iris-AhmedMosaad-20191020

May 26, 2023

1 IRIS Clustering and Classification

In this notebook I will use Iris dataset to cluster diffrent iris types and then using KNN to compare thr value from clustering

Name: Ahmed Mosaad Gelwan Student ID: 20191020

1.1 Import Reqired Libaraies

[3]:
```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```python
plt.style.use('seaborn-v0_8-deep')
```

1.2 Import Data and Explor it

[5]:
```python
df = pd.read_csv('3_iris.csv', header=None)
df.head()
```

[5]:

	0	1	2	3	4
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
```

Hence this data has not the header so we searched about it and find the identical data set and labeled as: - `sepal_length`: sepal length in cm - `sepal_width`: sepal width in cm - `petal_length`: petal length in cm - `petal_width`: petal width in cm - `class`: class value of the row

[6]:
```
df.columns =␣
 ↪['sepal_length','sepal_width',        'petal_length',        'petal_width','class']
```

[7]:
```
df.shape
```

[7]:
```
(150, 5)
```

[8]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   class         150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

All Data Types are correct.

[9]:
```
df.describe().T
```

[9]:

	count	mean	std	min	25%	50%	75%	max
sepal_length	150.0	5.843333	0.828066	4.3	5.1	5.80	6.4	7.9
sepal_width	150.0	3.054000	0.433594	2.0	2.8	3.00	3.3	4.4
petal_length	150.0	3.758667	1.764420	1.0	1.6	4.35	5.1	6.9
petal_width	150.0	1.198667	0.763161	0.1	0.3	1.30	1.8	2.5

[10]:
```
df.isnull().sum()
```

[10]:
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
class           0
dtype: int64
```

There is no missing values

[11]:
```
df.duplicated().sum()
```

[11]: 3

There 3 duplicated rows so we will drop them

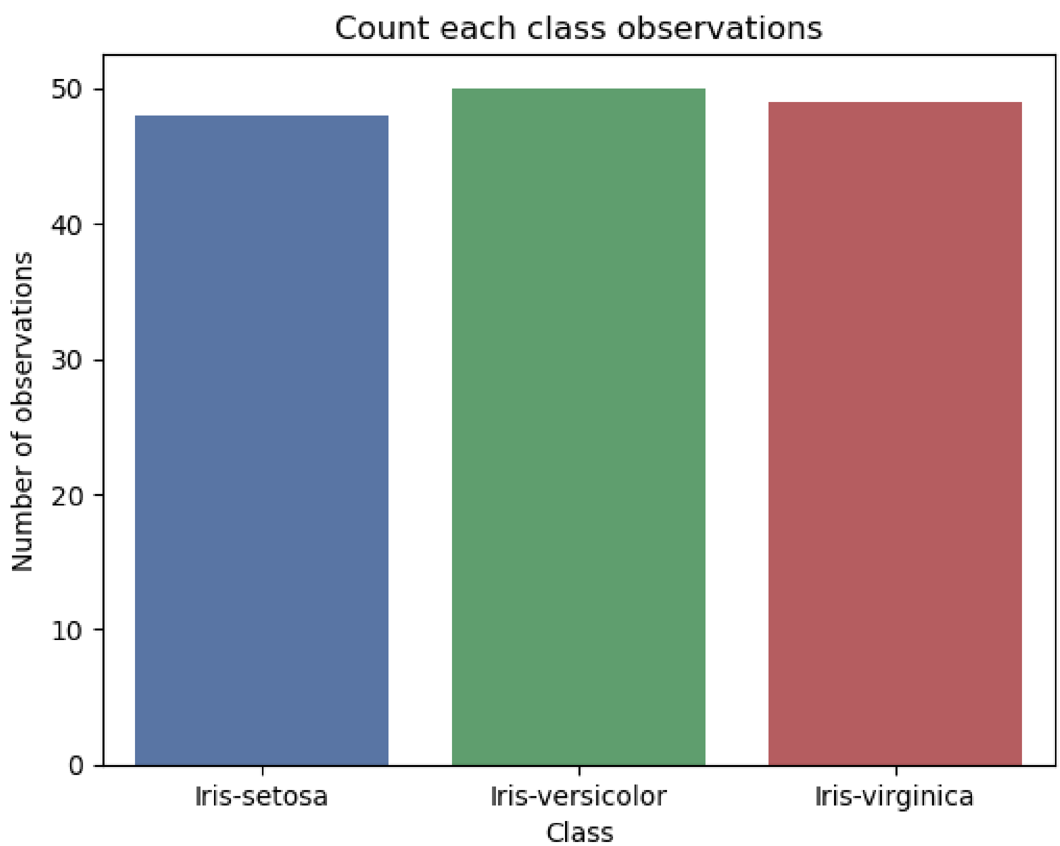
[12]:
```
df.drop_duplicates(inplace=True)
```

[13]:
```
df.shape
```

[13]: (147, 5)

To ensure that the dataset is blanced

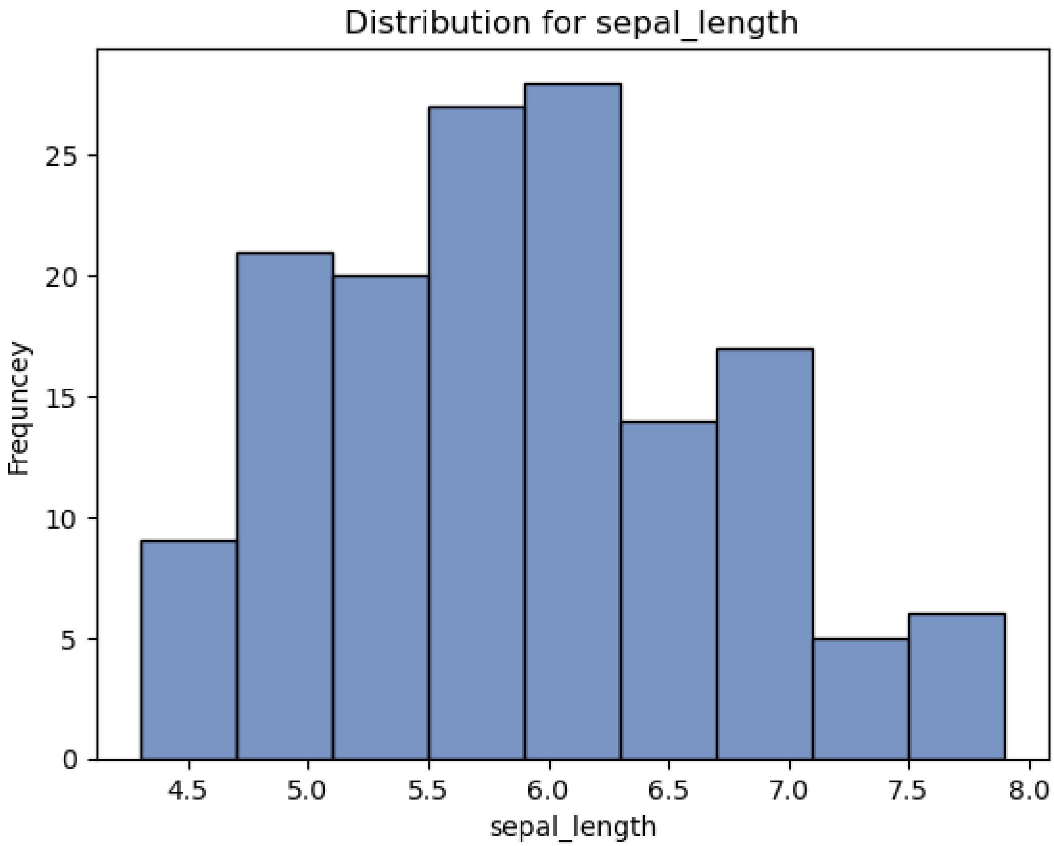
[14]:
```
sns.countplot(x=df['class'])
plt.xlabel('Class')
plt.ylabel('Number of observations')
plt.title('Count each class observations')
plt.show()
```

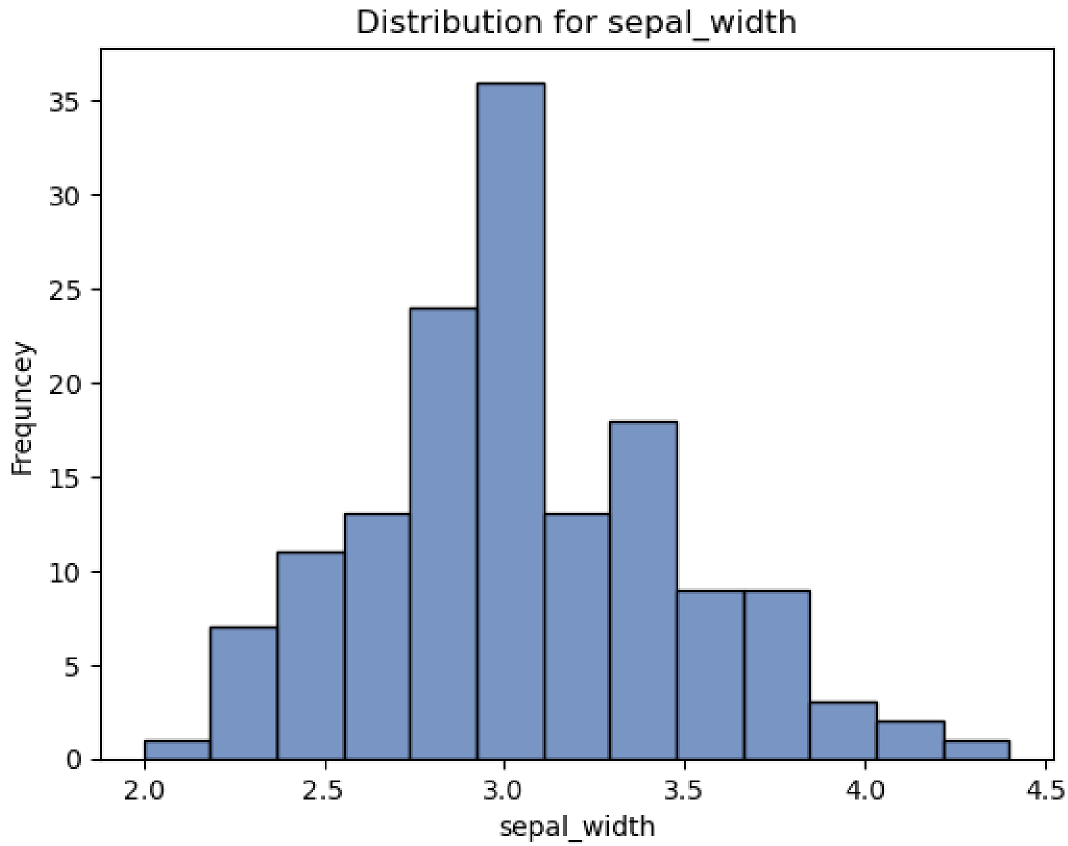


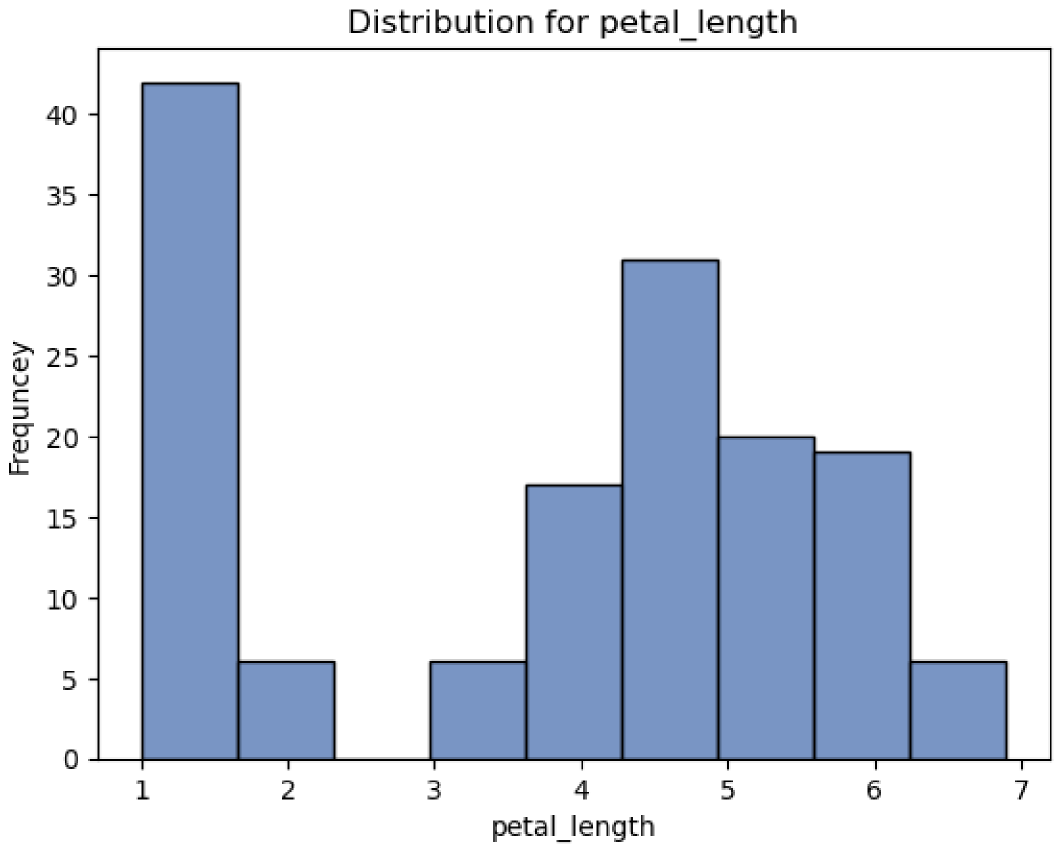
We find that this dataset is balenced. (:

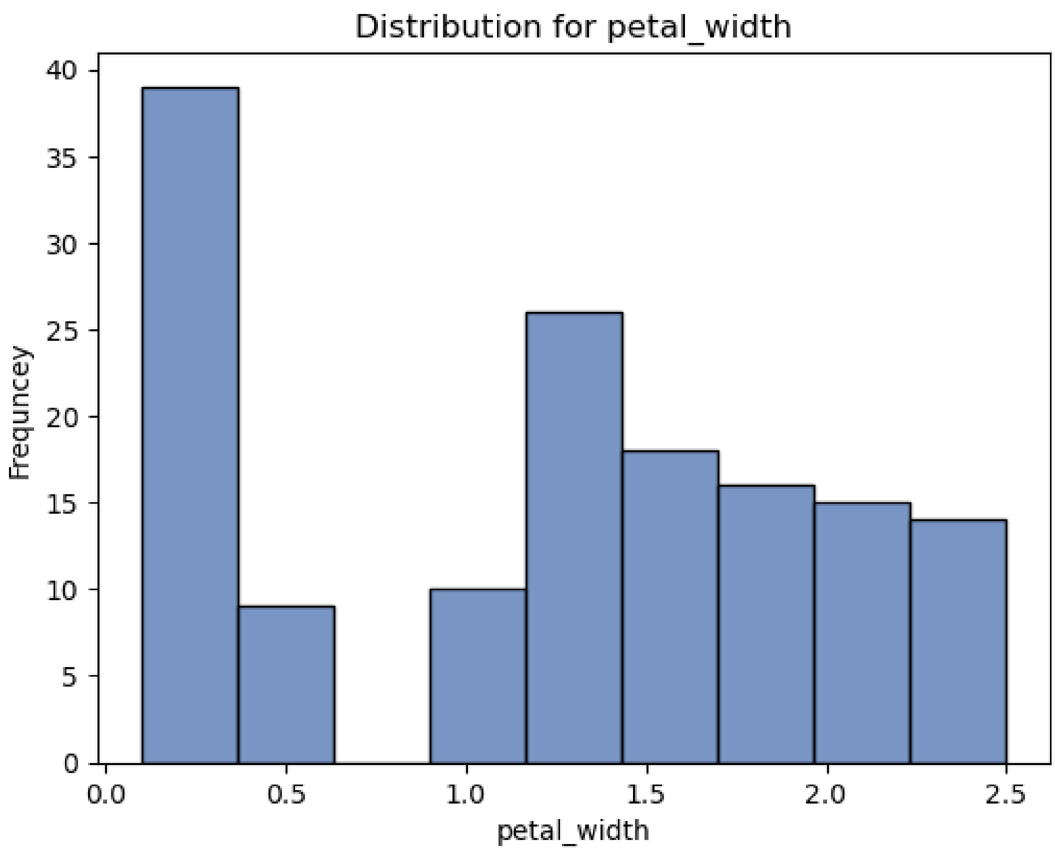
1.3 Distributions of Features

[15]:
```
for feature in df.drop(columns='class'):
    sns.histplot(data=df, x=feature)
    plt.xlabel(f'{feature}')
    plt.ylabel('Frequncey')
    plt.title(f'Distribution for {feature}')
    plt.show()
```




Distribution for sepal_width
Frequncey
35
30
25
20
15
10
5
0
2.0
2.5
3.0
3.5
4.0
4.5
sepal_width

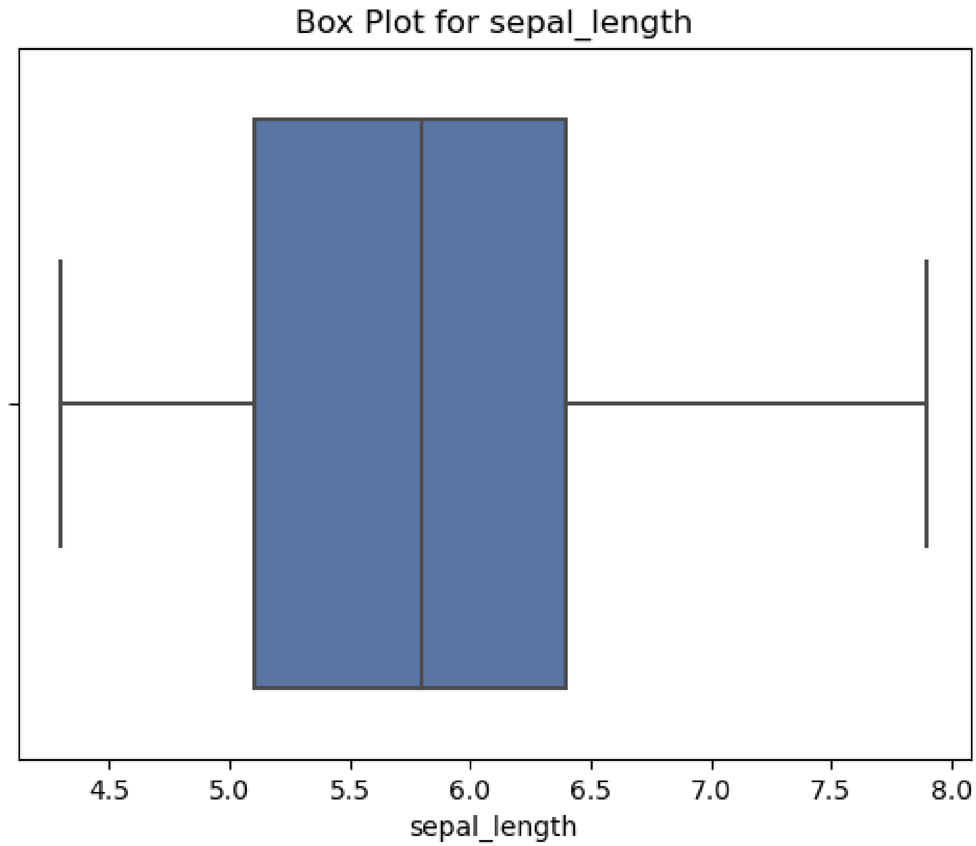

Distribution for petal_length
Frequncey
petal_length
0
5
10
15
20
25
30
35
40
1
2
3
4
5
6
7

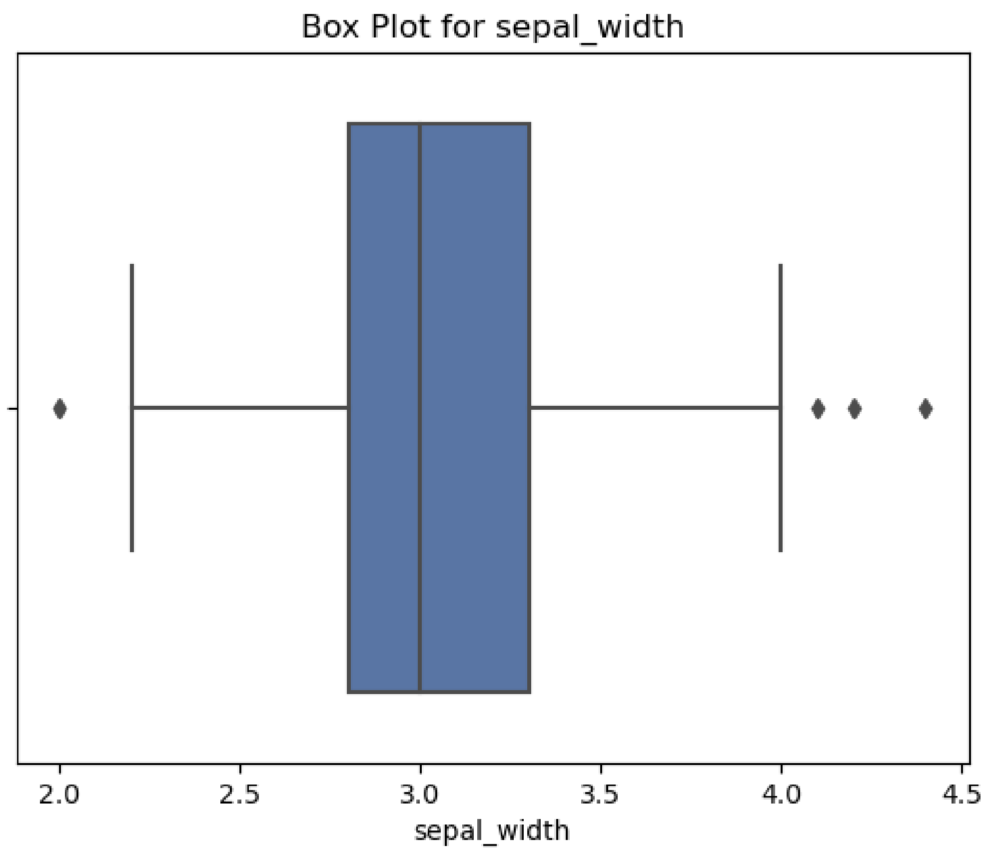


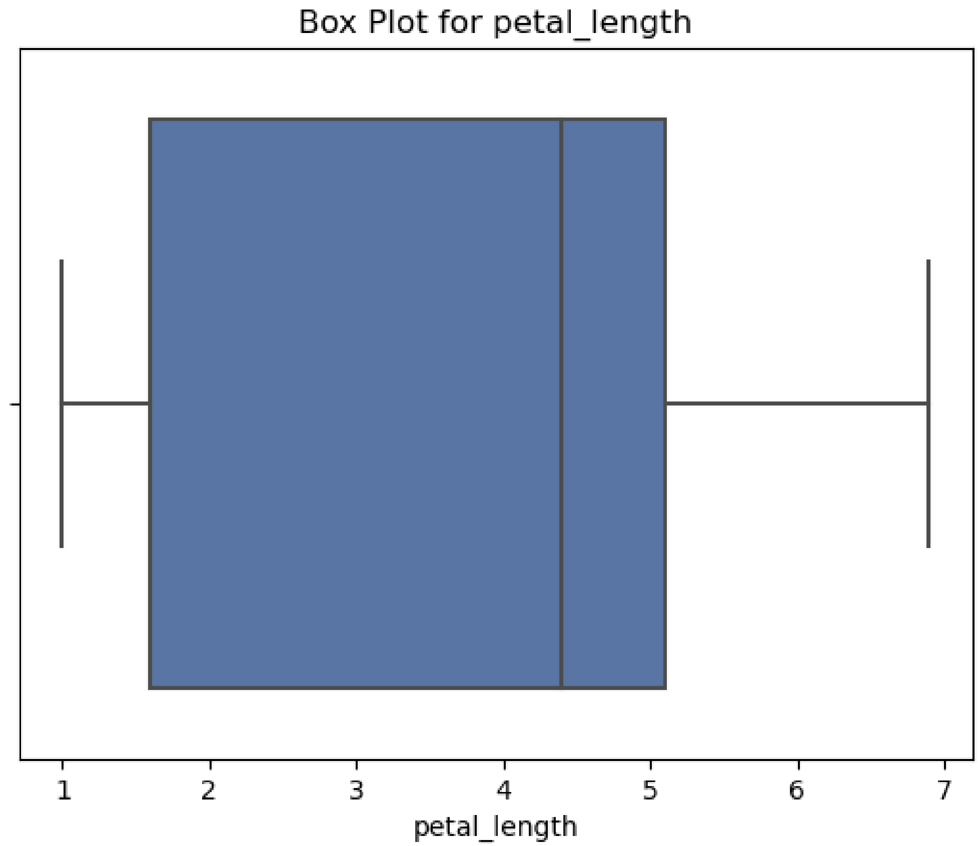
We can say that both `petal_length` and `petal_width` are divided into 2 groups i.e. they may be two classes has the near by values

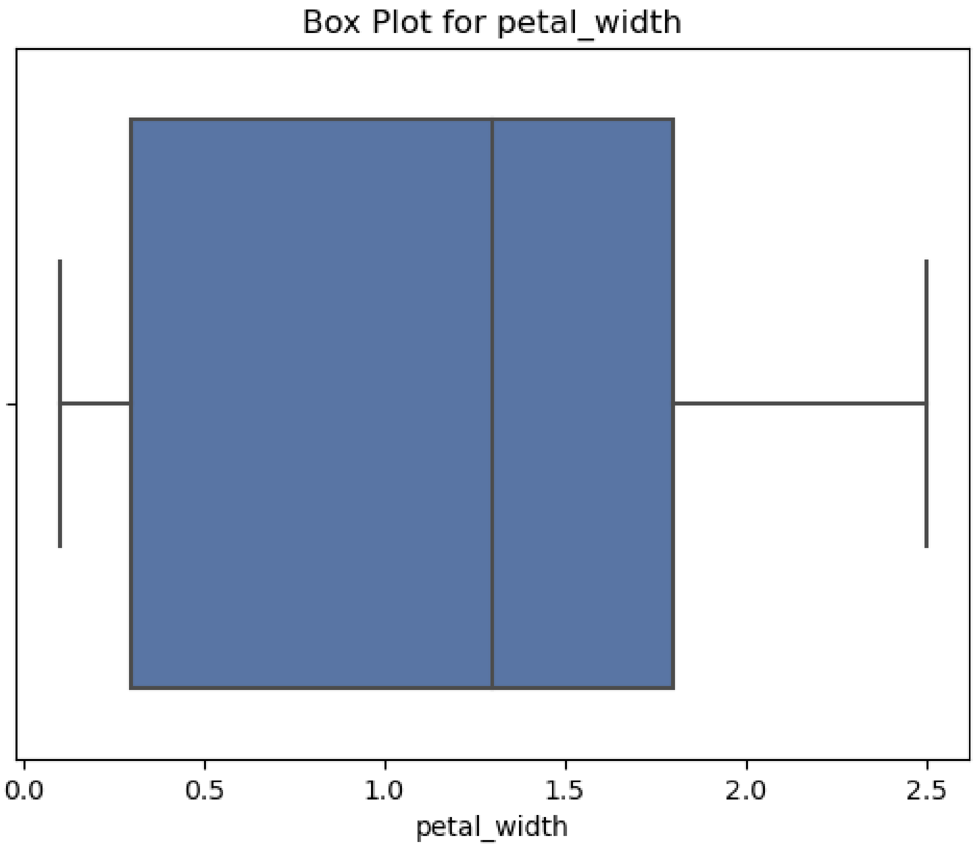
1.3.1 Outlier Anaysis

[16]:
```
for feature in df.drop(columns='class'):
    sns.boxplot(x=df[feature])
    # plt.xlabel(f'{feature}')
    # plt.ylabel('Frequncey')
    plt.title(f'Box Plot for {feature}')
    plt.show()
```


Box Plot for sepal_length
4.5
5.0
5.5
6.0
6.5
7.0
7.5
8.0
sepal_length


Box Plot for sepal_width
2.0
2.5
3.0
3.5
4.0
4.5
sepal_width


Box Plot for petal_length
1
2
3
4
5
6
7
petal_length



We can say that there is outliers in ***sepal_width*** *so we will remove them*

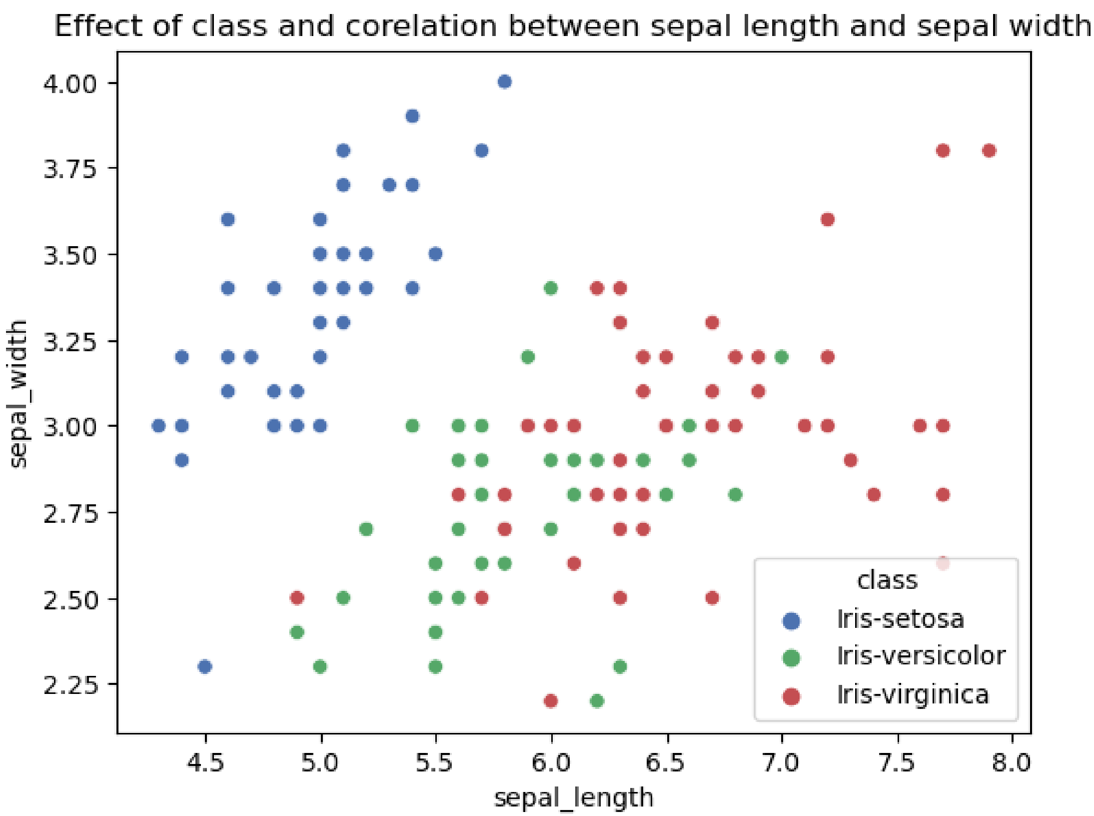
[17]:
```
lower , upper = df['sepal_width'].quantile([0.02,0.98]).to_list()
df = df[df['sepal_width'].between(lower,upper)]
```

1.3.2 Is there a corelation bettwen variables in each class?

[18]:
```
df.columns
```

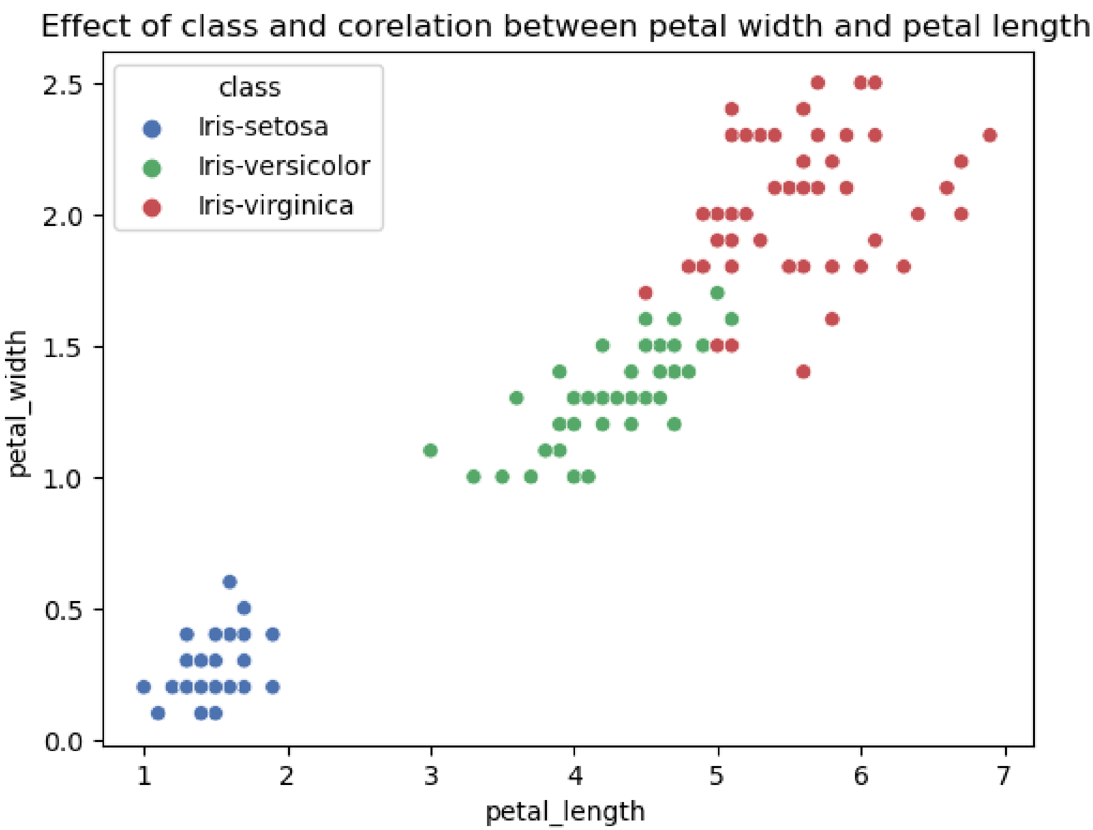
[18]:
```
Index(['sepal_length', 'sepal_width', 'petal_length', 'petal_width', 'class'],
dtype='object')
```

[19]:
```
sns.scatterplot(data=df, x='sepal_length',y='sepal_width', hue='class')
plt.title('Effect of class and corelation between sepal length and sepal␣
↪width');
```



We can deduce that `sepal_length` and `sepal_width` can not help us to cluster these data points becouse there is overlap between `versicolor` and `virhinica`

[20]:
```
sns.scatterplot(data=df, x='petal_length',y='petal_width', hue='class')
plt.title('Effect of class and corelation between petal width and petal␣
↪length');
```



We can deduce that `petal_length` and `petal_width` can help us to cluster these data points however ther is a tiny overlap `versicolor` and `virhinica`

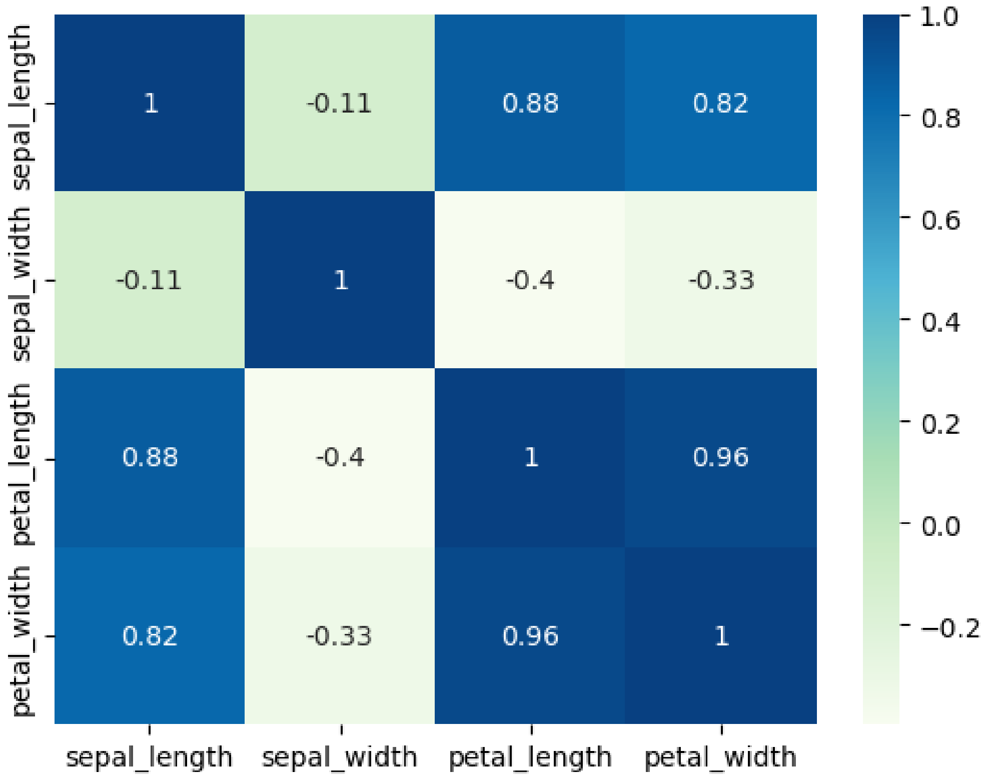
1.3.3 Corelation between fearures

[21]:
```
corealtion = df.drop(columns='class').corr()
corealtion
```

[21]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.113268	0.879015	0.821715
sepal_width	-0.113268	1.000000	-0.396539	-0.328102
petal_length	0.879015	-0.396539	1.000000	0.960785
petal_width	0.821715	-0.328102	0.960785	1.000000

[22]:
```
sns.heatmap(corealtion, annot=True, cmap='GnBu');
```



Hene `spepal_length` *and* `petal_length` *have the strong corealtionship we will drop* `spepal_length` *becouse it will not help us clustering the data*

[23]:
```
df.drop(columns=['sepal_length'], inplace=True)
```

1.4 Splitting Data

[24]:
```
target = 'class'
X = df.drop(columns=target)
y = df[target]
```

1.5 Model Building

1.5.1 K-Means

Choosing K: We will choose K=3 because will we cluster into 3 grups

[25]:
```
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.preprocessing import MinMaxScaler
from sklearn.cluster import KMeans
```

[26]:
```
k_means = make_pipeline(
    MinMaxScaler(),
    KMeans(n_clusters=3, random_state=42)
)
```

[27]:
```
k_means.fit(X)
```

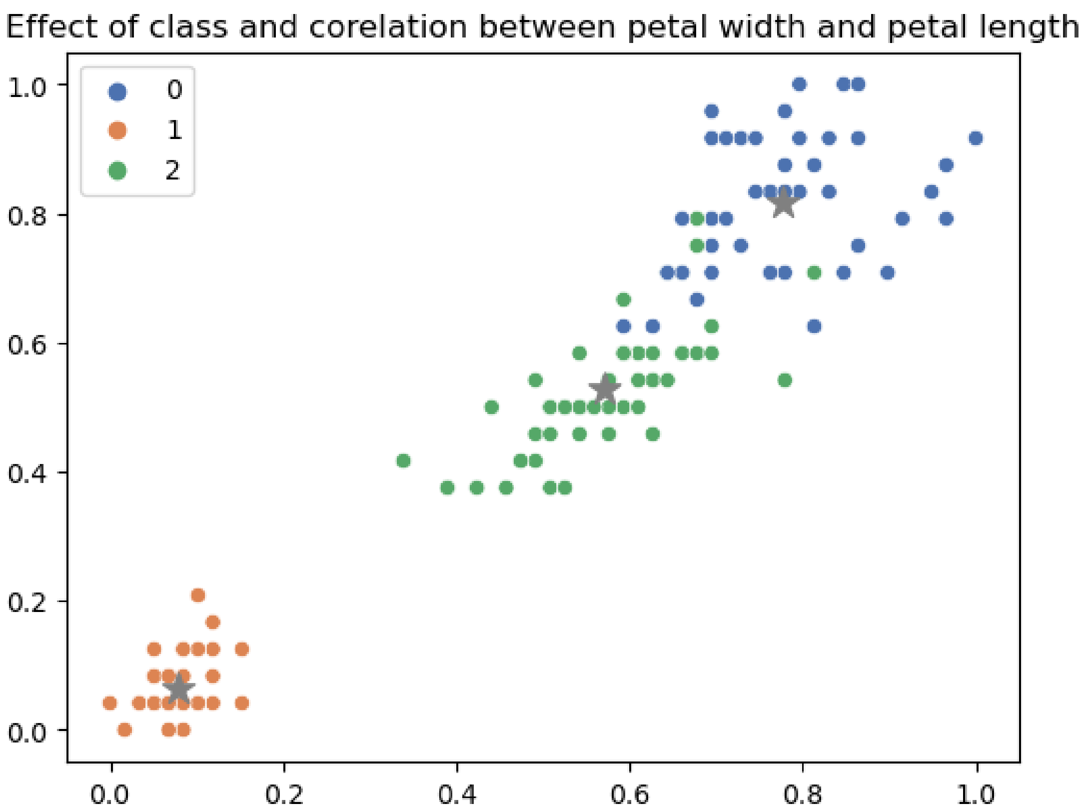
```
/home/ahmed/anaconda3/lib/python3.10/site-
packages/sklearn/cluster/_kmeans.py:870: FutureWarning: The default value of
`n_init` will change from 10 to 'auto' in 1.4. Set the value of `n_init`
explicitly to suppress the warning
  warnings.warn(
```

[27]:
```
Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                ('kmeans', KMeans(n_clusters=3, random_state=42))])
```

[28]:
```
labels = k_means.named_steps['kmeans'].labels_
centroids = k_means.named_steps['kmeans'].cluster_centers_
```

Now lets plot the above classified plot using petal width and petal length and include centroids

[29]:
```
X_transformed = k_means.named_steps['minmaxscaler'].fit_transform(X)
sns.scatterplot(x=X_transformed[:,1],y=X_transformed[:,2], hue=labels,␣
 ↪palette='deep')
plt.scatter(x=centroids[:,1], y=centroids[:,2], color='gray',marker='*',s=150)
plt.title('Effect of class and corelation between petal width and petal␣
 ↪length');
```



Error in model

[30]:
```
k_means.named_steps['kmeans'].inertia_
```

[30]: 5.5775982963497

Okay it is not large

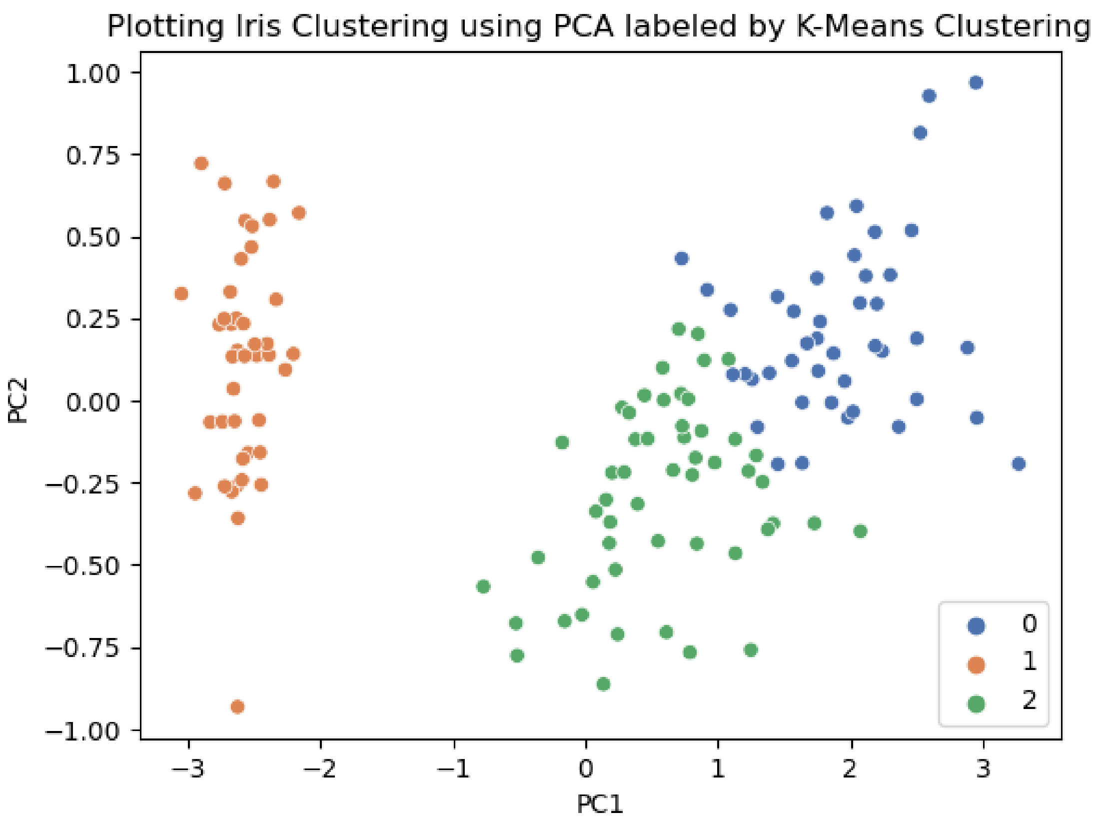
1.5.2 Plot all diminsions in 2D scatter plot using PCA

[34]:
```
from sklearn.decomposition import PCA

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

[37]:
```
sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=labels, palette='deep')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.title("Plotting Iris Clustering using PCA labeled by K-Means Clustering")
```

[37]: Text(0.5, 1.0, 'Plotting Iris Clustering using PCA labeled by K-Means Clustering')



1.5.3 Building KNN Model

[39]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
encoder = LabelEncoder()
y = encoder.fit_transform(y)
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, ConfusionMatrixDisplay,␣
 ↪classification_report
```

[40]:
```
encoder = LabelEncoder()
y = encoder.fit_transform(y)
```

[41]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2,␣
 ↪random_state=42, stratify=y)
```

[42]:
```
knn = make_pipeline(
    MinMaxScaler(),
    KNeighborsClassifier(n_neighbors=3)
)
```

[43]:
```
knn.fit(X_train, y_train)
```

[43]:
```
Pipeline(steps=[('minmaxscaler', MinMaxScaler()),
                ('kneighborsclassifier', KNeighborsClassifier(n_neighbors=3))])
```

[44]:
```
y_train_pred = knn.predict(X_train)
y_train_pred[:5]
```

[44]:
```
array([1, 0, 2, 0, 0])
```

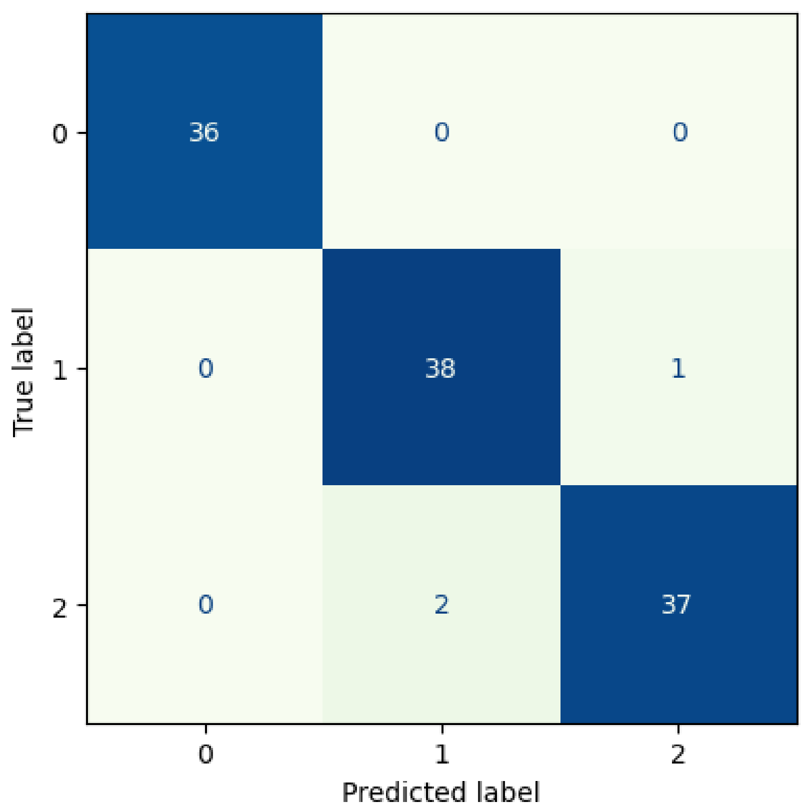
Model Evaliuation

[46]:
```
accuracy_score(y_train, y_train_pred)
```

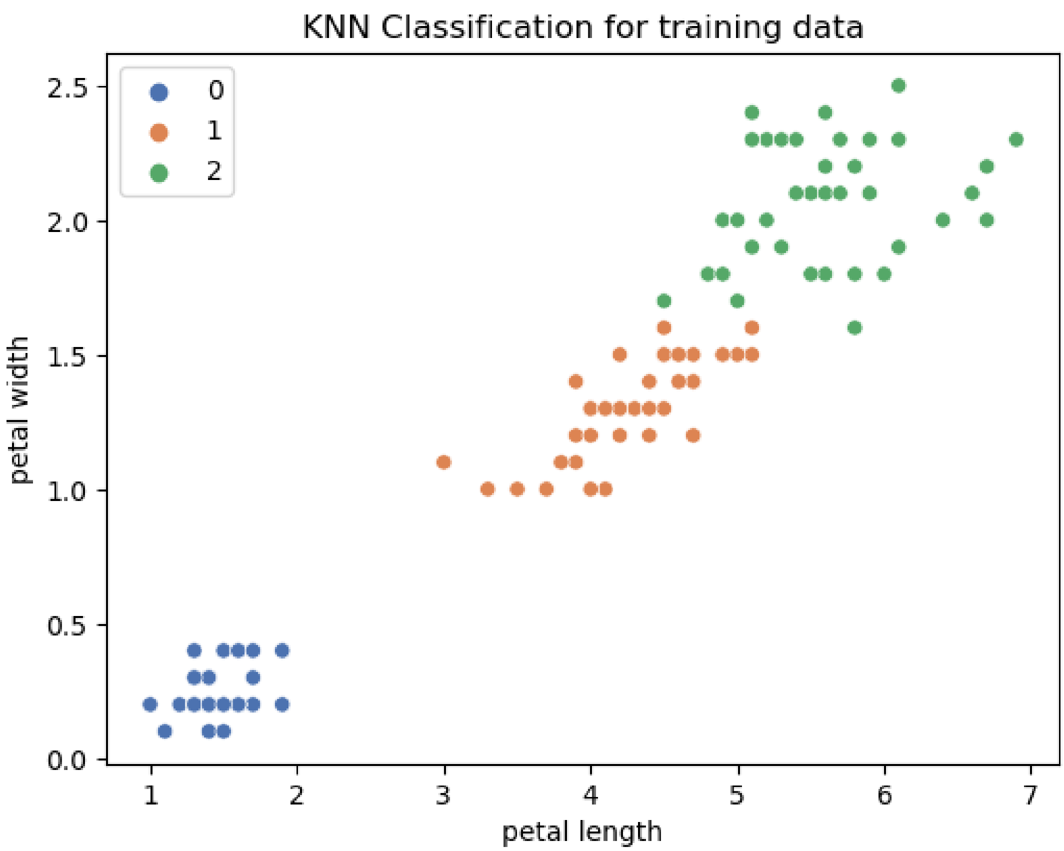
[46]:
```
0.9736842105263158
```

[47]:
```
ConfusionMatrixDisplay.from_estimator(knn,X_train, y_train, colorbar=False,␣
↪cmap='GnBu')
```

[47]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x7f7b2e474f70>
```



[48]:
```
# Create a scatter plot of the data points colored by labels
sns.scatterplot(x=X_train['petal_length'],y=X_train['petal_width'],␣
 ↪hue=y_train_pred, palette='deep')
plt.xlabel('petal length')
plt.ylabel('petal width')
plt.title('KNN Classification for training data')
plt.show()
```

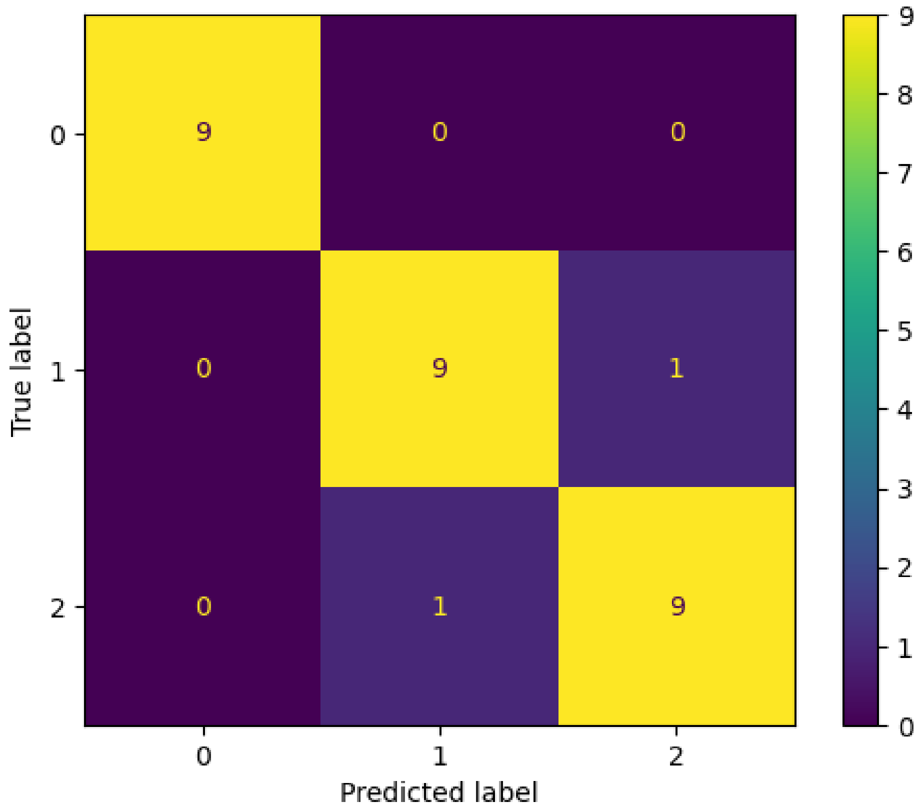


Plot hole dataset using PCA labeled by KNN

Make predictions

[49]:
```
y_test_pred = knn.predict(X_test)
```

[50]:
```
ConfusionMatrixDisplay.from_predictions(y_test,y_test_pred);
```



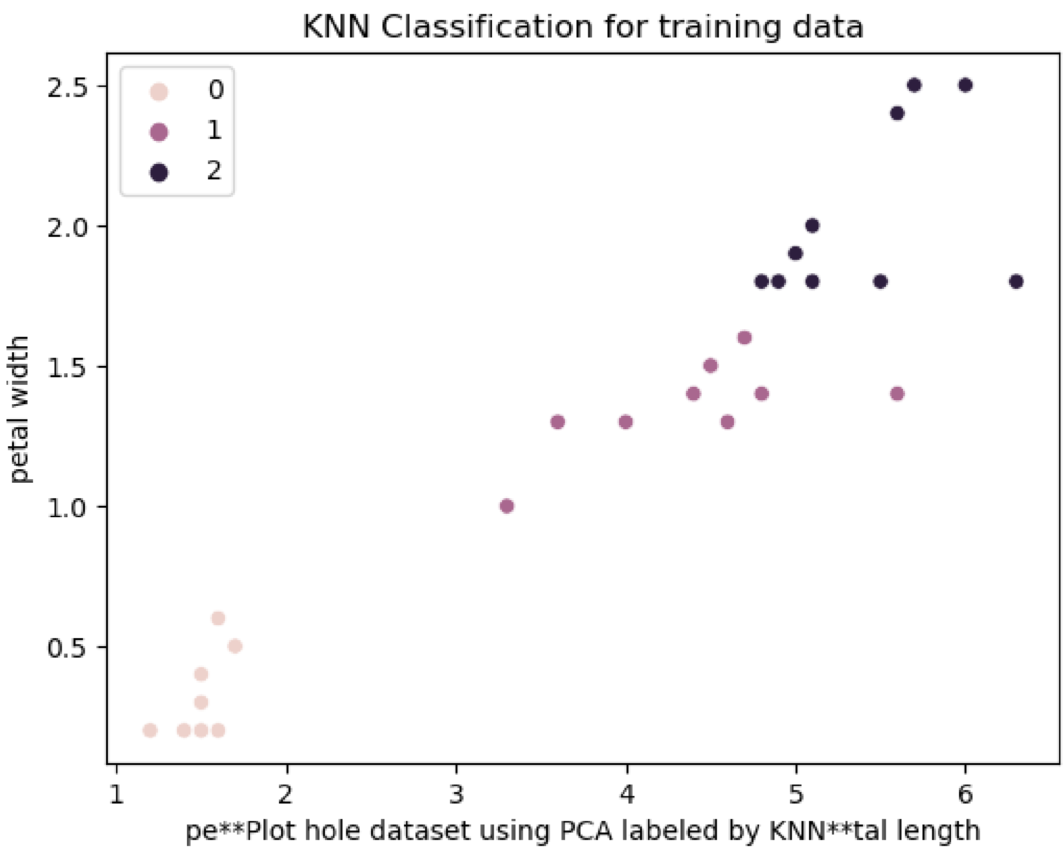
[52]:
```
print(classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         9
           1       0.90      0.90      0.90        10
           2       0.90      0.90      0.90        10

    accuracy                           0.93        29
   macro avg       0.93      0.93      0.93        29
weighted avg       0.93      0.93      0.93        29
```

[53]:
```
# Create a scatter plot of the data points colored by labels for testing data
sns.scatterplot(x=X_test['petal_length'],␣
 ↪y=X_test['petal_width'],hue=y_test_pred)
plt.xlabel('pe**Plot hole dataset using PCA labeled by KNN**tal length')
plt.ylabel('petal width')
plt.title('KNN Classification for training data')
plt.show()
```



1.6 Saving Trained Models

[173]:
```
import pickle

# Save the K-means model
with open('k_means.pkl', 'wb') as kmeans_file:
    pickle.dump(k_means, kmeans_file)

# Save the KNN model
with open('knn.pkl', 'wb') as knn_file:
    pickle.dump(knn, knn_file)
```