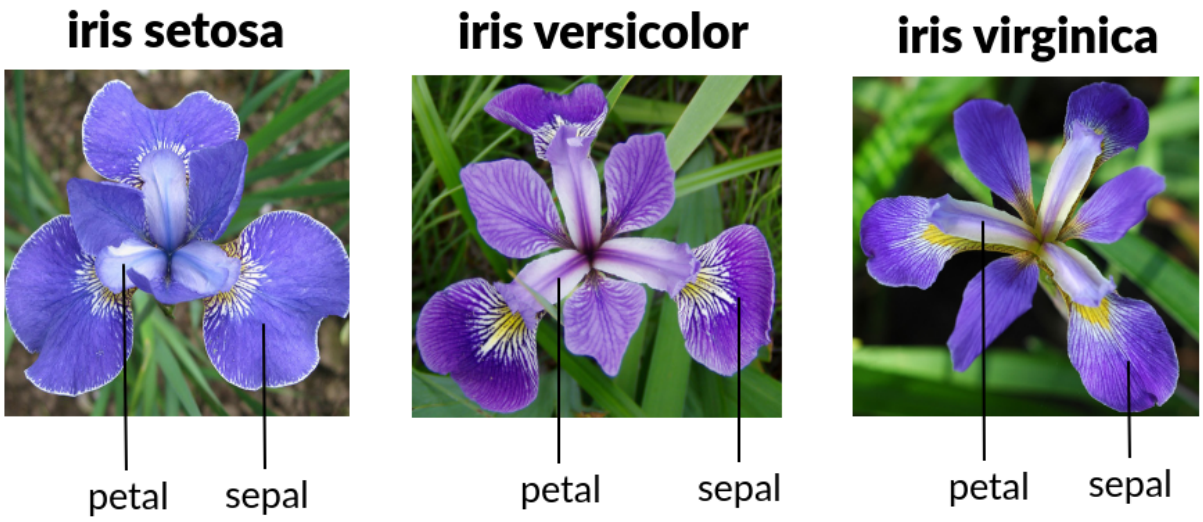
Principal Component Analysis (PCA) - IRIS dataset



Principal Component Analysis

- Principal Component Analysis is an unsupervised learning algorithm that is used for the dimensionality reduction in machine learning. It is a statistical process that converts the observations of correlated features into a set of linearly uncorrelated features with the help of orthogonal transformation. These new transformed features are called the Principal Components.
- It is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

PCA terminology

- Dimensionality: It is the number of features or variables present in the given dataset. More easily, it is the number of columns present in the dataset.
- Correlation: It signifies that how strongly two variables are related to each other. Such as if one changes, the other variable also gets changed. The correlation value ranges from -1 to +1. Here, -1 occurs if variables are inversely proportional to each other, and +1 indicates that variables are directly proportional to each other.
- Orthogonal: It defines that variables are not correlated to each other, and hence the correlation between the pair of variables is zero.
- Eigenvectors: If there is a square matrix M, and a non-zero vector v is given. Then v will be eigenvector if Av is the scalar multiple of v.
- Covariance Matrix: A matrix containing the covariance between the pair of variables is called the Covariance Matrix.
- Explained variance: It is a statistical measure of how much variation in a dataset can be attributed to each of the principal components (eigenvectors) generated by the principal component analysis (PCA) method.

Steps of PCA

- 1. Standardization - To standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.
- 2. Covariance matrix computation - To understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them.
- 3. Compute the eigenvectors & eigenvalues of the covariance matrix to identify the principal components.
- 4. Feature vector - To choose whether to keep all these components or discard those of low eigenvalues, and form with the remaining ones a matrix of vectors.

Objective

- The objective is to build K Nearest Neighbor classifier model using Principal Components Analysis (PCA).

Dataset source & brief

- The Iris dataset has been sourced from Kaggle & was used in R.A. Fisher's classic 1936 paper, The Use of Multiple Measurements in Taxonomic Problems.It includes three iris species with 50 samples each as well as some properties about each flower. One flower species is linearly separable from the other two, but the other two are not linearly separable from each other.
- The columns in this dataset are: Id, SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm & Species

Import the libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
sns.set()
import warnings
warnings.filterwarnings("ignore")
```

Load and read the dataset

In [109]:

```
df=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stats and ML\Dataset\Iris.csv")
df.head()
```

Out[109]:

	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

Basic info about the dataset

In [110]:

```
df.shape  #check shape
```

Out[110]:

```
(150, 6)
```

In [111]:

```
df.columns  #check column names
```

Out[111]:

```
Index(['Id', 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm',
       'Species'],
      dtype='object')
```

In [112]:

```
df.duplicated().sum()  #check duplicates
```

Out[112]:

```
0
```

In [113]:

```
df.isnull().sum()   #check null values
```

Out[113]:

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

In [114]:

```
df.info()    #check info
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Id             150 non-null    int64  
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object 
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

In [115]:

```
df.describe().style.background_gradient(cmap='Reds')     #statistical summary
```

Out[115]:

	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

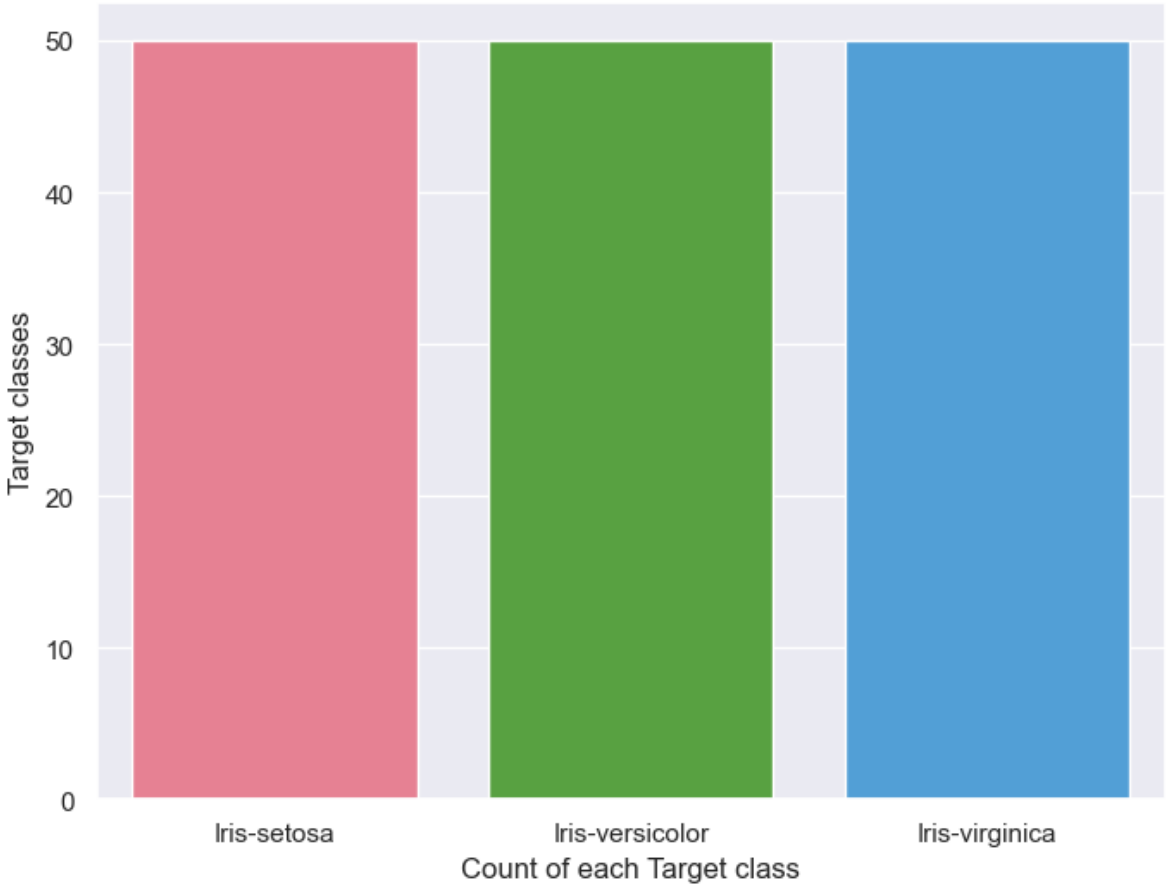
In [116]:

```
df.drop(['Id'],axis=1,inplace=True)  # Dropping non significant variable
```

Data Visualization

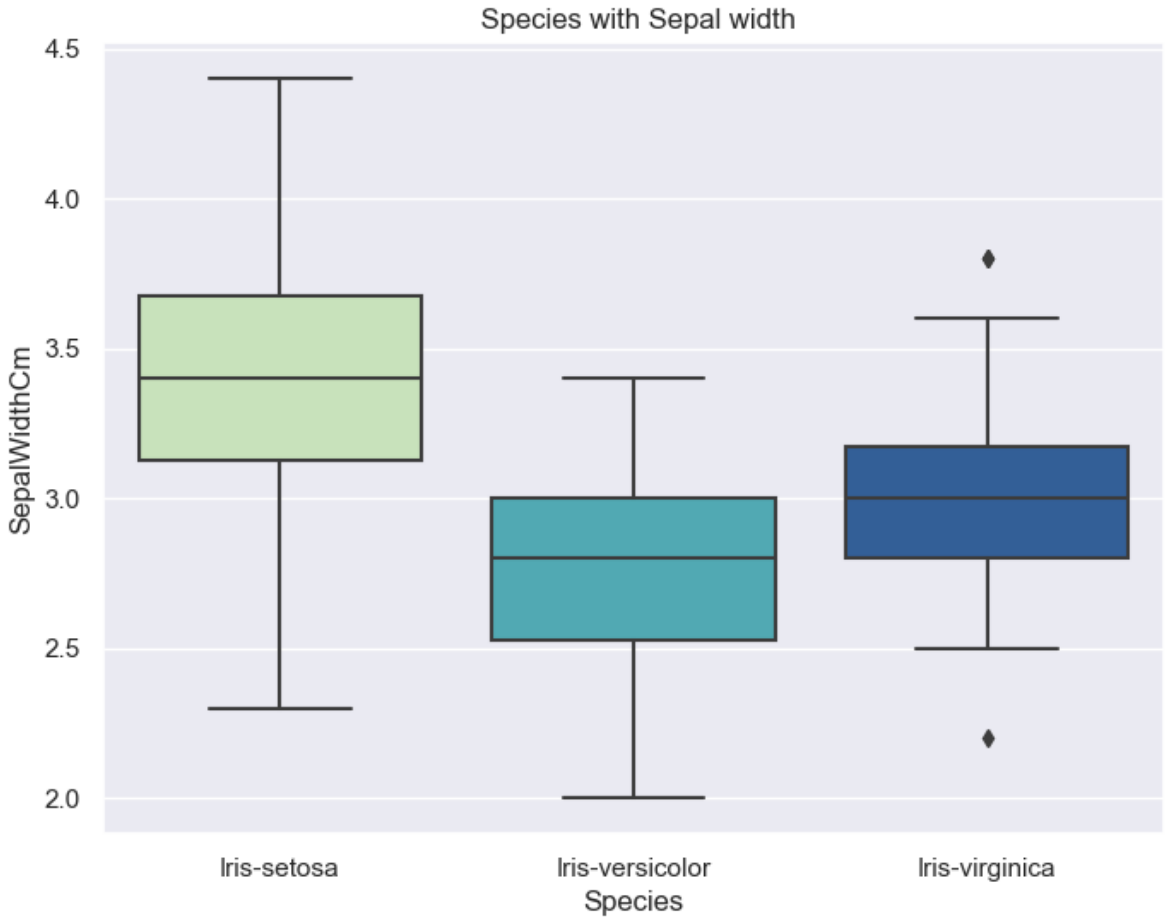
In [123]:

```
plt.figure(figsize=(8,6))
sns.countplot(x='Species',data=df, palette="husl")
plt.xlabel("Count of each Target class")
plt.ylabel("Target classes")
plt.show()
```



In [124]:

```
plt.figure(figsize=(8,6))
sns.boxplot(x='Species',y='SepalWidthCm',data=df ,palette='YlGnBu')
plt.title(' Species with Sepal width')
plt.show()
```

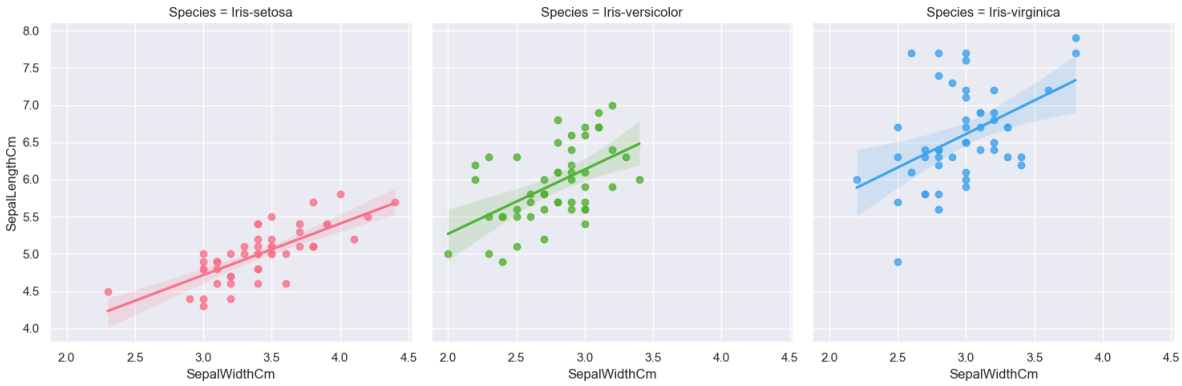


In [125]:

```
plt.figure(figsize=(8,6))
sns.violinplot(x='Species', y='PetalLengthCm', data=df, palette='OrRd')
plt.title(' Species with Petal length')
plt.show()
```

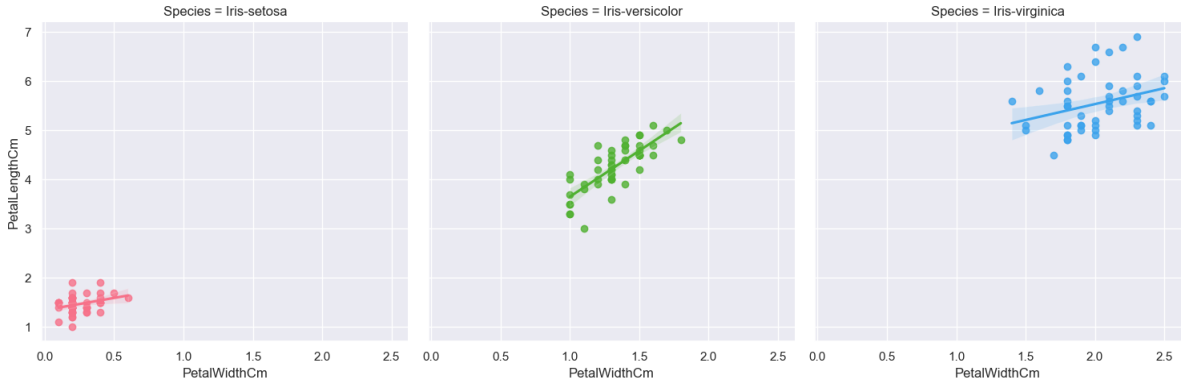
In [129]:

```
sns.lmplot(x = 'SepalWidthCm', y = 'SepalLengthCm', data = df, col = 'Species', hue = 'Speci
plt.show()
```



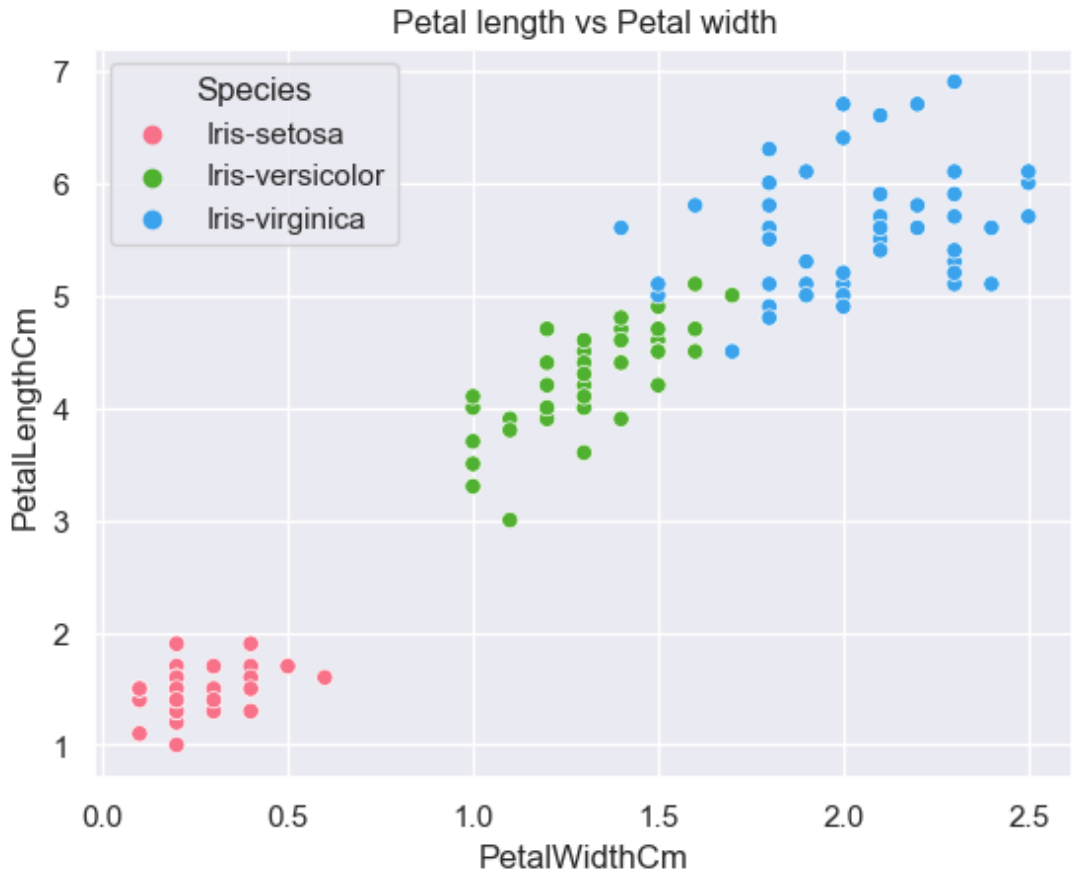
In [149]:

```
sns.lmplot(x = 'PetalWidthCm', y = 'PetalLengthCm', data = df, col = 'Species', hue = 'Speci
plt.show()
```



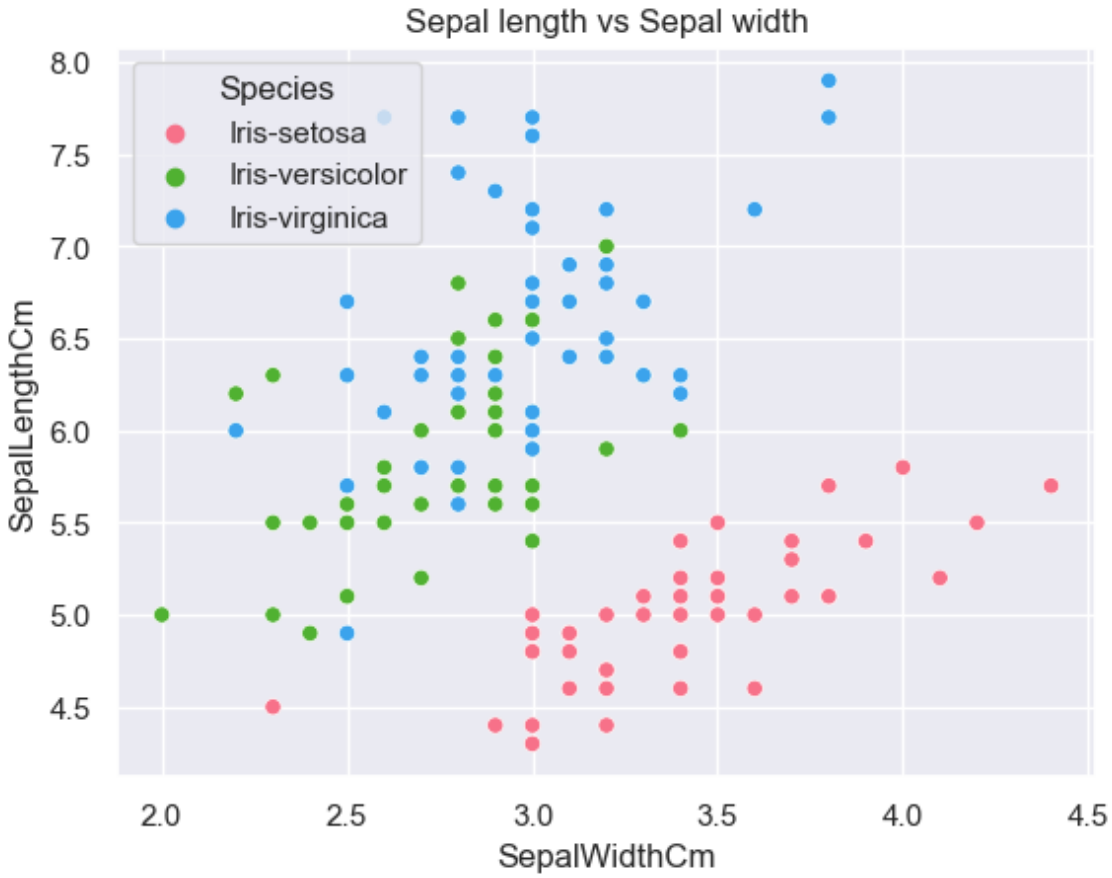
In [142]:

```
sns.scatterplot(data=df, x='PetalWidthCm', y='PetalLengthCm', hue = 'Species', palette= 'hus
plt.title('Petal length vs Petal width')
plt.show()
```



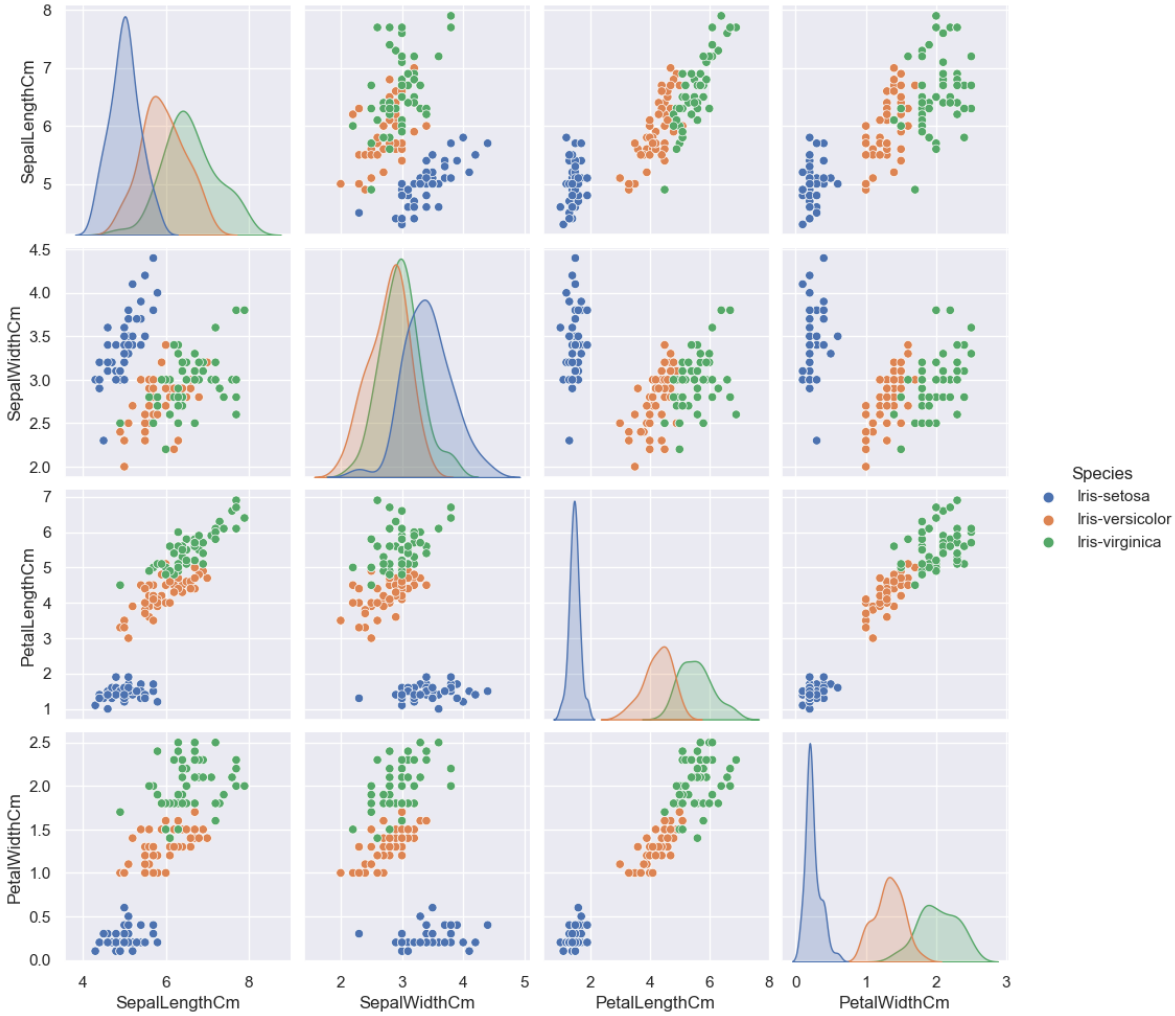
In [148]:

```
sns.scatterplot(data=df, x='SepalWidthCm', y='SepalLengthCm', hue = 'Species', palette= 'hus
plt.title('Sepal length vs Sepal width')
plt.show()
```



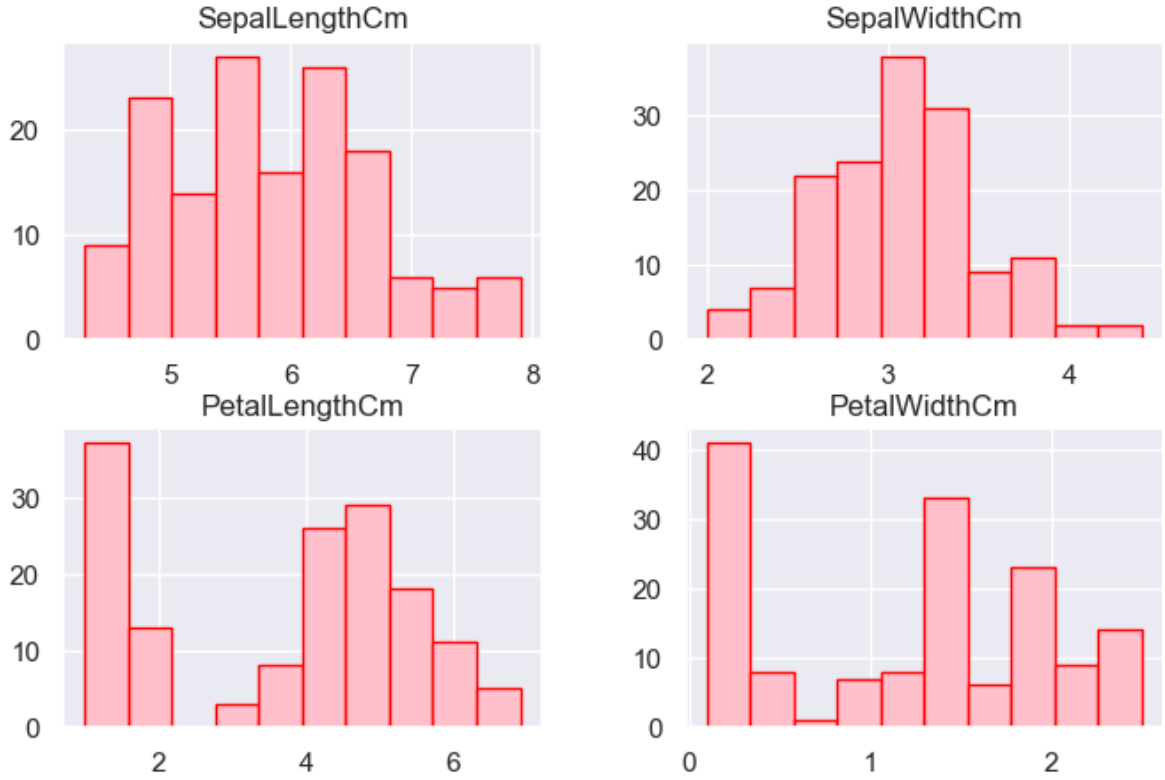
In [150]:

```
sns.pairplot(df, hue='Species')
plt.show()
```



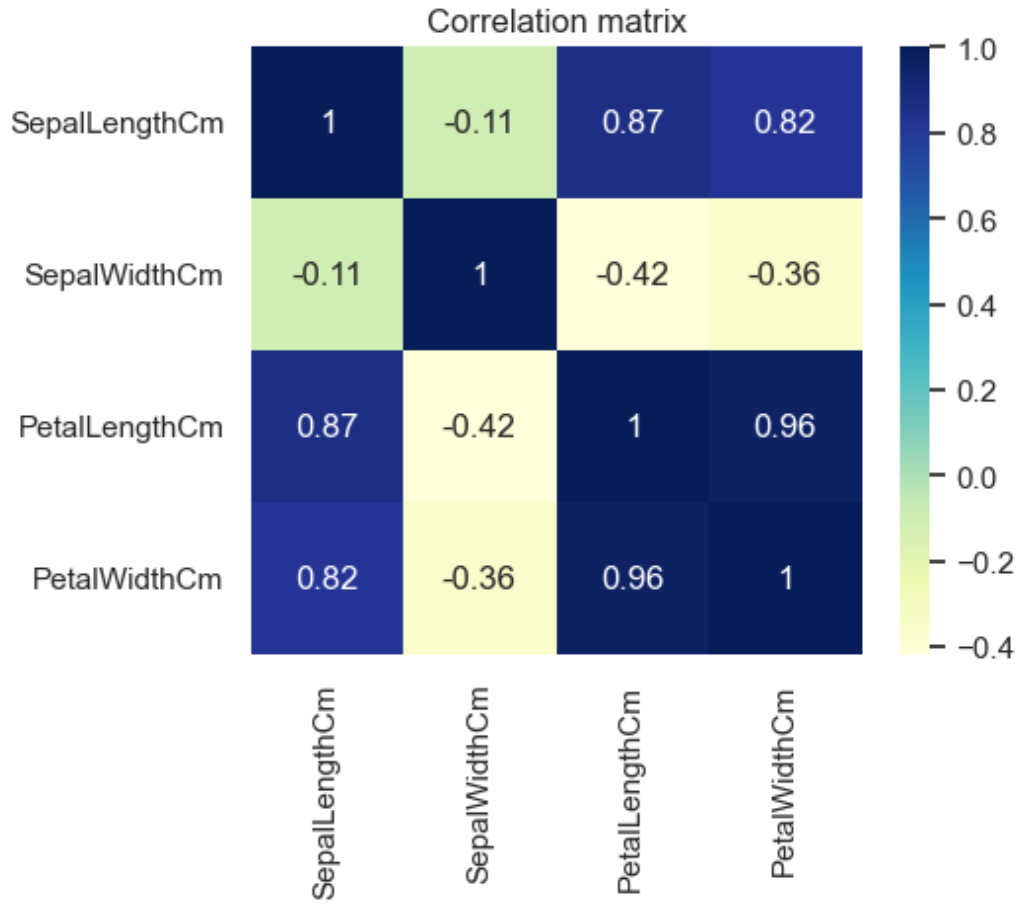
In [154]:

```
df.hist(bins=10, figsize=(8,5),color='pink', edgecolor='red')
plt.show()
```



In [156]:

```
plt.figure(figsize=(5,4))
sns.heatmap(df.corr(),annot=True,cmap='YlGnBu')
plt.title('Correlation matrix')
plt.show()
```



Label encoding for Species variable

In [107]:

```
df['Species']=df['Species'].astype('category')
df['Species']=df['Species'].cat.codes
```

Split the data into x and y variables

In [26]:

```
x= df.drop(['Species'],axis=1)
y= df[['Species']]
```

In [27]:

```
x.head(2)
```

Out[27]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2

In [28]:

```
y.head(2)
```

Out[28]:

	Species
0	0
1	0

Feature scaling

In [29]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
x_scaled=pd.DataFrame(sc.fit_transform(x),columns=x.columns)
```

In [30]:

```
x_scaled
```

Out[30]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
0	-0.900681	1.065722	-1.341272	-1.312977
1	-1.143017	-0.120170	-1.341272	-1.312977
2	-1.385353	0.354187	-1.398138	-1.312977
3	-1.506521	0.117008	-1.284407	-1.312977
4	-1.021849	1.302901	-1.341272	-1.312977
...	...	...	...	...
145	1.038005	-0.120170	0.819624	1.447956
146	0.553333	-1.306063	0.705893	0.922064
147	0.795669	-0.120170	0.819624	1.053537
148	0.432165	0.828544	0.933356	1.447956
149	0.068662	-0.120170	0.762759	0.790591

150 rows × 4 columns

In [31]:

```
# Check balance of data
df['Species'].value_counts()
```

Out[31]:

```
0    50
1    50
2    50
Name: Species, dtype: int64
```

Split the data into train and test

In [42]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.20, random_state=101)
```

Building KNN Classifier Model without PCA

In [166]:

```
# Model building with K point as 7
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=7)
knn.fit(x_train, y_train)
# Predict
y_pred_train_knn = knn.predict(x_train)
y_pred_test_knn = knn.predict(x_test)
# Evaluate
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score,precisio
accuracy_knn_test=accuracy_score(y_test,y_pred_test_knn)
accuracy_knn_train=accuracy_score(y_train,y_pred_train_knn)
print('K nearest neighbor - Train accuracy:', accuracy_score(y_train, y_pred_train_knn))
print('-------'*10)
print('K nearest neighbor - Test accuracy:', accuracy_score(y_test, y_pred_test_knn))
```

```
K nearest neighbor - Train accuracy: 0.9666666666666667
----------------------------------------------------------------------
K nearest neighbor - Test accuracy: 1.0
```

Cross validation

In [167]:

```
from sklearn.model_selection import cross_val_score
train_accuracy_knn = cross_val_score(knn,x_train, y_train, cv=10)
crossval_train_knn=train_accuracy_knn.mean()
test_accuracy_knn = cross_val_score(knn,x_test, y_test, cv=10)
crossval_test_knn=test_accuracy_knn.mean()
print('K Nearest Neighbor after Cross validation Train accuracy:', crossval_train_knn)
print('---------------'*5)
print('K Nearest Neighbor after Cross validation Test accuracy:', crossval_test_knn)
```

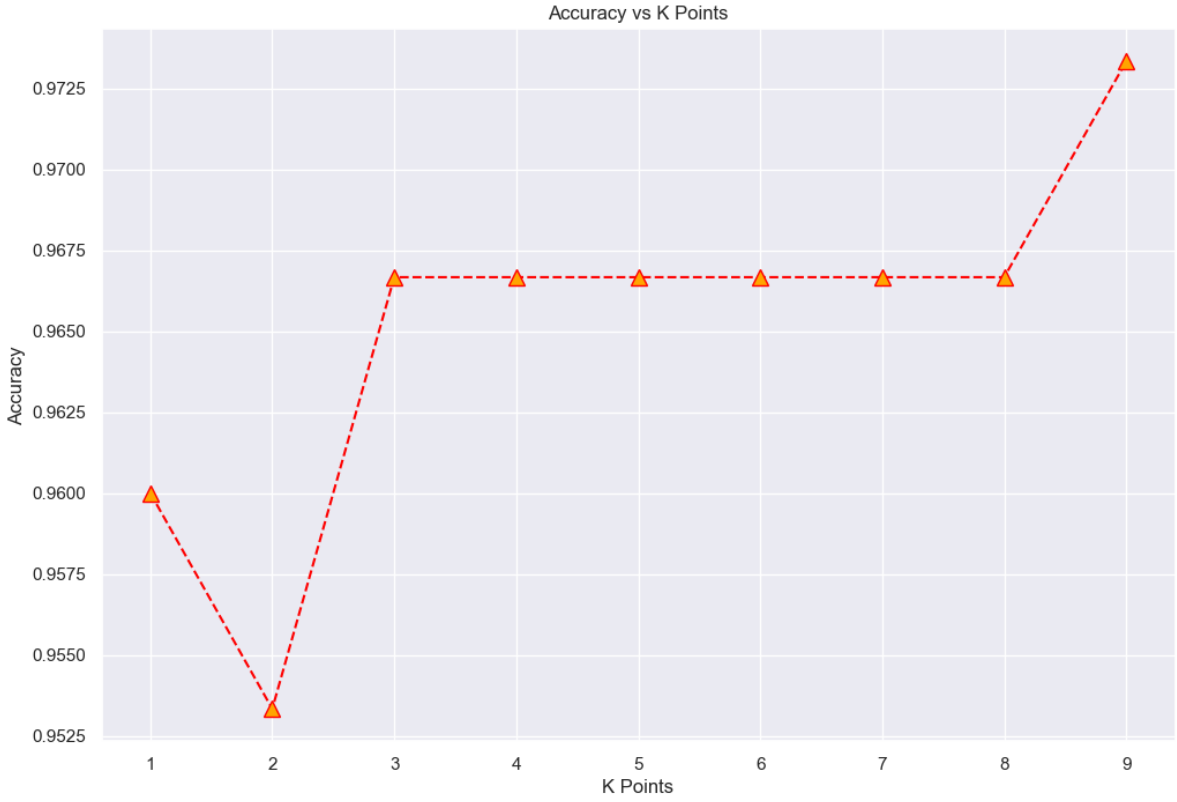
```
K Nearest Neighbor after Cross validation Train accuracy: 0.9583333333333333
---------------------------------------------------------------------------
K Nearest Neighbor after Cross validation Test accuracy: 0.9666666666666666
```

In [168]:

```
accuracy=[]
for i in range(1,10):
    knn = KNeighborsClassifier(n_neighbors=i)
    score=cross_val_score(knn,x,y,cv=10)
    accuracy.append(score.mean())
```

In [169]:

```
plt.figure(figsize=(12,8))
plt.plot(range(1,10), accuracy, color='red', linestyle='dashed', marker='^',
         markerfacecolor='orange', markersize=10)
plt.title('Accuracy vs K Points')
plt.xlabel('K Points')
plt.ylabel('Accuracy')
plt.show()
```



Building KNN Classifier Model with PCA

In [170]:

```
from sklearn.decomposition import PCA
pca = PCA(n_components=None)
X = pca.fit_transform(x)
```

In [171]:

```
print('PCA applied:')
print(pd.DataFrame(X).head())
print()
print('Original DataPoint:')
print(pd.DataFrame(x).head())
```

```
PCA applied:
          0         1         2         3
0 -2.683934  0.333896 -0.009183 -0.004268
1 -2.716273 -0.157627 -0.200812  0.098874
2 -2.890297 -0.130422  0.027831  0.017030
3 -2.747131 -0.305354  0.036506 -0.076924
4 -2.728111  0.338392  0.108397 -0.069099

Original DataPoint:
   SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
0            5.1           3.5            1.4           0.2
1            4.9           3.0            1.4           0.2
2            4.7           3.2            1.3           0.2
3            4.6           3.1            1.5           0.2
4            5.0           3.6            1.4           0.2
```

Co variance

In [172]:

```
pca.get_covariance()
```

Out[172]:

```
array([[ 0.68569351, -0.03865324,  1.27368233,  0.5169038 ],
       [-0.03865324,  0.17895928, -0.31336152, -0.11479776],
       [ 1.27368233, -0.31336152,  3.11317942,  1.29638747],
       [ 0.5169038 , -0.11479776,  1.29638747,  0.58241432]])
```

Explained variance

In [173]:

```
explained_variance = pca.explained_variance_ratio_
explained_variance
```

Out[173]:

```
array([0.92613385, 0.05166261, 0.01701985, 0.00518368])
```

PCA with 2 components

In [174]:

```
pca=PCA(n_components=2)
X_2=pca.fit_transform(X)
```

In [175]:

```
x_train, x_test, y_train, y_test = train_test_split(X_2, y, test_size=0.2, random_state=101)
```

In [176]:

```
knn_pca = KNeighborsClassifier(n_neighbors=7)
knn_pca.fit(x_train,y_train)
#Evaluate
print("Train score after PCA 2",knn_pca.score(x_train,y_train),"%")
print("Test score after PCA 2",knn_pca.score(x_test,y_test),"%")
```

```
Train score after PCA 2 0.9666666666666667 %
Test score after PCA 2 1.0 %
```

PCA with 3 components

In [179]:

```
pca=PCA(n_components=3)
X_3=pca.fit_transform(X)
```

In [180]:

```
x_train, x_test, y_train, y_test = train_test_split(X_3, y, test_size=0.2, random_state=101)
```

In [181]:

```
knn_pca = KNeighborsClassifier(n_neighbors=7)
knn_pca.fit(x_train,y_train)
#Evaluate
print("Train score after PCA 3",knn_pca.score(x_train,y_train),"%")
print("Test score after PCA 3",knn_pca.score(x_test,y_test),"%")
```

```
Train score after PCA 3 0.9666666666666667 %
Test score after PCA 3 1.0 %
```

PCA with 4 components

In [182]:

```
pca=PCA(n_components=4)
X_4=pca.fit_transform(X)
```

In [183]:

```
x_train, x_test, y_train, y_test = train_test_split(X_4, y, test_size=0.2, random_state=101)
```

In [184]:

```
# Model building with K point as 3
knn_pca = KNeighborsClassifier(n_neighbors=7)
knn_pca.fit(x_train,y_train)
#Evaluate
print("Train score after PCA 4",knn_pca.score(x_train,y_train),"%")
print("Test score after PCA 4",knn_pca.score(x_test,y_test),"%")
```

```
Train score after PCA 4 0.9666666666666667 %
Test score after PCA 4 1.0 %
```

Conclusion

- Before Cross validation Train accuracy is 95% & Test accuracy is 100%
- After Cross validation Train accuracy is 95% & Test accuracy is 96%
- At PCA 2, 3 & 4 Train accuracy is 96% & Test accuracy 100%