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

In [2]:
```
df = pd.read_csv("one_way_manova.csv")
df.head()
```

Out[2]:

	id	Sepal.Length	Petal.Length	Species
0	1	5.1	1.4	setosa
1	2	4.9	1.4	setosa
2	3	4.7	1.3	setosa
3	4	4.6	1.5	setosa
4	5	5.0	1.4	setosa

Summary Statistics

In [3]:
```
stats = df.groupby(['Species'])['Sepal.Length', 'Petal.Length'].agg(['count', 'std', 'mean'])
stats
```

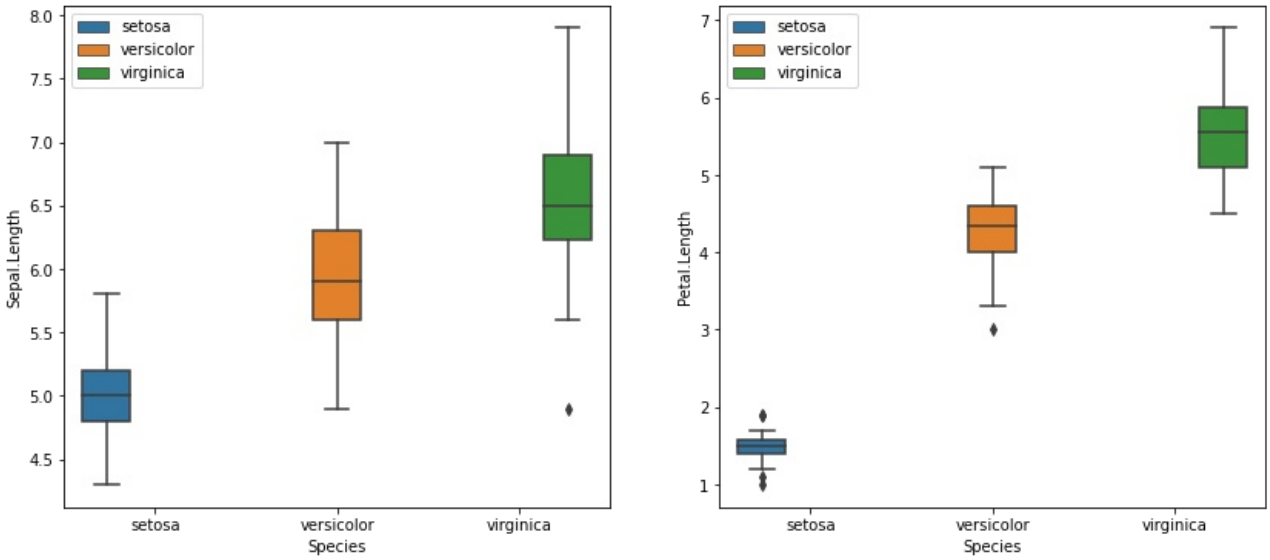
```
C:\Users\Administrator\AppData\Local\Temp\ipykernel_5784\1053844134.py:1: FutureWarning: Indexing with multiple k
eys (implicitly converted to a tuple of keys) will be deprecated, use a list instead.
  stats = df.groupby(['Species'])['Sepal.Length', 'Petal.Length'].agg(['count', 'std', 'mean'])
```

Out[3]:

	Sepal.Length			Petal.Length		
	count	std	mean	count	std	mean
Species						
setosa	50	0.352490	5.006	50	0.173664	1.462
versicolor	50	0.516171	5.936	50	0.469911	4.260
virginica	50	0.635880	6.588	50	0.551895	5.552

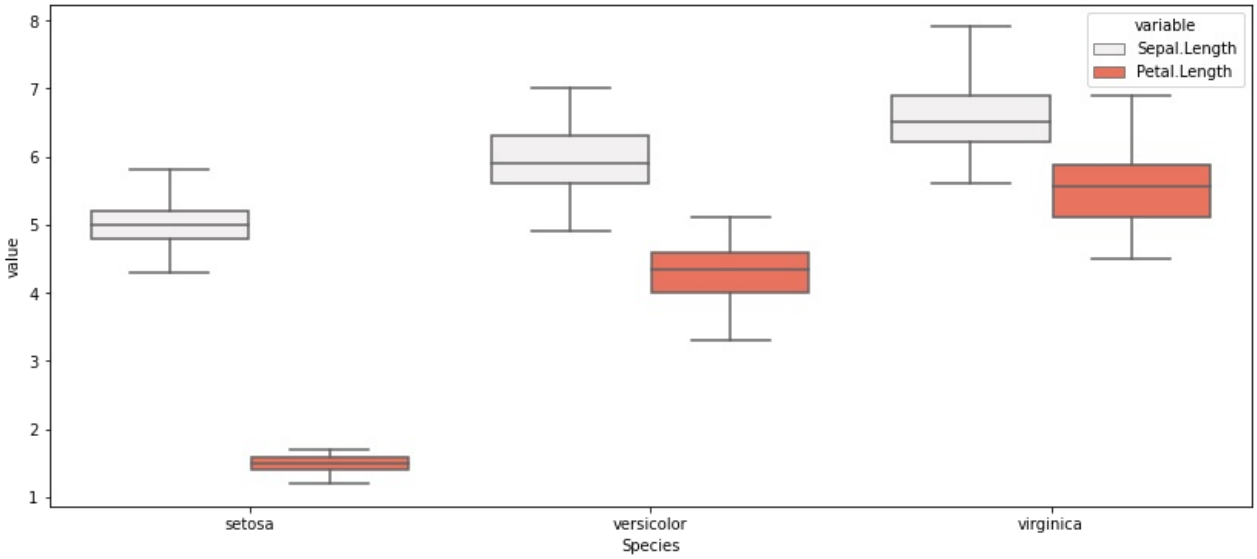
Boxplot

In [4]:
```
import seaborn as sns
import matplotlib.pyplot as plt
fig, axs = plt.subplots(ncols=2, figsize=(14,6))
sns.boxplot(data=df, x="Species", y="Sepal.Length", hue=df.Species.tolist(), ax=axs[0])
sns.boxplot(data=df, x="Species", y="Petal.Length", hue=df.Species.tolist(), ax=axs[1])
plt.show()
```



In [5]:

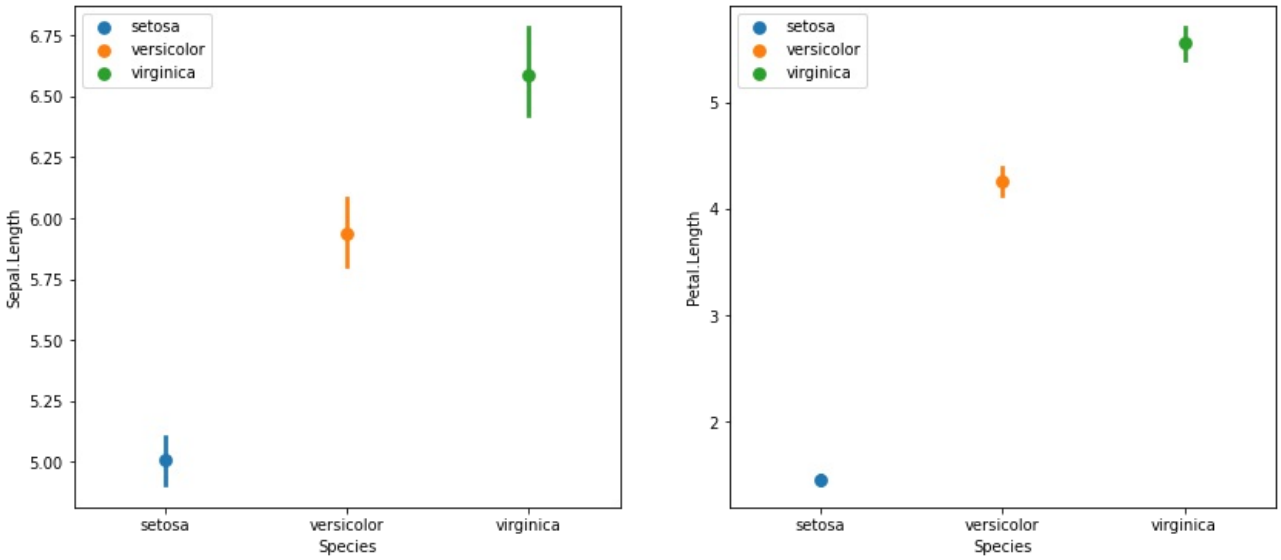
```
import seaborn as sns
import matplotlib.pyplot as plt
fig = plt.subplots(ncols=1, figsize=(14,6))
dfl = pd.melt(df, id_vars='Species', value_vars=['Sepal.Length', 'Petal.Length'])
sns.boxplot(x='Species', y='value', data=dfl, showfliers=False, color='tomato', hue='variable')
plt.show()
```



Interaction plot using Seaborn

In [6]:

```
import seaborn as sns
import matplotlib.pyplot as plt
fig, axs = plt.subplots(ncols=2, figsize=(14,6))
sns.pointplot(data=df, x="Species", y="Sepal.Length", hue=df.Species.tolist(), ax=axs[0])
sns.pointplot(data=df, x="Species", y="Petal.Length", hue=df.Species.tolist(), ax=axs[1])
plt.show()
```



Assumptions

Outliers

Identify univariate outliers

In [7]:

```
def outliers(dep, ind):
    final = []
    for ind_cat in pd.unique(df[ind]):
        Q1, Q3 = np.percentile(df[dep][df[ind] == ind_cat], [25,75])
        IQR = Q3-Q1
        lower_fence, upper_fence= Q1-(1.5*IQR), Q3+(1.5*IQR)
```

```
        new_dataset = []
        for i in df[dep][df[ind] == ind_cat]:
            if i < lower_fence or i > upper_fence:
                new_dataset.append(i)
        final.append({'Outliers':{f'Outliers in {ind_cat}': new_dataset}})
    return pd.DataFrame(final)
```

In [8]:
```
outliers('Sepal.Length', 'Species')
```

Out[8]:

	Outliers
0	{'Outliers in setosa': []}
1	{'Outliers in versicolor': []}
2	{'Outliers in virginica': [4.9]}

In [9]:
```
def outliers(dep, ind):
    final = []
    for ind_cat in pd.unique(df[ind]):
        Q1, Q3 = np.percentile(df[dep][df[ind] == ind_cat], [25,75])
        IQR = Q3-Q1
        lower_fence, upper_fence= Q1-(1.5*IQR), Q3+(1.5*IQR)
        new_dataset = []
        for i in df[dep][df[ind] == ind_cat]:
            if i < lower_fence or i > upper_fence:
                new_dataset.append(i)
        final.append({'Outliers':{f'Outliers in {ind_cat}': new_dataset}})
    return pd.DataFrame(final)
```

In [10]:
```
outliers('Petal.Length', 'Species')
```

Out[10]:

	Outliers
0	{'Outliers in setosa': [1.1, 1.0, 1.9, 1.9]}
1	{'Outliers in versicolor': [3.0]}
2	{'Outliers in virginica': []}

Identify multivariate outliers

In [11]:
```
setosa = df[df['Species'] == 'setosa'][['Sepal.Length', 'Petal.Length']]
versicolor = df[df['Species'] == 'versicolor'][['Sepal.Length', 'Petal.Length']]
virginica = df[df['Species'] == 'virginica'][['Sepal.Length', 'Petal.Length']]
```

In [12]:
```
def Mahalanobis_distance(categories):
    import numpy as np
    import pandas as pd
    import scipy as stats
    from scipy.stats import chi2

    # calculateMahalanobis Function to calculate
    # the Mahalanobis distance
    def calculateMahalanobis(y=None, data=None, cov=None):
        y_mu = y - np.mean(data)
        if not cov:
            cov = np.cov(data.values.T)
        inv_covmat = np.linalg.inv(cov)
        left = np.dot(y_mu, inv_covmat)
        mahal = np.dot(left, y_mu.T)
        return mahal.diagonal()

    results = pd.DataFrame()

    i = ['setosa', 'versicolor', 'virginica']
    for i, cat in zip(i, categories):
        # Creating a new column in the dataframe that holds
        # the Mahalanobis distance for each row
        results[f'{i}_Mahalanobis'] = calculateMahalanobis(y=cat, data=cat)

        # calculate p-value for each mahalanobis distance
        results[f'{i}_p'] = 1 - chi2.cdf(results[f'{i}_Mahalanobis'], 3)
    return results
```

In [13]:
```
Mahalanobis_distance([setosa, versicolor, virginica])
```

```
C:\ProgramData\Anaconda3\envs\All\lib\site-packages\numpy\core\fromnumeric.py:3438: FutureWarning: In a future ve
rsion, DataFrame.mean(axis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior,
use 'frame.mean(axis=0)' or just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
C:\ProgramData\Anaconda3\envs\All\lib\site-packages\numpy\core\fromnumeric.py:3438: FutureWarning: In a future ve
rsion, DataFrame.mean(axis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior,
use 'frame.mean(axis=0)' or just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
C:\ProgramData\Anaconda3\envs\All\lib\site-packages\numpy\core\fromnumeric.py:3438: FutureWarning: In a future ve
rsion, DataFrame.mean(axis=None) will return a scalar mean over the entire DataFrame. To retain the old behavior,
use 'frame.mean(axis=0)' or just 'frame.mean()'
  return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

Out[13]:

	setosa_Mahalanobis	setosa_p	versicolor_Mahalanobis	versicolor_p	virginica_Mahalanobis	virginica_p
0	0.268621	0.965818	5.134374	0.162218	5.924332	0.115350
1	0.172860	0.981847	0.872792	0.831988	1.786526	0.617871
2	1.282637	0.733260	3.492890	0.321685	0.665170	0.881364
3	1.625224	0.653684	0.729700	0.866197	1.109309	0.774827
4	0.134070	0.987457	1.217263	0.748867	1.298084	0.729588
5	2.486509	0.477734	1.905510	0.592248	3.615587	0.306077
6	1.329274	0.722194	0.876748	0.831035	7.641622	0.054029
7	0.054015	0.996715	4.678285	0.196928	1.847442	0.604666
8	2.966920	0.396754	1.795616	0.615889	0.379849	0.944373
9	0.186806	0.979691	2.254587	0.521277	1.028931	0.794252
10	1.256257	0.739546	3.433086	0.329545	1.951687	0.582495
11	1.315010	0.725572	0.017935	0.999365	0.247181	0.969636
12	0.384990	0.943325	0.985073	0.804864	0.688722	0.875853
13	6.596634	0.085928	1.226295	0.746705	2.118968	0.548085
14	9.870519	0.019699	2.358788	0.501352	1.786526	0.617871
15	3.978015	0.263848	3.742409	0.290656	0.247181	0.969636
16	2.882501	0.410098	2.749046	0.431957	0.021697	0.999156
17	0.268621	0.965818	0.116039	0.989846	4.336300	0.227365
18	4.644259	0.199778	0.297849	0.960433	6.483808	0.090303
19	0.094564	0.992482	0.599357	0.896580	1.014754	0.797682
20	2.486509	0.477734	3.352467	0.340407	0.336735	0.952978
21	0.094564	0.992482	1.558151	0.668916	2.517076	0.472213
22	7.286713	0.063300	2.094963	0.552933	4.336300	0.227365
23	1.888825	0.595799	1.226295	0.746705	2.670578	0.445251
24	8.065972	0.044668	1.622393	0.654323	0.084137	0.993671
25	0.688084	0.876003	2.701812	0.439920	0.927887	0.818693
26	0.688084	0.876003	2.831428	0.418352	3.128522	0.372231
27	0.308454	0.958431	2.678449	0.443902	1.649637	0.648188
28	0.576512	0.901788	0.418910	0.936309	0.550817	0.907593
29	1.888484	0.595872	3.962841	0.265505	1.504049	0.681336
30	1.315010	0.725572	0.984571	0.804985	1.679024	0.641607
31	1.256257	0.739546	1.426927	0.699235	4.497423	0.212520
32	0.308454	0.958431	0.815748	0.845697	0.550817	0.907593
33	2.540230	0.468064	6.667745	0.083277	0.927415	0.818807
34	0.186806	0.979691	4.958121	0.174890	2.812542	0.421440
35	2.436552	0.486867	0.418910	0.936309	4.119788	0.248816
36	3.804416	0.283373	2.265670	0.519129	1.109309	0.774827
37	0.172860	0.981847	0.624025	0.890911	0.190188	0.979158
38	3.197090	0.362224	0.476134	0.924102	2.109247	0.550044
39	0.094564	0.992482	0.729700	0.866197	2.173604	0.537166
40	0.928249	0.818606	2.739314	0.433588	0.047842	0.997256
41	2.385598	0.496321	0.643864	0.886321	6.345227	0.095970
42	3.197090	0.362224	0.360920	0.948192	1.786526	0.617871
43	0.688084	0.876003	4.346203	0.226426	0.574401	0.902267
44	6.539581	0.088115	0.729442	0.866258	0.084137	0.993671

45	0.384990	0.943325	0.318272	0.956556	2.496833	0.475864
46	0.634632	0.888460	0.318272	0.956556	1.669288	0.643783
47	1.329274	0.722194	0.470962	0.925223	1.080063	0.781889
48	0.695685	0.874218	7.564762	0.055918	0.623041	0.891138
49	0.134070	0.987457	0.209086	0.976109	1.223871	0.747285

In [14]:
```
# There were no multivariate outliers in the data, as assessed by Mahalanobis distance (p > 0.001)
```

Normality

Check univariate normality assumption

In [15]:
```
setosa = df[df['Species'] == 'setosa'][['Sepal.Length', 'Petal.Length']]
versicolor = df[df['Species'] == 'versicolor'][['Sepal.Length', 'Petal.Length']]
virginica = df[df['Species'] == 'virginica'][['Sepal.Length', 'Petal.Length']]
```

In [16]:
```
def shapiro(categories):
    normality = []
    import scipy.stats as stats
    i = ['setosa', 'versicolor', 'virginica']
    for i, cat in zip(i, categories):
        columns = cat.columns
        for j in columns:
            result = stats.shapiro(cat[j])
            normality.append({'Shapiro':{f'{i}, {j}:': result}})
    return pd.DataFrame(normality)
```

In [17]:
```
shapiro([setosa, versicolor, virginica])
```

Out[17]:

	Shapiro
0	{'setosa, Sepal.Length:': (0.9776989221572876,...
1	{'setosa, Petal.Length:': (0.9549766182899475,...
2	{'versicolor, Sepal.Length:': (0.9778355956077...
3	{'versicolor, Petal.Length:': (0.9660047888755...
4	{'virginica, Sepal.Length:': (0.97117984294891...
5	{'virginica, Petal.Length:': (0.96218627691268...

Multivariate normality

In [18]:
```
X = df[['Sepal.Length', 'Petal.Length']]
```

In [19]:
```
from pingouin import multivariate_normality
#perform the Henze-Zirkler Multivariate Normality Test
multivariate_normality(X, alpha=.05)
```

Out[19]:
```
HZResults(hz=5.450383639328617, pval=1.035769444922176e-11, normal=False)
```

Identify multicollinearity

In [20]:
```
import pingouin as pg
```

In [21]:
```
pg.corr(df['Sepal.Length'], df['Petal.Length']).round(3)
```

Out[21]:

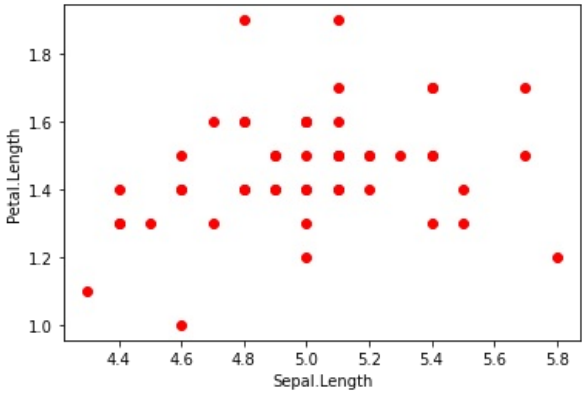
	n	r	CI95%	p-val	BF10	power
pearson	150	0.872	[0.83, 0.91]	0.0	1.811e+44	1.0

Check linearity assumption

Check linearity assumption

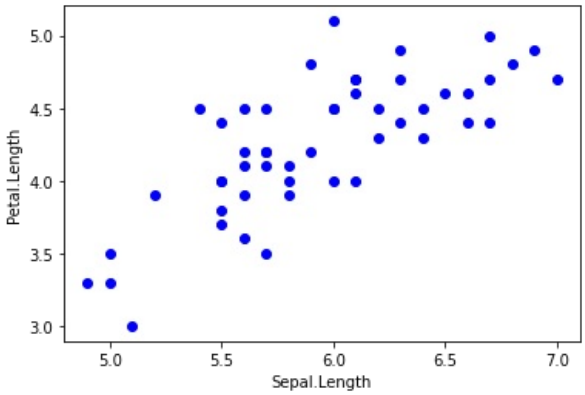
In [22]:
```
# Here we are plotting sepal_length vs sepal_width
# setosa - 'red'; versicolor - 'blue'; virginica - 'green'
print(pg.corr(df['Sepal.Length'][df['Species']=='setosa'], df['Petal.Length'][df['Species']=='setosa']).round(3))
for n in range(0,150):
    if df['Species'][n] == 'setosa':
        plt.scatter(df['Sepal.Length'][n], df['Petal.Length'][n], color = 'red')
        plt.xlabel('Sepal.Length')
        plt.ylabel('Petal.Length')
```

```
           n      r          CI95%  p-val   BF10  power
pearson   50  0.267  [-0.01, 0.51]  0.061  0.975  0.473
```



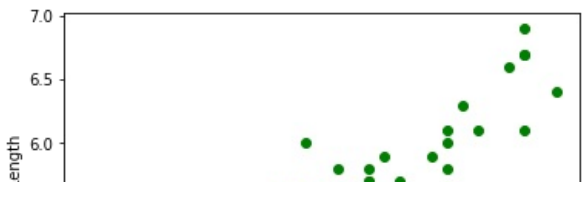
In [23]:
```
# Here we are plotting sepal_length vs sepal_width
# setosa - 'red'; versicolor - 'blue'; virginica - 'green'
print(pg.corr(df['Sepal.Length'][df['Species']=='versicolor'], df['Petal.Length'][df['Species']=='versicolor']).r
for n in range(0,150):
    if df['Species'][n] == 'versicolor':
        plt.scatter(df['Sepal.Length'][n], df['Petal.Length'][n], color = 'blue')
        plt.xlabel('Sepal.Length')
        plt.ylabel('Petal.Length')
```

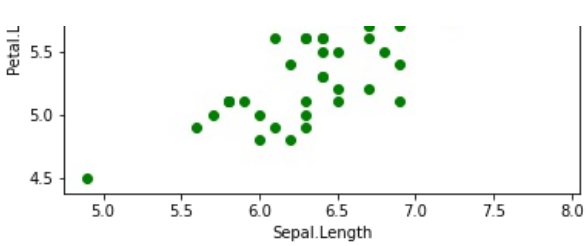
```
           n      r        CI95%  p-val       BF10  power
pearson   50  0.754  [0.6, 0.85]    0.0  4.637e+07    1.0
```



In [24]:
```
# Here we are plotting sepal_length vs sepal_width
# setosa - 'red'; versicolor - 'blue'; virginica - 'green'
print(pg.corr(df['Sepal.Length'][df['Species']=='virginica'], df['Petal.Length'][df['Species']=='virginica']).rou
for n in range(0,150):
    if df['Species'][n] == 'virginica':
        plt.scatter(df['Sepal.Length'][n], df['Petal.Length'][n], color = 'green')
        plt.xlabel('Sepal.Length')
        plt.ylabel('Petal.Length')
```

```
           n      r         CI95%  p-val       BF10  power
pearson   50  0.864  [0.77, 0.92]    0.0  9.996e+12    1.0
```





Check the homogeneity of covariances assumption

In [25]:
```
import pingouin as pg
```

In [26]:
```
pg.box_m(df, dvs=['Sepal.Length', 'Petal.Length'], group='Species')
```

Out[26]:

	Chi2	df	pval	equal_cov
box	58.375558	6.0	9.616505e-11	False

Check the homogneity of variance assumption

In [27]:
```
# if normality exists then perform bartlett otherwise levene's tests
```

In [28]:
```
# Null Hypothesis: the variances are equal across all samples/groups
# Alternative Hypothesis:  the variances are not equal across all samples/groups
```

In [29]:
```
from scipy.stats import levene
Sepal_Length_Levene = levene(df['Sepal.Length'][df['Species']=='setosa'],df['Sepal.Length'][df['Species']=='versi
print(Sepal_Length_Levene)
Petal_Length_Levene = levene(df['Petal.Length'][df['Species']=='setosa'],df['Petal.Length'][df['Species']=='versi
print(Petal_Length_Levene)
```

```
LeveneResult(statistic=6.35272002048269, pvalue=0.0022585277836218586)
LeveneResult(statistic=19.480338801923573, pvalue=3.1287566394085344e-08)
```

In [30]:
```
from scipy.stats import bartlett
Sepal_Length_bartlett = bartlett(df['Sepal.Length'][df['Species']=='setosa'],df['Sepal.Length'][df['Species']=='v
print(Sepal_Length_bartlett)
Petal_Length_bartlett = bartlett(df['Petal.Length'][df['Species']=='setosa'],df['Petal.Length'][df['Species']=='v
print(Petal_Length_bartlett)
```

```
BartlettResult(statistic=16.005701874401502, pvalue=0.0003345076070163035)
BartlettResult(statistic=55.42250284023702, pvalue=9.229037733034152e-13)
```

One-Way MANOVA Computation

In [31]:
```
df['Sepal_Length'] = df['Sepal.Length']
df['Petal_Length'] = df['Petal.Length']
```

In [32]:
```
from statsmodels.multivariate.manova import MANOVA
fit = MANOVA.from_formula('Sepal_Length + Petal_Length  ~ Species', data=df)
fit.mv_test().summary()
```

```
C:\ProgramData\Anaconda3\envs\All\lib\site-packages\statsmodels\compat\pandas.py:61: FutureWarning: pandas.Int64I
ndex is deprecated and will be removed from pandas in a future version. Use pandas.Index with the appropriate dty
pe instead.
  from pandas import Int64Index as NumericIndex
```

Out[32]:

Intercept	Value	Num DF	Den DF	F Value	Pr > F
Wilks' lambda	0.0219	2.0000	146.0000	3255.3901	0.0000
Pillai's trace	0.9781	2.0000	146.0000	3255.3901	0.0000
Hotelling-Lawley trace	44.5944	2.0000	146.0000	3255.3901	0.0000

Roy's greatest root	44.5944	2.0000	146.0000	3255.3901	0.0000

Species	Value	Num DF	Den DF	F Value	Pr > F
Wilks' lambda	0.0399	4.0000	292.0000	292.5565	0.0000
Pillai's trace	0.9885	4.0000	294.0000	71.8288	0.0000
Hotelling-Lawley trace	23.3647	4.0000	174.1653	850.8986	0.0000
Roy's greatest root	23.3342	2.0000	147.0000	1715.0602	0.0000

Post-hoct tests

Compute univariate one-way ANOVA

Compute simple main effects

Note that, there are different R function to compute one-way ANOVA depending whether the assumptions are met or not: anova: can be used when normality and homogeneity of variance assumptions are met welch_anova_test: can be used when the homogeneity of variance assumption is violated, as in our example. kruskal_test: Kruskal-Wallis test, a non parametric alternative of one-way ANOVA test

anova

In [33]:
```
def anova():
    import pingouin as pg
    series = ['Sepal.Length', 'Petal.Length']
    aov_results = []
    for i in series:
        aov = pg.anova(dv=i, between='Species', data=df, detailed=True)
        dicts = {f'{i}':aov}
        aov_results.append(dicts)
    return aov_results
anova()
```

Out[33]:
```
[{'Sepal.Length':     Source         SS   DF         MS           F         p-unc       np2
  0  Species  63.212133    2  31.606067  119.264502  1.669669e-31  0.618706
  1   Within  38.956200  147   0.265008         NaN           NaN       NaN},
 {'Petal.Length':     Source         SS   DF          MS            F         p-unc       np2
  0  Species  437.1028    2  218.551400  1180.161182  2.856777e-91  0.941372
  1   Within   27.2226  147    0.185188          NaN           NaN       NaN}]
```

welch_anova

In [34]:
```
def anova():
    from pingouin import welch_anova
    series = ['Sepal.Length', 'Petal.Length']
    aov_results = []
    for i in series:
        wel = welch_anova(dv=i, between='Species', data=df)
        dicts = {f'{i}':wel}
        aov_results.append(dicts)
    return aov_results
anova()
```

Out[34]:
```
[{'Sepal.Length':     Source  ddof1      ddof2           F         p-unc       np2
  0  Species      2  92.211145  138.908285  1.505059e-28  0.618706},
 {'Petal.Length':     Source  ddof1      ddof2            F         p-unc       np2
  0  Species      2  78.072955  1828.091945  2.693327e-66  0.941372}]
```

kruskal

In [35]:
```
def kruskal():
    from pingouin import kruskal
    series = ['Sepal.Length', 'Petal.Length']
    aov_results = []
    for i in series:
        krus = kruskal(dv=i, between='Species', data=df)
        dicts = {f'{i}':krus}
```

```
        aov_results.append(dicts)
    return aov_results
kruskal()
```

Out[35]:
```
[{'Sepal.Length':           Source  ddof1          H         p-unc
  Kruskal  Species      2  96.937436  8.918734e-22},
 {'Petal.Length':           Source  ddof1          H         p-unc
  Kruskal  Species      2  130.411049  4.803974e-29}]
```

Compute multiple pairwise comparisons

If you had violated the assumption of homogeneity of variances, as in our example, you might prefer to run a Games-Howell post-hoc test insted of tukey_hsd

In [36]:
```
import pingouin as pg
pg.pairwise_gameshowell(data=df, dv='Sepal.Length', between='Species').round(3)
```

Out[36]:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	hedges
0	setosa	versicolor	5.006	5.936	-0.930	0.088	-10.521	86.538	0.0	-2.088
1	setosa	virginica	5.006	6.588	-1.582	0.103	-15.386	76.516	0.0	-3.054
2	versicolor	virginica	5.936	6.588	-0.652	0.116	-5.629	94.025	0.0	-1.117

In [37]:
```
import pingouin as pg
pg.pairwise_gameshowell(data=df, dv='Petal.Length', between='Species').round(3)
```

Out[37]:

	A	B	mean(A)	mean(B)	diff	se	T	df	pval	hedges
0	setosa	versicolor	1.462	4.260	-2.798	0.071	-39.493	62.140	0.0	-7.838
1	setosa	virginica	1.462	5.552	-4.090	0.082	-49.986	58.609	0.0	-9.921
2	versicolor	virginica	4.260	5.552	-1.292	0.103	-12.604	95.570	0.0	-2.501

Report

In [38]:
```
from statannotations.Annotator import Annotator

states_palette = sns.color_palette("YlGnBu", n_colors=3)

states_order = ["Sepal.Length", "Petal.Length"]

pvalues = [0.0001, 0.0001, 0.0001, 0.0001, 0.0001, 0.0001]

# Putting the parameters in a dictionary avoids code duplication
# since we use the same for `sns.boxplot` and `Annotator` calls..

pairs = [
    [('setosa', 'Sepal.Length'), ('versicolor', 'Sepal.Length')],
    [('setosa', 'Sepal.Length'), ('virginica', 'Sepal.Length')],
    [('versicolor', 'Sepal.Length'), ('virginica', 'Sepal.Length')],

    [('setosa', 'Petal.Length'), ('versicolor', 'Petal.Length')],
    [('setosa', 'Petal.Length'), ('virginica', 'Petal.Length')],
    [('versicolor', 'Petal.Length'), ('virginica', 'Petal.Length')],
]

dfl = pd.melt(df, id_vars='Species', value_vars=['Sepal.Length', 'Petal.Length'])

plotting_parameters = {
    'data': dfl,
    'x': 'Species',
    'y': "value",
    'showfliers':False,
    'color':'tomato',
    "hue": 'variable',
    "hue_order": states_order,
    "palette": states_palette
}

with sns.plotting_context('notebook', font_scale = 1.4):

    from matplotlib import pyplot as plt
```

```
    plt.figure(figsize=(15,8))

    # Plot with seaborn
    ax = sns.boxplot(**plotting_parameters)

    # Add annotations
    annotator = Annotator(ax, pairs, **plotting_parameters)
    annotator.set_pvalues(pvalues)
    annotator.annotate()

    # Label and show
    plt.show()
```

```
p-value annotation legend:
      ns: p <= 1.00e+00
       *: 1.00e-02 < p <= 5.00e-02
      **: 1.00e-03 < p <= 1.00e-02
     ***: 1.00e-04 < p <= 1.00e-03
    ****: p <= 1.00e-04

setosa_Sepal.Length vs. versicolor_Sepal.Length: Custom statistical test, P_val:1.000e-04
versicolor_Sepal.Length vs. virginica_Sepal.Length: Custom statistical test, P_val:1.000e-04
setosa_Petal.Length vs. versicolor_Petal.Length: Custom statistical test, P_val:1.000e-04
versicolor_Petal.Length vs. virginica_Petal.Length: Custom statistical test, P_val:1.000e-04
setosa_Sepal.Length vs. virginica_Sepal.Length: Custom statistical test, P_val:1.000e-04
setosa_Petal.Length vs. virginica_Petal.Length: Custom statistical test, P_val:1.000e-04
```