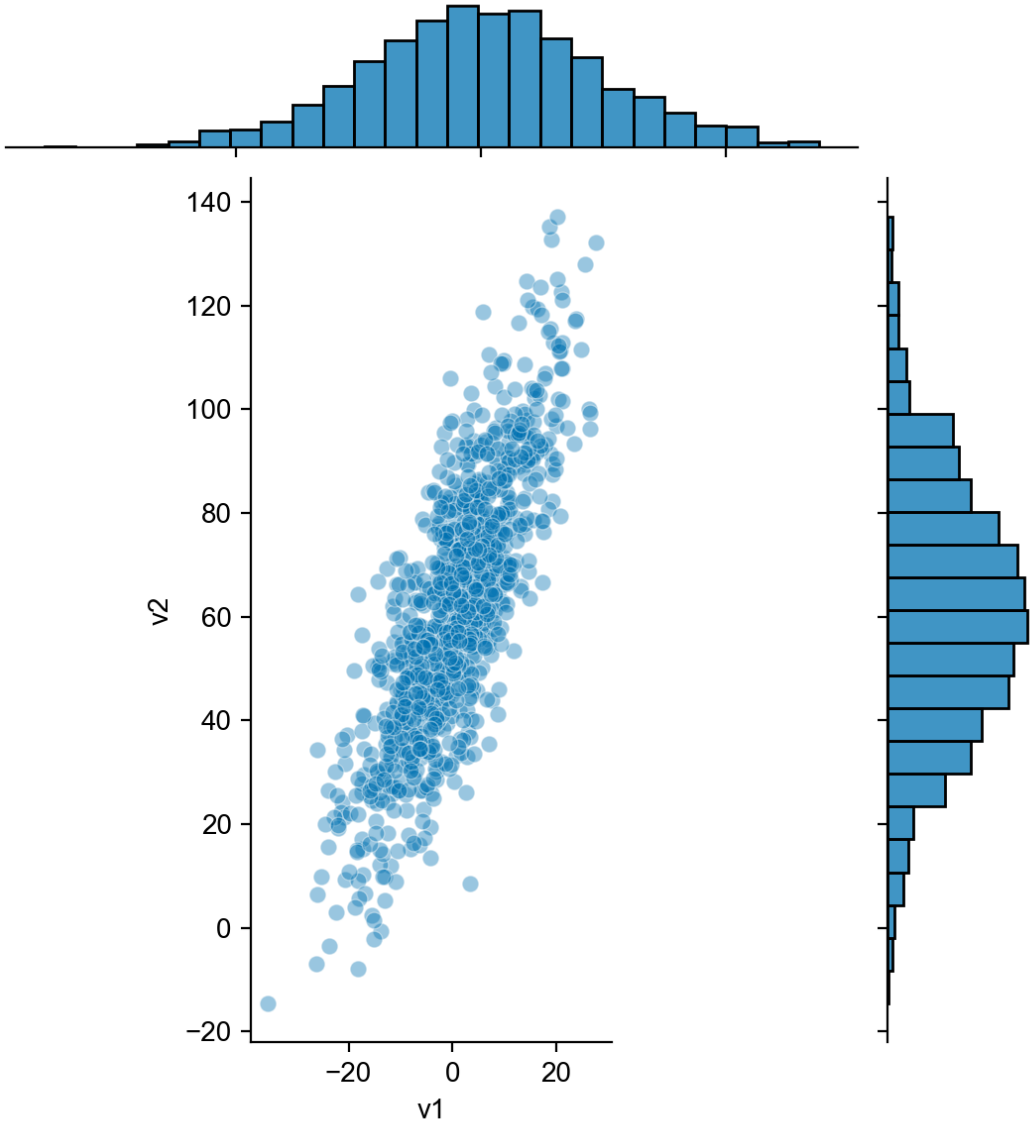
Modifications To Default

In both cases, matplotlib can be used to further tweak the plots. For example, sns.jointplot returns a `Seaborn.grid` object. We can plot a `matplotlib.axis.subplot` object using `grid.ax_joint`

In [21]:

```
grid = sns.jointplot(v1, v2, alpha=0.4);
grid.ax_joint.set_aspect('equal')
```

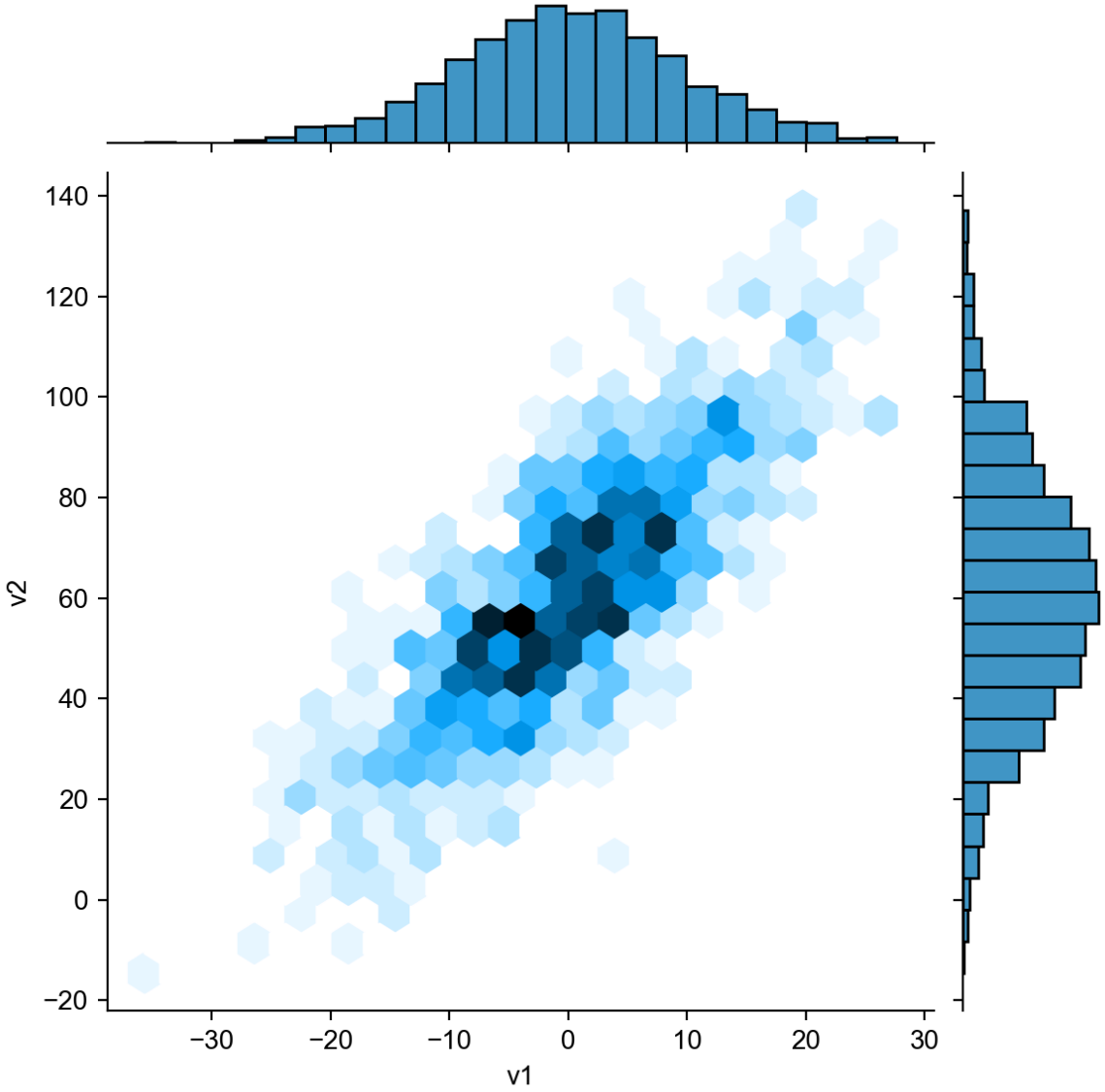
```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyw
ord args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in
an error or misinterpretation.
  warnings.warn(
```



Using a hexbin plot

In [22]:
```
sns.jointplot(v1, v2, kind='hex');
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyw
ord args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in
an error or misinterpretation.
  warnings.warn(
```



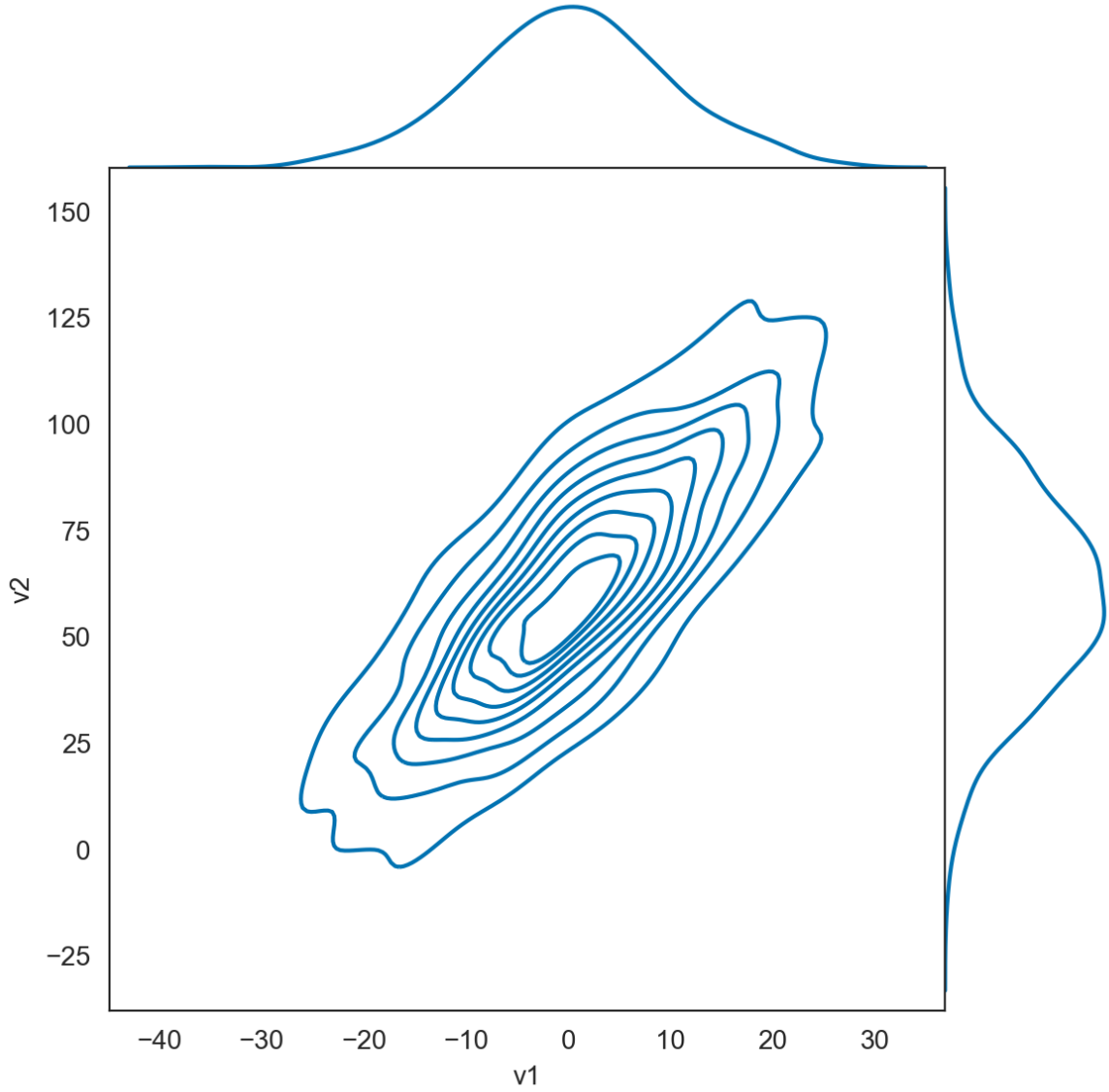
Using Kernal Density Estimation Plots in 2 Dimensions

In [23]:

```
# set the seaborn style for all the following plots
sns.set_style('white')

sns.jointplot(v1, v2, kind='kde', space=0);
```

```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
seaborn/_decorators.py:36: FutureWarning: Pass the following variables as keyw
ord args: x, y. From version 0.12, the only valid positional argument will be
`data`, and passing other arguments without an explicit keyword will result in
an error or misinterpretation.
  warnings.warn(
```



In [24]:

```
iris = pd.read_csv('iris.csv')
iris.head()
```

Out[24]:

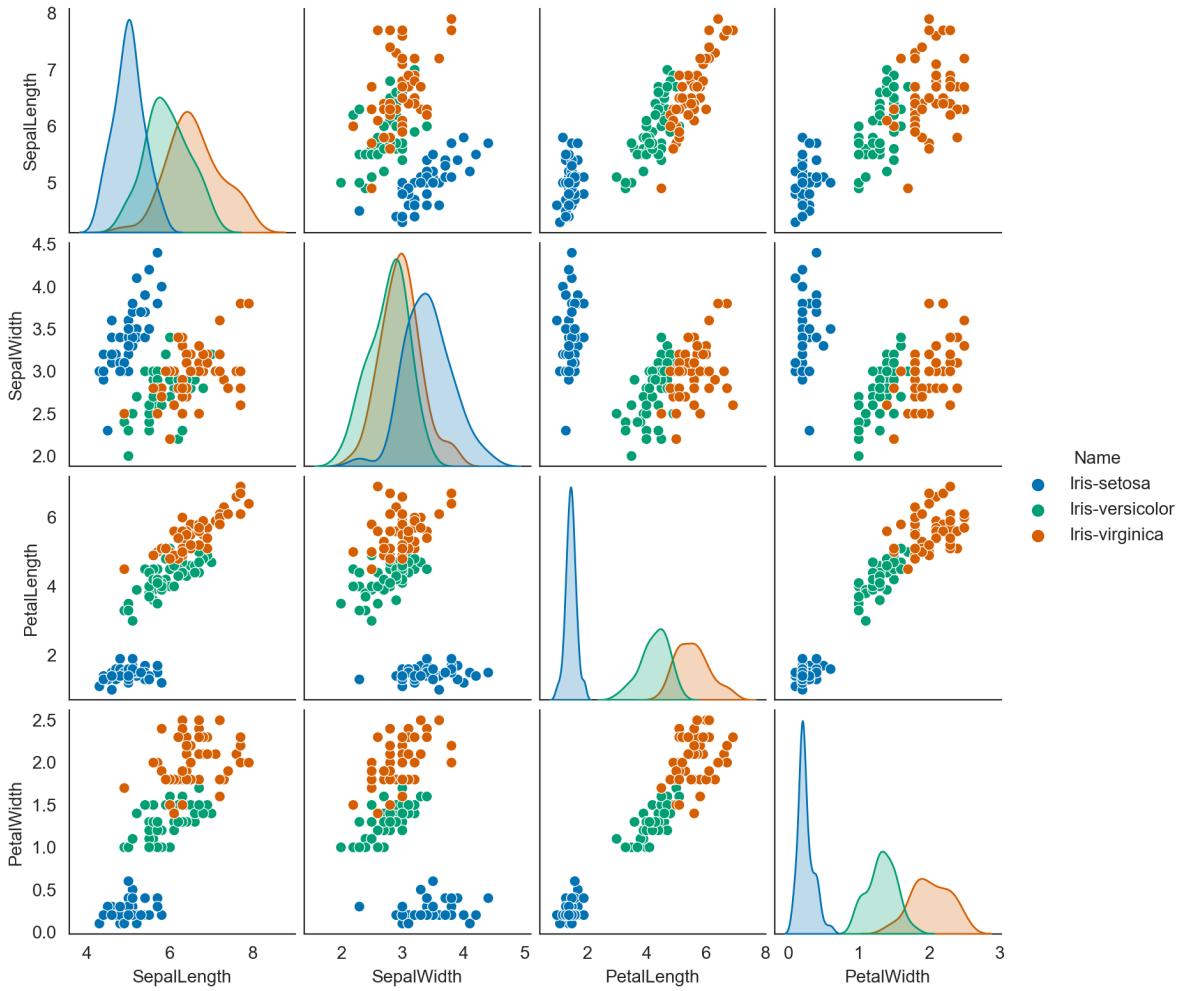
	SepalLength	SepalWidth	PetalLength	PetalWidth	Name
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Using pair plot

Using pair plot to look at your dataframe can be a very useful tool in **exploratory data analysis**. It looks like peddle length and peddle width are good options for separating the observation, whereas width is not a strong separator.

In [25]:

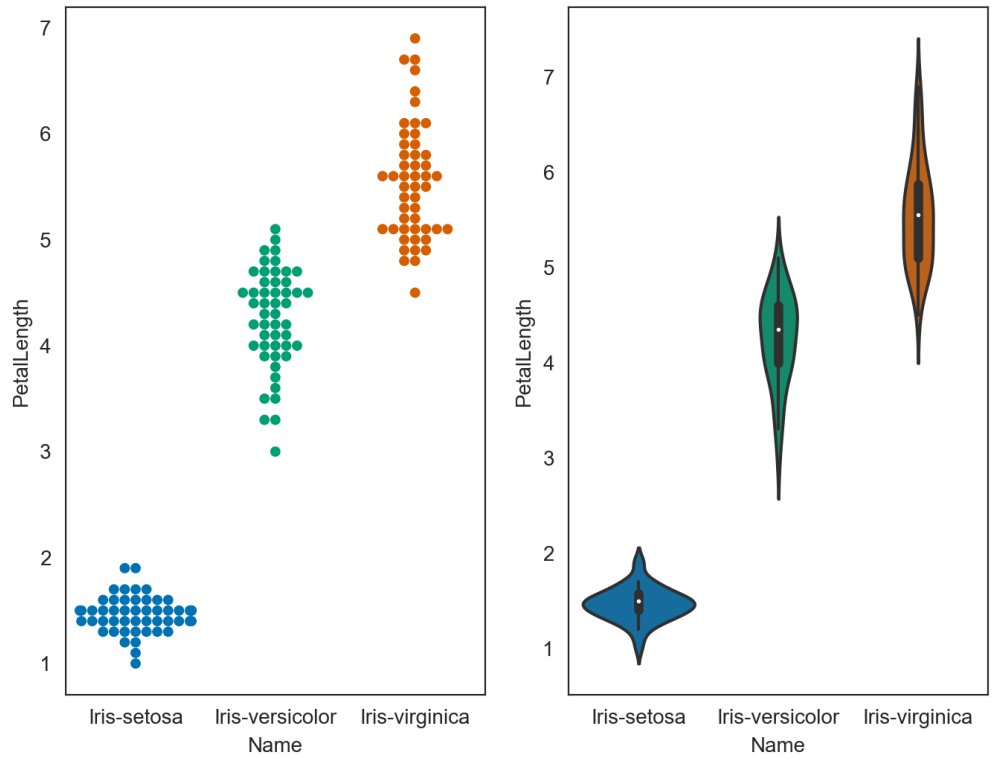
```
sns.pairplot(iris, hue='Name', diag_kind='kde', height=2);
```



Violin Plots

In [26]:

```
plt.figure(figsize=(8,6))
plt.subplot(121)
sns.swarmplot(x = 'Name', y = 'PetalLength', data=iris);
plt.subplot(122)
sns.violinplot(x = 'Name', y = 'PetalLength', data=iris);
```




```
/Library/Frameworks/Python.framework/Versions/3.8/lib/python3.8/site-packages/
seaborn/categorical.py:1296: UserWarning: 8.0% of the points cannot be placed;
you may want to decrease the size of the markers or use stripplot.
  warnings.warn(msg, UserWarning)
```

In [ ]: