Importing neccessary libraries.

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

Reading and Understanding about data.

```
In [2]:
df=pd.read_csv('iris.csv')
```

```
In [3]:
df
```

```
Out[3]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	Iris-virginica
146	147	6.3	2.5	5.0	1.9	Iris-virginica
147	148	6.5	3.0	5.2	2.0	Iris-virginica
148	149	6.2	3.4	5.4	2.3	Iris-virginica
149	150	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 6 columns

```
In [4]:
df.isnull().sum()
```

```
Out[4]:
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

```
In [5]:
df.describe()
```

```
Out[5]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

Encoding

```
In [6]:
from sklearn.preprocessing import LabelEncoder
```

```
In [7]:
lab_encLabelEncoder()
df['Species']=lab_enc.fit_transform(df['Species'])
```

```
In [8]:
df
```

```
Out[8]:
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	0
1	2	4.9	3.0	1.4	0.2	0
2	3	4.7	3.2	1.3	0.2	0
3	4	4.6	3.1	1.5	0.2	0
4	5	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	2
146	147	6.3	2.5	5.0	1.9	2
147	148	6.5	3.0	5.2	2.0	2
148	149	6.2	3.4	5.4	2.3	2
149	150	5.9	3.0	5.1	1.8	2

150 rows × 6 columns

Identifying target column and removing unnecessary columns.

```
In [9]:
df=df.drop(['Id'],axis=1)
```

```
In [10]:
df
```

```
Out[10]:
```

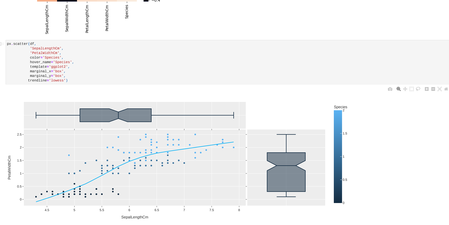
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	2
146	6.3	2.5	5.0	1.9	2
147	6.5	3.0	5.2	2.0	2
148	6.2	3.4	5.4	2.3	2
149	5.9	3.0	5.1	1.8	2

150 rows × 5 columns

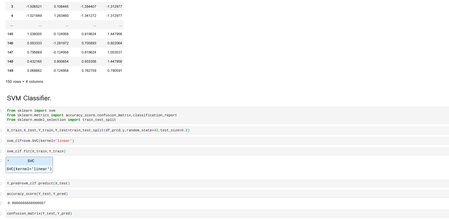
Exploratory Data Analysis.

```
In [11]:
import pandas_profiling as pp
```

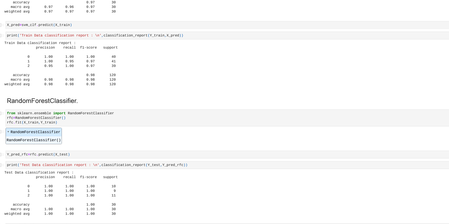
```
In [12]:
sns.heatmap(corr,annot=True, square=True,robust=True)
plt.show()
```



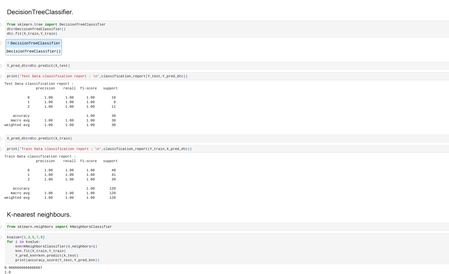

```
In [13]:
px.scatter(df,
           'SepalLengthCm',
           'PetalWidthCm',
           color='Species',
           hover_name='Species',
           template='ggplot2',
           marginal_x='box',
           marginal_y='box',
           trendline='lowess')
```




```
In [14]:
px.scatter(df,
           'SepalLengthCm',
           'PetalLengthCm',
           color='Species',
           hover_name='Species',
           template='ggplot2',
           trendline='lowess')
```




```
In [15]:
px.scatter(df,
           'PetalLengthCm',
           'PetalWidthCm',
           color='Species',
           hover_name='Species',
           template='plotly_dark',
           trendline='lowess')
```



Scaling.

```
In [16]:
from sklearn.preprocessing import StandardScaler
```

```
In [17]:
y=df['Species']
df=df.drop('Species',axis=1)
```

```
In [18]:
std_slcStandardScaler()
cols=df.columns
df_std=pd.DataFrame(std_slc.fit_transform(df),columns=cols)
```

```
In [19]:
df_std
```

```
Out[19]:
```

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977
...	...	...	...	...
145	1.038005	-0.124958	0.819624	1.447956
146	0.553333	-1.281972	0.705893	0.922304
147	0.795669	-0.124958	0.819624	1.053537
148	0.432165	0.800654	0.933356	1.447956
149	0.068662	-0.124958	0.762759	0.790591

150 rows × 4 columns

SVM Classifier.

```
In [20]:
from sklearn import svm
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
from sklearn.model_selection import train_test_split
```

```
In [21]:
X_train,X_test,y_train,y_test=train_test_split(df_std,y,random_state=42,test_size=0.2)
```

```
In [22]:
svm_clf=svm.SVC(kernel='linear')
```

```
In [23]:
svm_clf.fit(X_train,y_train)
```

```
Out[23]:
SVC(kernel='linear')
```

```
In [24]:
y_pred=svm_clf.predict(X_test)
```

```
In [25]:
accuracy_score(y_test,y_pred)
```

```
Out[25]:
0.9666666666666667
```

```
In [26]:
confusion_matrix(y_test,y_pred)
```

```
Out[26]:
array([[ 9,  0,  0],
       [ 0, 11,  1],
       [ 0,  0,  9]])
```

```
In [27]:
print('Test Data classification report : \n',classification_report(y_test,y_pred))
```

```
Test Data classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      0.85      0.92         9
           2       0.92      1.00      0.96        11

    accuracy                           0.97        30
   macro avg       0.97      0.95      0.96        30
weighted avg       0.97      0.97      0.97        30
```

```
In [28]:
y_pred=svm_clf.predict(X_train)
```

```
In [29]:
print('Train Data classification report : \n',classification_report(y_train,y_pred))
```

```
Train Data classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      0.95      0.97        41
           2       0.95      1.00      0.97        39

    accuracy                           0.98       120
   macro avg       0.98      0.98      0.98       120
weighted avg       0.98      0.98      0.98       120
```

RandomForestClassifier.

```
In [30]:
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier()
rfc.fit(X_train,y_train)
```

```
Out[30]:
RandomForestClassifier()
```

```
In [31]:
y_pred_rfc=rfc.predict(X_test)
```

```
In [32]:
print('Test Data classification report : \n',classification_report(y_test,y_pred_rfc))
```

```
Test Data classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
In [33]:
y_pred_rfc=rfc.predict(X_train)
```

```
In [34]:
print('Train Data classification report : \n',classification_report(y_train,y_pred_rfc))
```

```
Train Data classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        41
           2       1.00      1.00      1.00        39

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

DecisionTreeClassifier.

```
In [35]:
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(X_train,y_train)
```

```
Out[35]:
DecisionTreeClassifier()
```

```
In [36]:
y_pred_dtc=dtc.predict(X_test)
```

```
In [37]:
print('Test Data classification report : \n',classification_report(y_test,y_pred_dtc))
```

```
Test Data classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
In [38]:
y_pred_dtc=dtc.predict(X_train)
```

```
In [39]:
print('Train Data classification report : \n',classification_report(y_train,y_pred_dtc))
```

```
Train Data classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       1.00      1.00      1.00        41
           2       1.00      1.00      1.00        39

    accuracy                           1.00       120
   macro avg       1.00      1.00      1.00       120
weighted avg       1.00      1.00      1.00       120
```

K-nearest neighbours.

```
In [40]:
from sklearn.neighbors import KNeighborsClassifier
```

```
In [41]:
knalist([3,5,7,9]
for i in knalist:
    knnKNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    Y_pred_knn=knn.predict(X_test)
    print(accuracy_score(y_test,y_pred_knn))
```

```
0.9666666666666667
1.0
1.0
1.0
```

```
In [42]:
knnKNeighborsClassifier(n_neighbors=3)
knn.fit(X_train,y_train)
Y_pred_knn=knn.predict(X_test)
print(accuracy_score(y_test,y_pred_knn))
```

```
1.0
```

```
In [43]:
print('Test Data classification report : \n',classification_report(y_test,Y_pred_knn))
```

```
Test Data classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        10
           1       1.00      1.00      1.00         9
           2       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

```
In [44]:
Y_pred_knn=knn.predict(X_train)
print('Train Data classification report : \n',classification_report(y_train,Y_pred_knn))
```

```
Train Data classification report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        40
           1       0.98      0.93      0.95        41
           2       0.93      0.97      0.95        39

    accuracy                           0.97       120
   macro avg       0.97      0.97      0.97       120
weighted avg       0.97      0.97      0.97       120
```

```
weighted avg       0.94      0.94      0.94       120
```

In [ ]: