Seaborn tutorial from geeksforgeeks.com

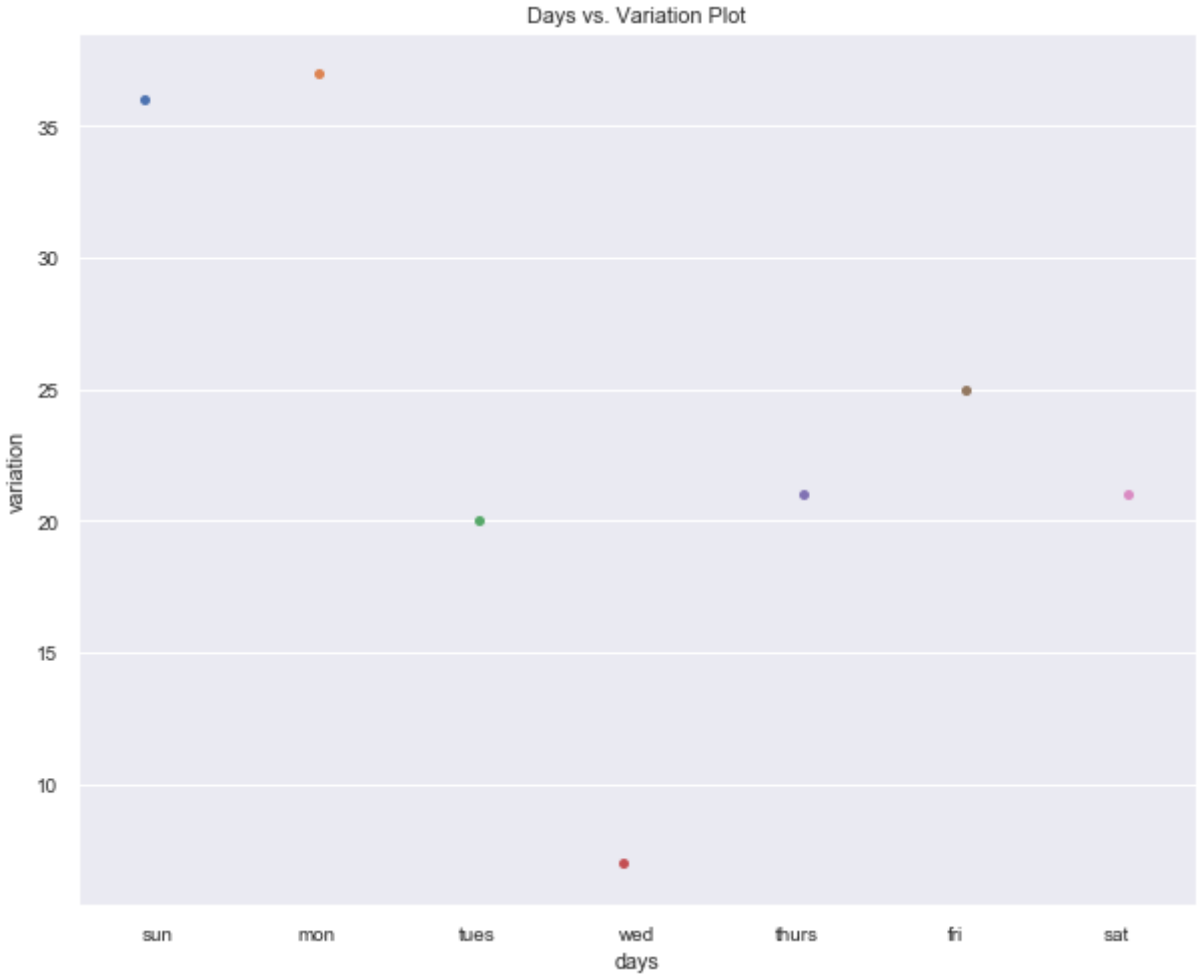
In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
import os
```

In [2]:
```
os.chdir(r'C:\Users\Admin\PycharmProjects\Sagun')
```

In [3]:
```
iris = pd.read_csv('iris.csv')
```

Sample Stripplot.

In [16]:
```
x = ['sun', 'mon','tues','wed','thurs','fri','sat']
y = np.random.randint(low = 1, high = 50, size = len(x))

plt.figure(figsize = (10,8))
sns.set(style = 'darkgrid', font_scale = 0.90)  #can use style = 'whitegrid' as well.

sns.stripplot(x,y)

plt.xlabel('days')
plt.ylabel('variation')
plt.title('Days vs. Variation Plot')

plt.show()
```

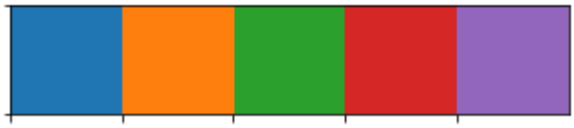


Color Palette.

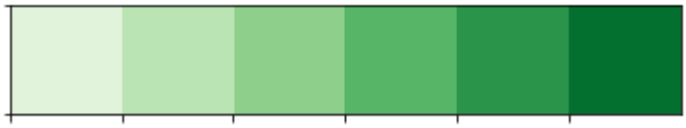
Types of Color palette:

- pastel
- muted
- bright
- deep
- colorblind
- dark
- cubehelix
- coolwarm
- husl.

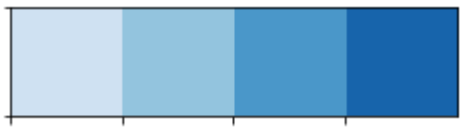
In [8]:
```
current_palette = sns.color_palette(n_colors = 5, desat = True)
sns.palplot(current_palette)
plt.show()
```



In [14]:
```
current_palette = sns.color_palette(palette = 'Greens')
sns.palplot( current_palette )
plt.show()
```



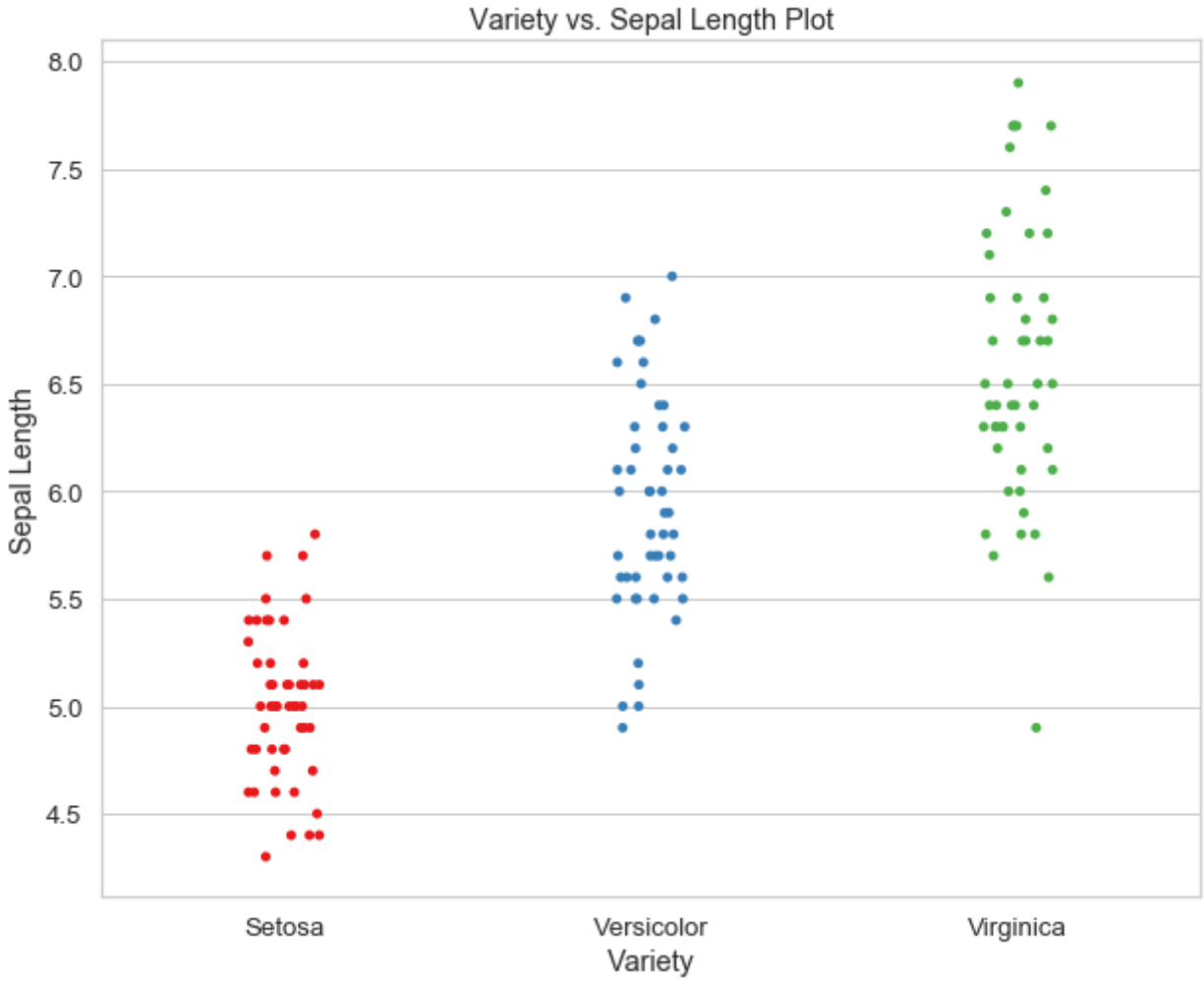
In [13]:
```
current_palette = sns.color_palette(palette = 'Blues', n_colors = 4)
sns.palplot( current_palette )
plt.show()
```



In [62]:
```
current_palette = sns.color_palette(palette = 'husl', n_colors = 6)
sns.palplot( current_palette )
plt.show()
```

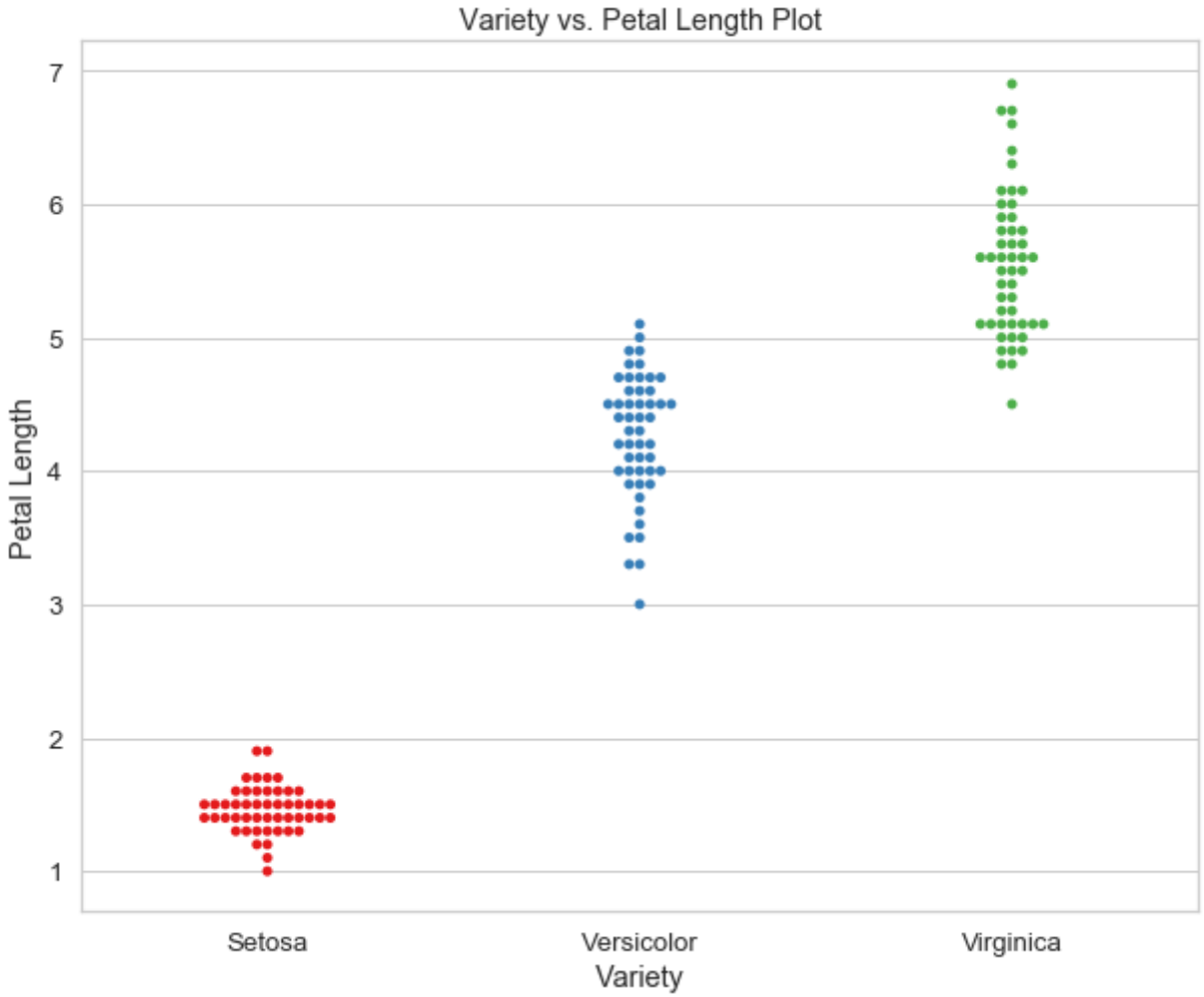
1. Stripplot.

In [27]:
```
plt.figure(figsize = (10,8))
sns.set(style = 'whitegrid', font_scale = 1.2)
sns.stripplot(x = iris['variety'], y = iris['sepal.length'], palette = 'Set1')

plt.xlabel('Variety')
plt.ylabel('Sepal Length')
plt.title('Variety vs. Sepal Length Plot')

plt.show()
```



2. Swarmplots.

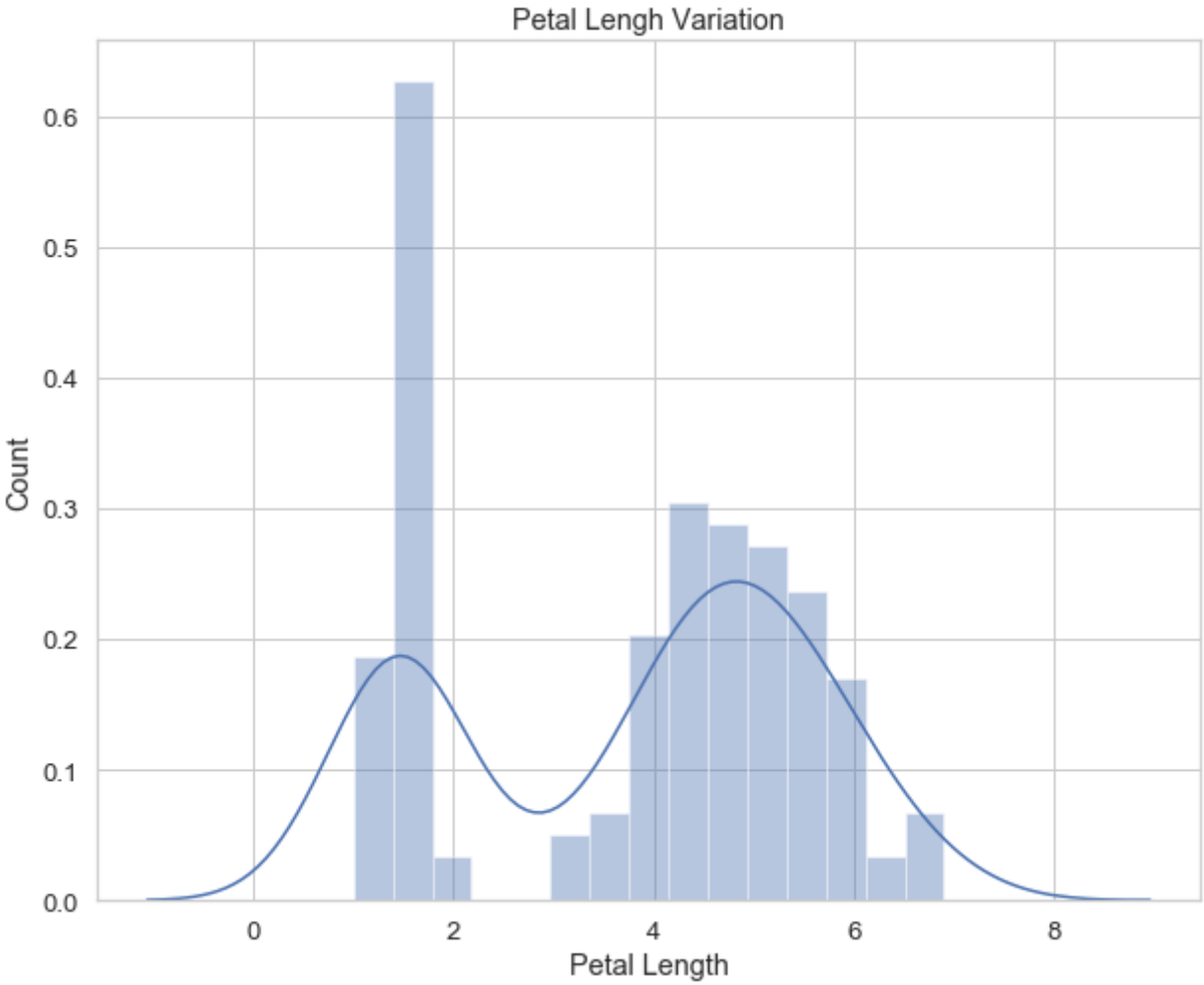
In [31]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

sns.swarmplot(x = iris['variety'], y = iris['petal.length'], palette = 'Set1', marker =
#Swarmplot doesn't allow overlapping of the markers unlike Stripplot.

plt.xlabel('Variety')
plt.ylabel('Petal Length')
plt.title('Variety vs. Petal Length Plot')

plt.show()
```



3. Distplot.

- It is used basically for univariant set of observations and visualizes it through a histogram i.e. only one observation and hence we choose one particular column of the dataset.
  - Syntax: distplot(a[, bins, hist, kde, rug, fit, ...])
- KDE stands for Kernel Density Estimation. It is a way to estimate the probability density function of a continuous random variable. It is used for non-parametric analysis.

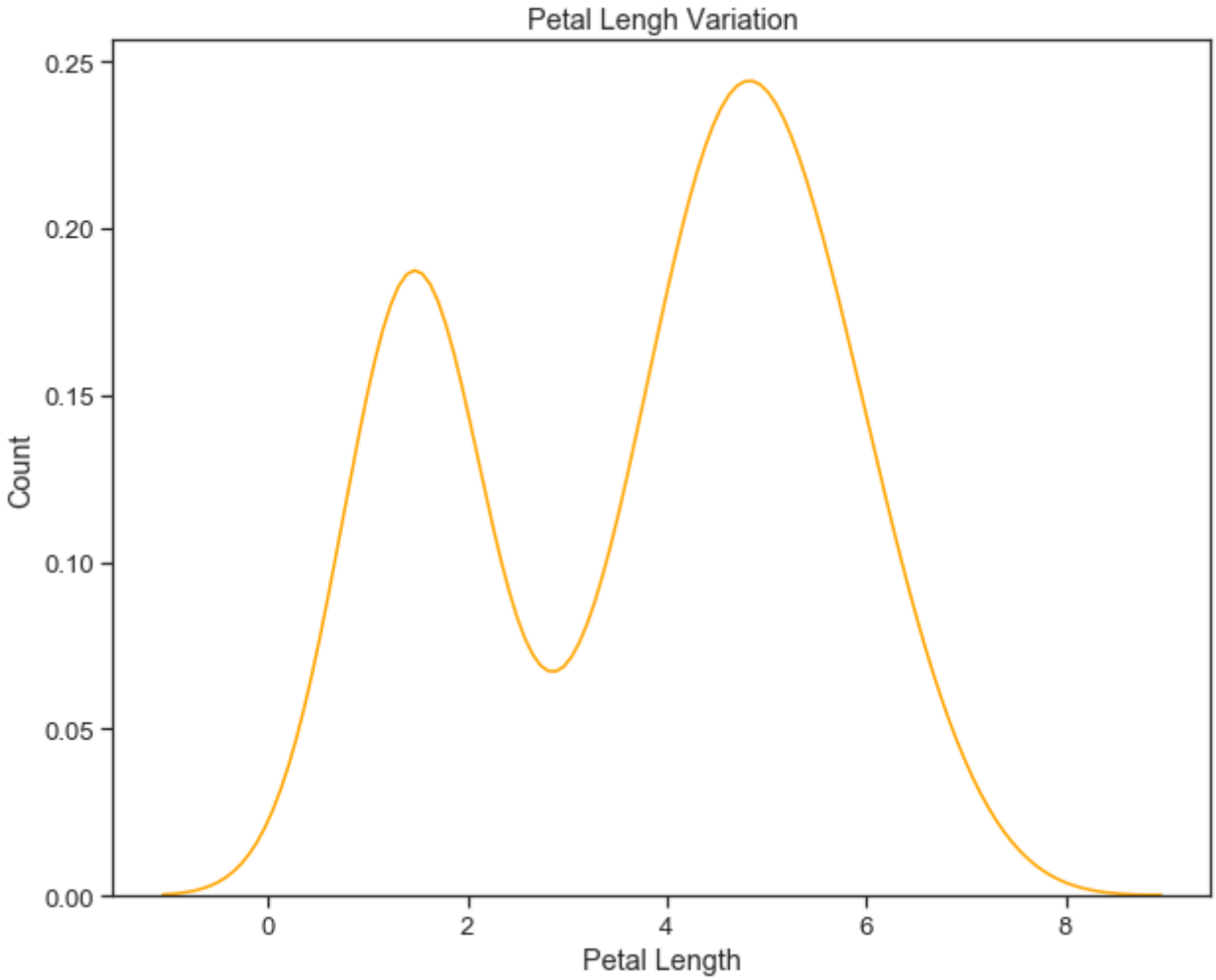
In [18]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

sns.distplot(a = iris['petal.length'], bins=15, color = 'b', vertical = False )      #kd

plt.xlabel('Petal Length')
plt.ylabel('Count')
plt.title('Petal Lengh Variation')

plt.show()
```



To plot only KDE without histogram bins.

- put hist = False

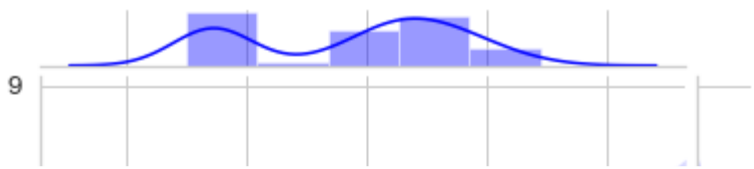
In [24]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'ticks', font_scale = 1.2)

sns.distplot(a = iris['petal.length'], hist = False, color = 'orange' )      #kde = True

plt.xlabel('Petal Length')
plt.ylabel('Count')
plt.title('Petal Lengh Variation')

plt.show()
```



4. Jointplot.

- It is used to draw a plot of two variables with bivariate and univariate graphs. It basically combines two different plots.
- Syntax: jointplot(x, y[, data, kind, stat_func, ...])

In [69]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

sns.jointplot(x = iris['petal.length'], y = iris['sepal.length'], kind = 'scatter', col
#'scatter' is default.

sns.jointplot(x = iris['petal.length'], y = iris['sepal.length'], kind = 'reg', color =
#‘reg’ plots a linear regression line.

sns.jointplot(x = iris['petal.length'], y = iris['sepal.length'], kind= 'resid', color
#‘resid’ plots the residual of the data to the regression line.

sns.jointplot(x = iris['petal.length'], y = iris['sepal.length'], kind = 'kde', color =
#kde plots a kernel density estimate in the margins and converts the interior into a sh

sns.jointplot(x = iris['petal.length'], y = iris['sepal.length'], kind = 'hex', color =
#‘hex’ bins the data into hexagons with histograms in the margins.

plt.show()
```

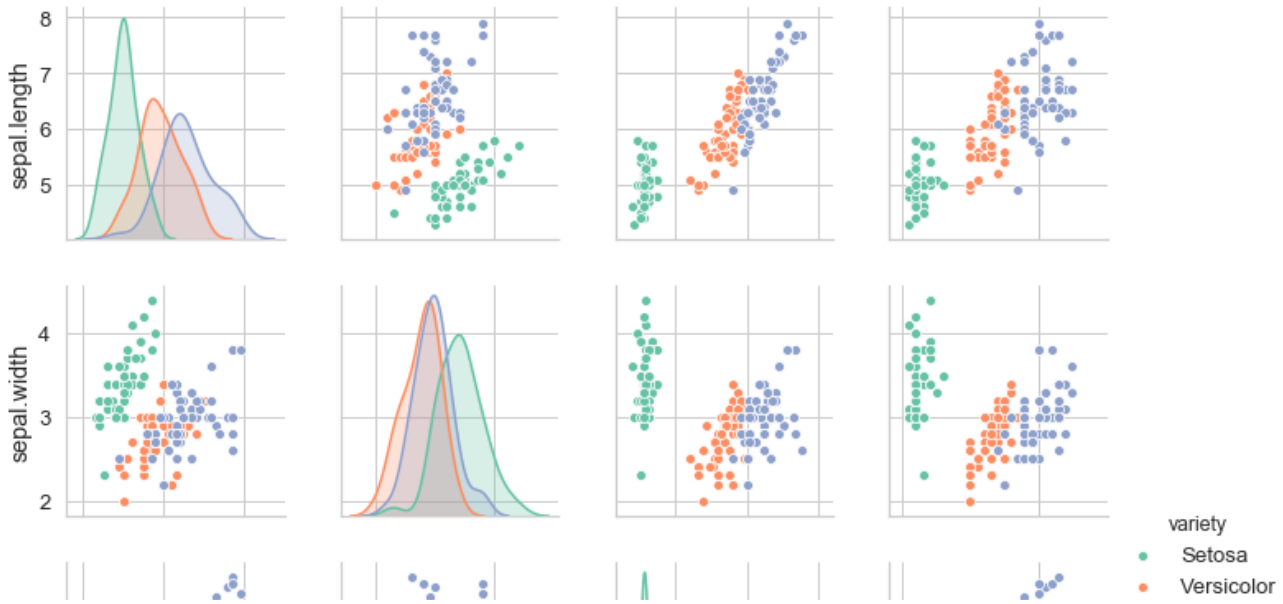


5. Pairplots.

- It represents pairwise relation across the entire dataframe and supports an additional argument called hue for categorical separation. What it does basically is create a jointplot between every possible numerical column.
- Syntax: pairplot(data[, hue, hue_order, palette, …])

In [70]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)


sns.pairplot(data=iris, hue='variety', palette='Set2')
plt.show()
```

```
<Figure size 720x576 with 0 Axes>
```

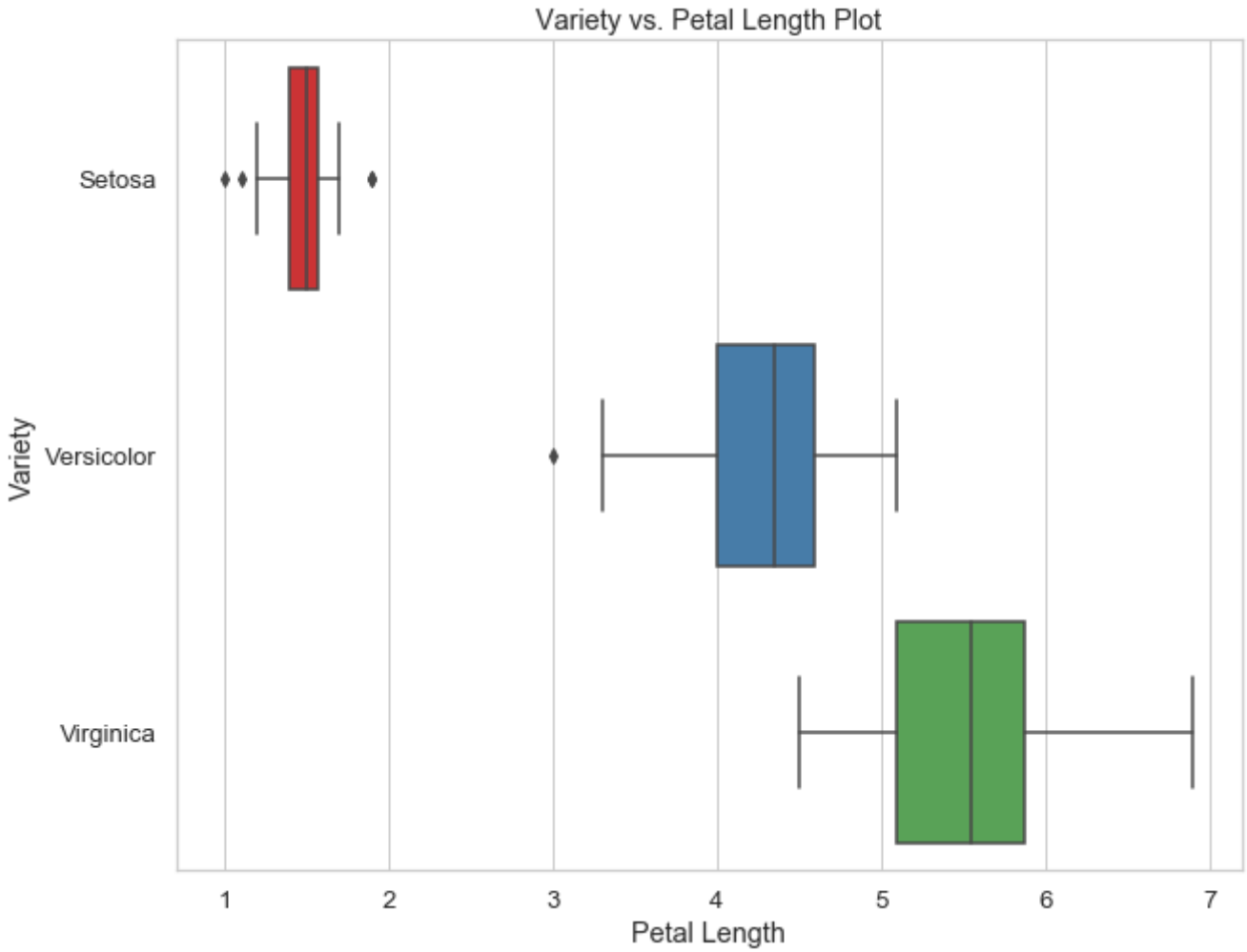


Seaborn Tutorial from Tutorialspoint.

6. Boxplots.

- Box plots usually have vertical lines extending from the boxes which are termed as whiskers which indicate variability outside the upper and lower quartiles.
- Any Outliers in the data are plotted as individual points.

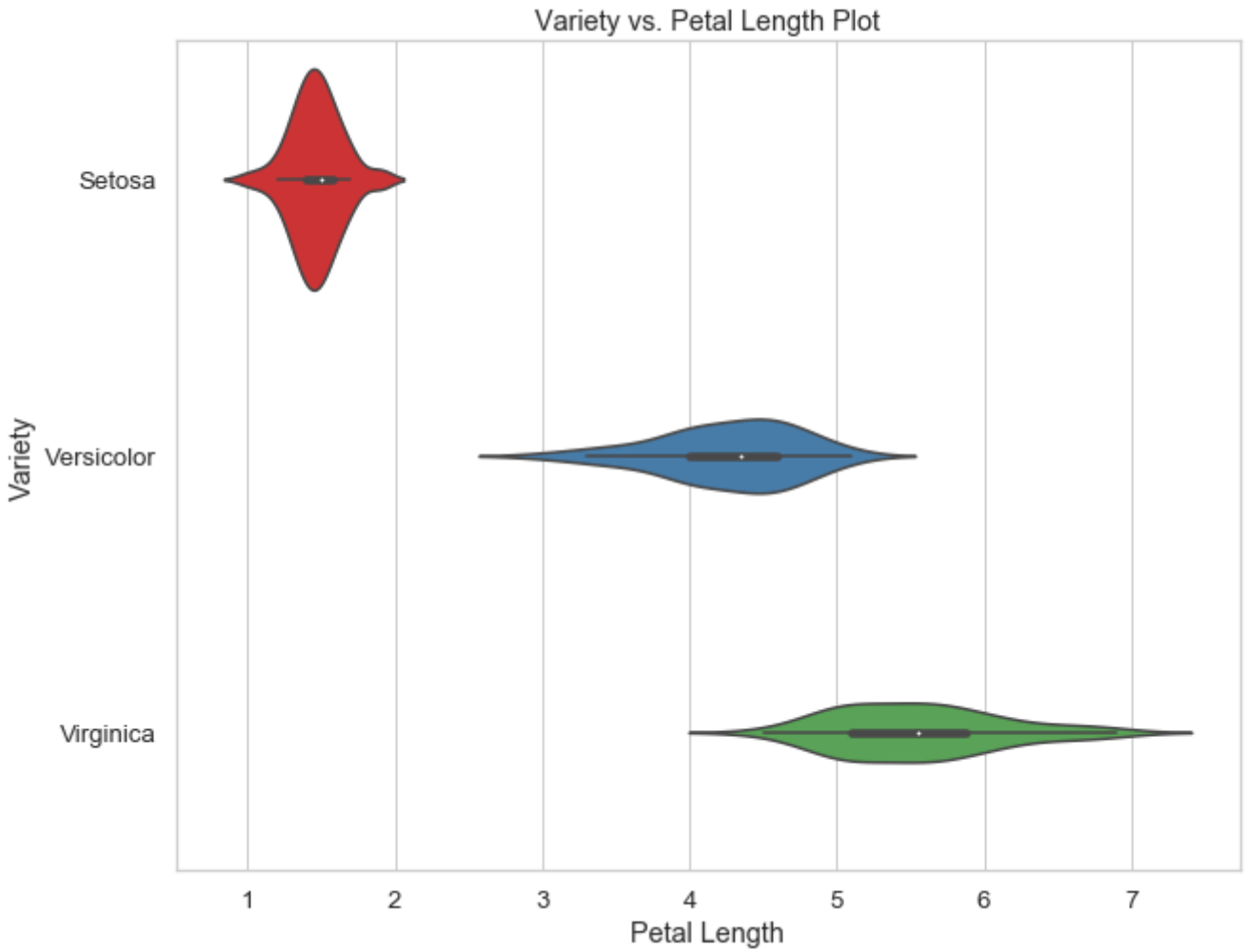
In [26]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

sns.boxplot(y = iris['variety'], x = iris['petal.length'], palette = 'Set1')
#For horizontal orientation, we can also use orient = 'h'.

plt.ylabel('Variety')
plt.xlabel('Petal Length')
plt.title('Variety vs. Petal Length Plot')

plt.show()
```



7. Violin Plots.

- Violin Plots are a combination of the box plot with the kernel density estimates.
- The quartile and whisker values from the boxplot are shown inside the violin.
- As the violin plot uses KDE, the wider portion of violin indicates the higher density and narrow region represents relatively lower density.
- The Inter-Quartile range in boxplot and higher density portion in kde fall in the same region of each category of violin plot.

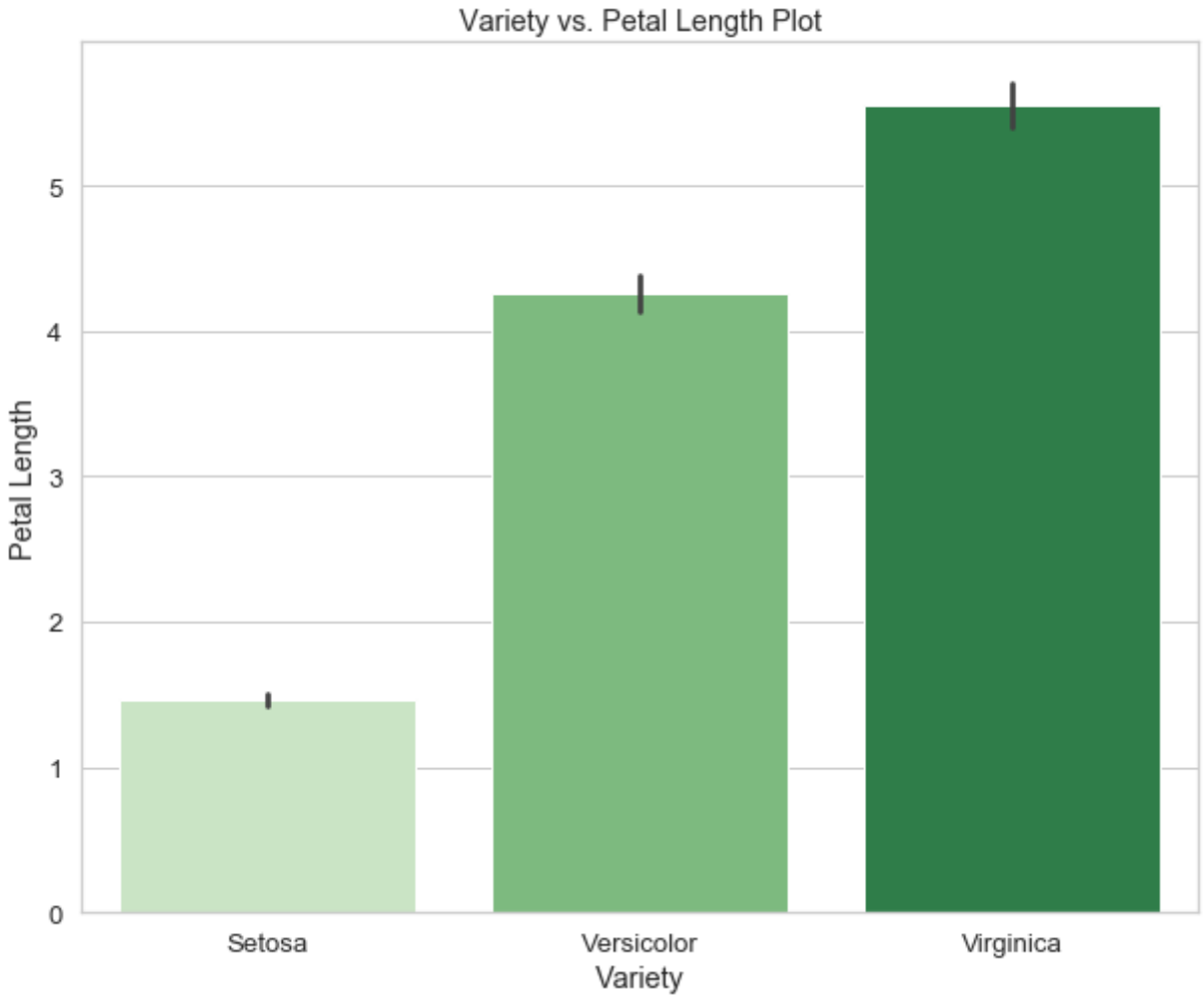
In [29]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

sns.violinplot(y = iris['variety'], x = iris['petal.length'], palette = 'Set1')

plt.ylabel('Variety')
plt.xlabel('Petal Length')
plt.title('Variety vs. Petal Length Plot')

plt.show()
```



8. Barplots.

- The barplot() shows the relation between a categorical variable and a continuous variable.
- Bar plot represents the estimate of central tendency.

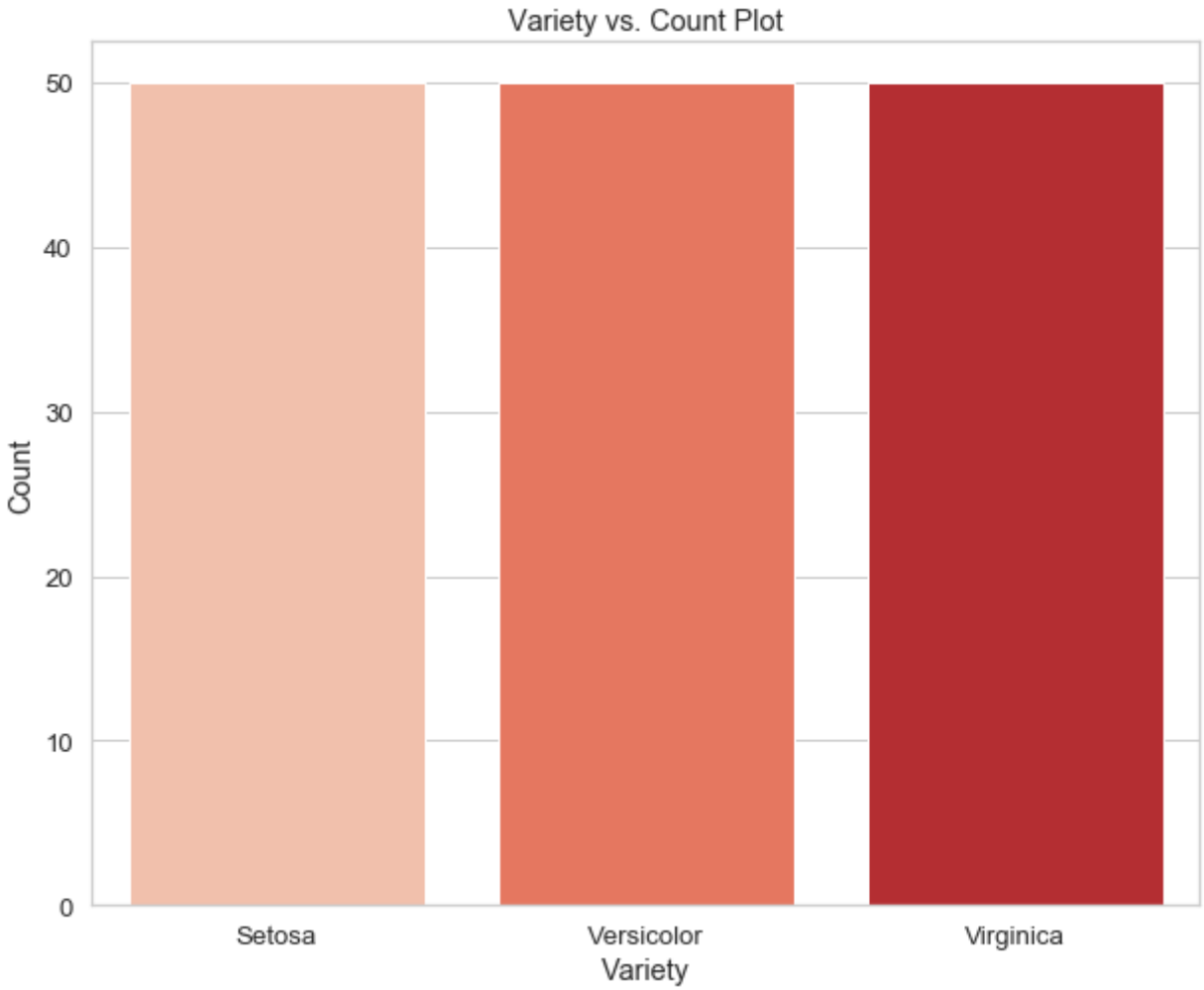
In [33]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

sns.barplot(x = iris['variety'], y = iris['petal.length'], palette = 'Greens')

plt.xlabel('Variety')
plt.ylabel('Petal Length')
plt.title('Variety vs. Petal Length Plot')

plt.show()
```



9. Countplot.

- A special case in barplot is to show the no of observations in each category rather than computing a statistic for a second variable. For this, we use countplot().

In [34]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

sns.countplot(x = iris['variety'], palette = 'Reds')      #Only one variable. 'y' value

plt.xlabel('Variety')
plt.ylabel('Count')
plt.title('Variety vs. Count Plot')

plt.show()
```



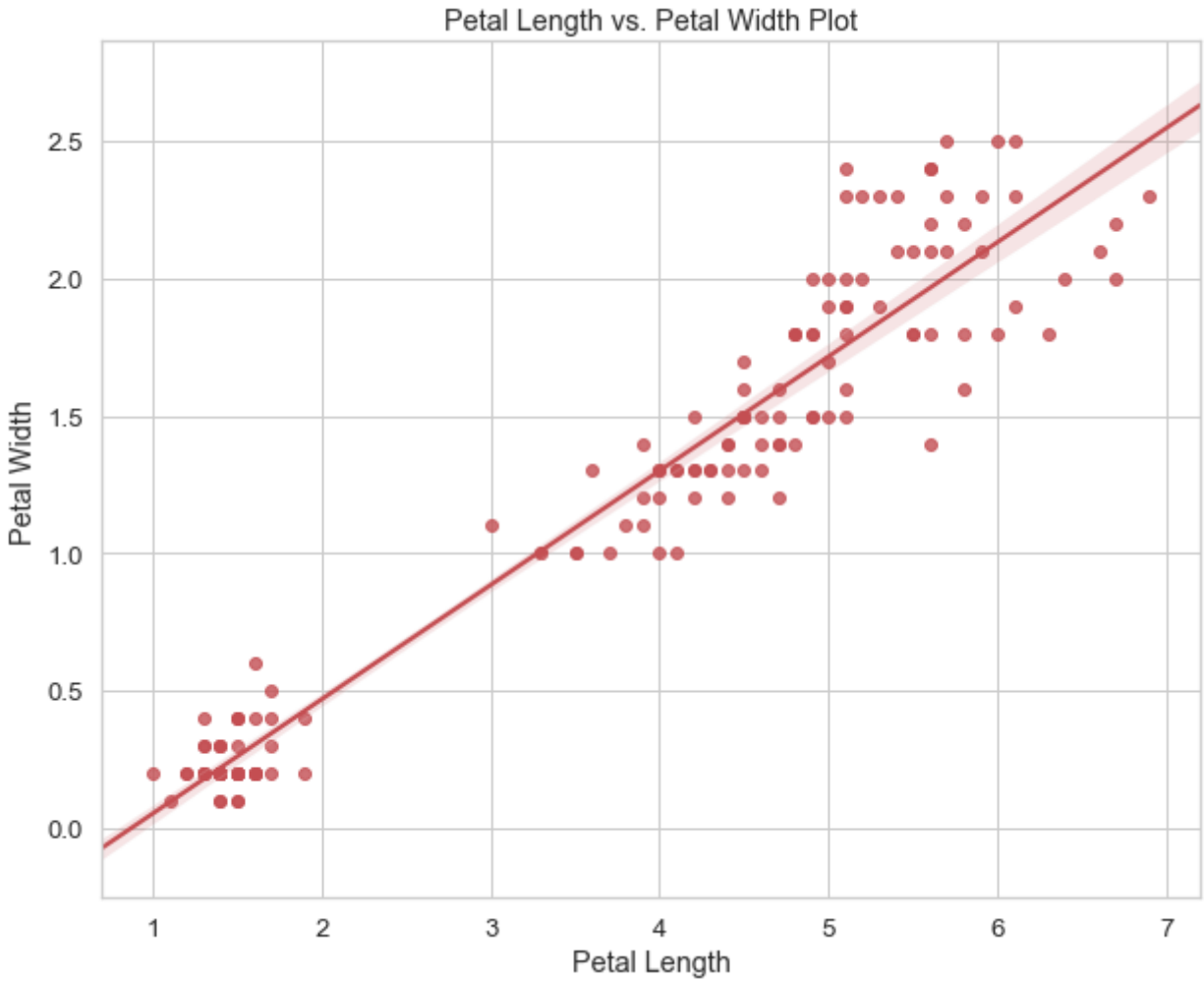
10. regplots and lmplots.

- regplot() performs a simple linear regression model fit and plot. lmplot() combines regplot() and FacetGrid.

- The FacetGrid class helps in visualizing the distribution of one variable as well as the relationship between multiple variables separately within subsets of your dataset using multiple panels.
- lmplot() is more computationally intensive and is intended as a convenient interface to fit regression models across conditional subsets of a dataset.

regplot()

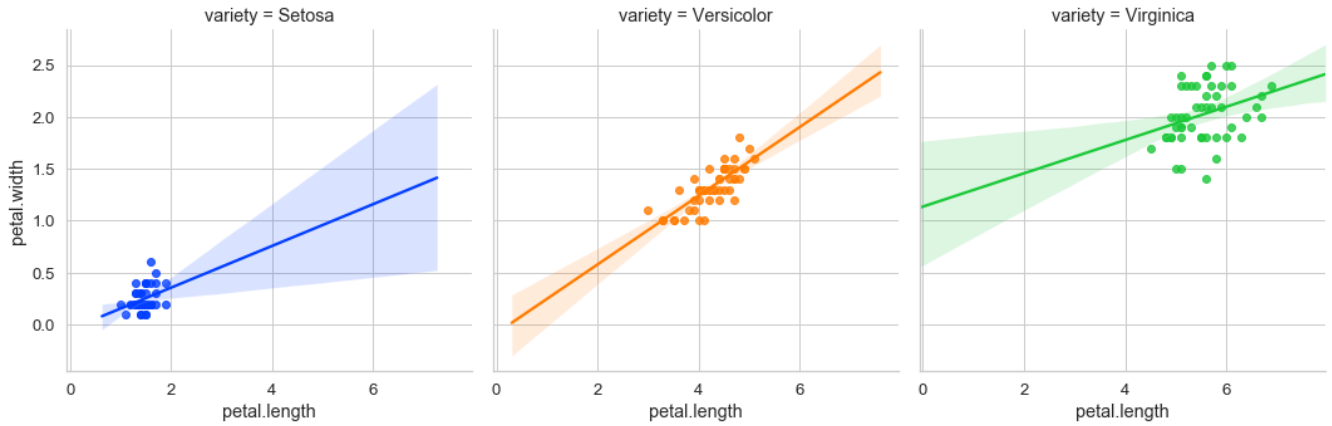
In [41]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

sns.regplot(x = iris['petal.length'], y = iris['petal.width'], color = 'r')

#sns.lmplot(x = iris['petal.length'], y = iris['petal.width'], data = iris)

plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.title('Petal Length vs. Petal Width Plot')

plt.show()
```



lmplot()

In [63]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

#sns.regplot(x = iris['petal.length'], y = iris['petal.width'], color = 'r')

sns.lmplot(data = iris,x = 'petal.length', y = 'petal.width', hue = 'variety', col = 'v

plt.show()
```

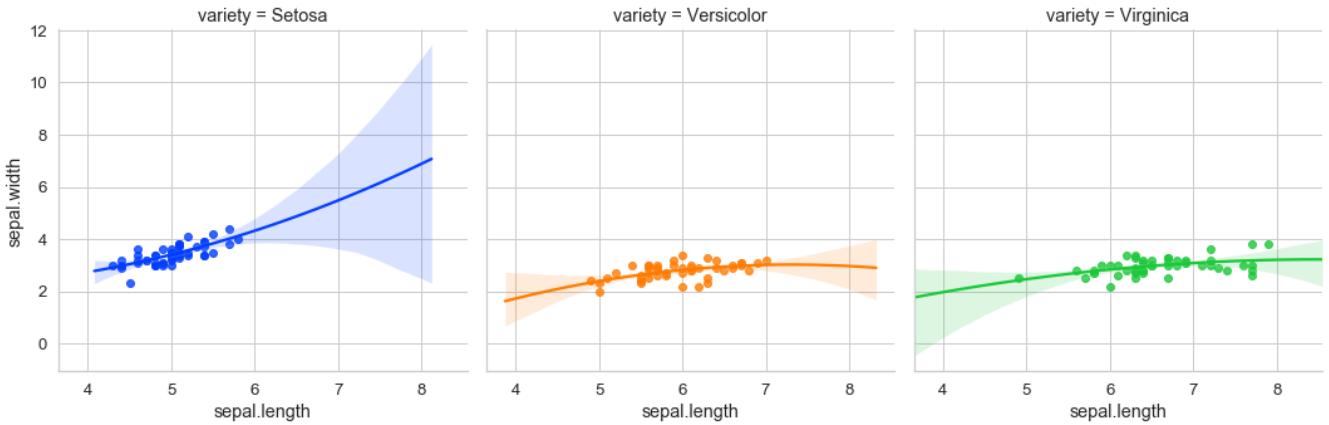
```
<Figure size 720x576 with 0 Axes>
```



Polynomial fit of order 2.

In [66]:
```
plt.figure( figsize = (10,8) )
sns.set(style = 'whitegrid', font_scale = 1.2)

#sns.regplot(x = iris['petal.length'], y = iris['petal.width'], color = 'r')

sns.lmplot(data = iris,x = 'sepal.length', y = 'sepal.width', hue = 'variety', col = 'v

plt.show()
```

```
<Figure size 720x576 with 0 Axes>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: