In [122…
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 21,8
import warnings
warnings.simplefilter('ignore')
%matplotlib inline
```

In [123…
```
#Alogrithms to train ML module
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
```

In [124…
```
#module requires for Machine Learning
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

In [125…
```
iris_DF = pd.read_csv('iris.csv')
```

In [126…
```
iris_DF.head(10)
```

Out[126]:

	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
5	6	5.4	3.9	1.7	0.4	Iris-setosa
6	7	4.6	3.4	1.4	0.3	Iris-setosa
7	8	5.0	3.4	1.5	0.2	Iris-setosa
8	9	4.4	2.9	1.4	0.2	Iris-setosa
9	10	4.9	3.1	1.5	0.1	Iris-setosa

In [127…
```
iris_DF.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 [128…
```
iris_DF.describe()
```

Out[128]:

	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 [129…
```
iris_DF.dtypes
```

Out[129]:
```
Id                 int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

In [130…
```
iris_DF.isnull().sum()
```

Out[130]:
```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

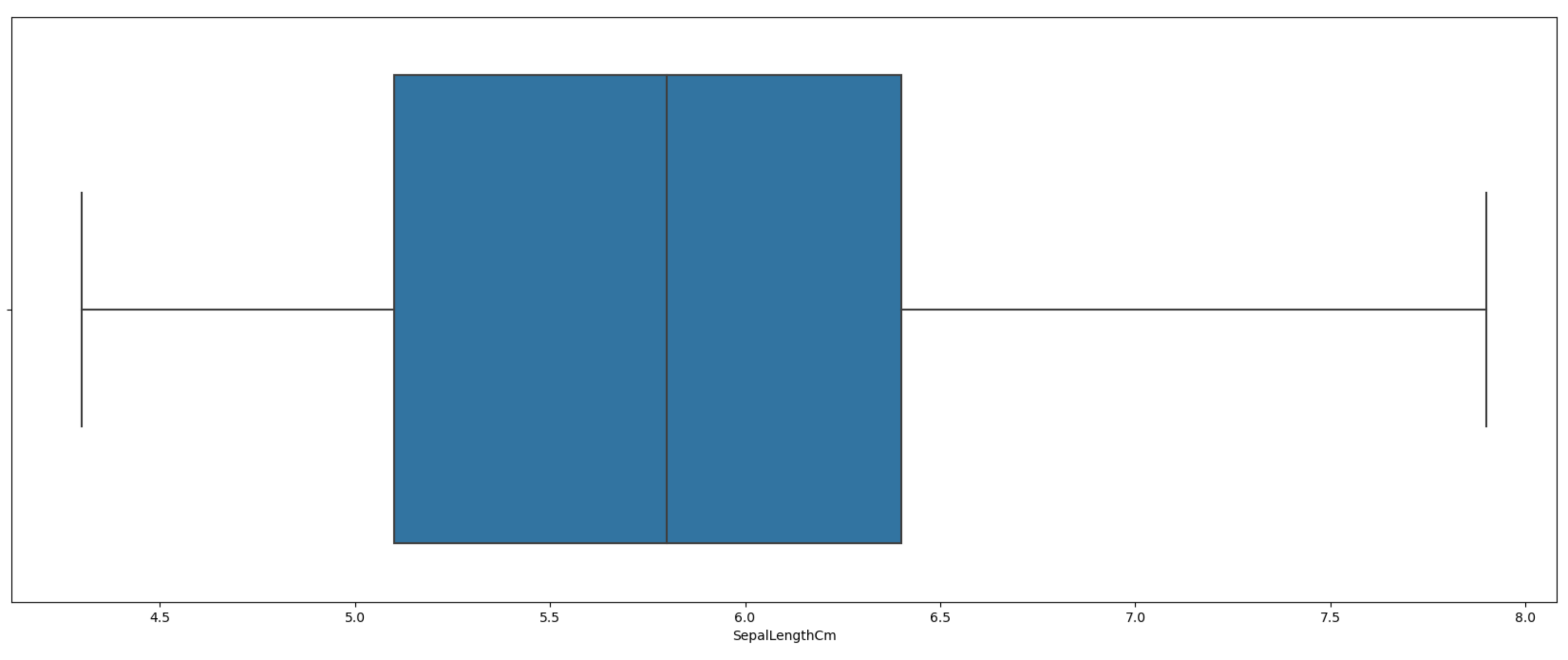
In [131…
```
{col : iris_DF[col].unique() for col in iris_DF if iris_DF[col].dtype == object}
```

Out[131]:
```
{'Species': array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)}
```

In [132…
```
iris_DF = iris_DF.astype( {'Species' : 'category'})
```

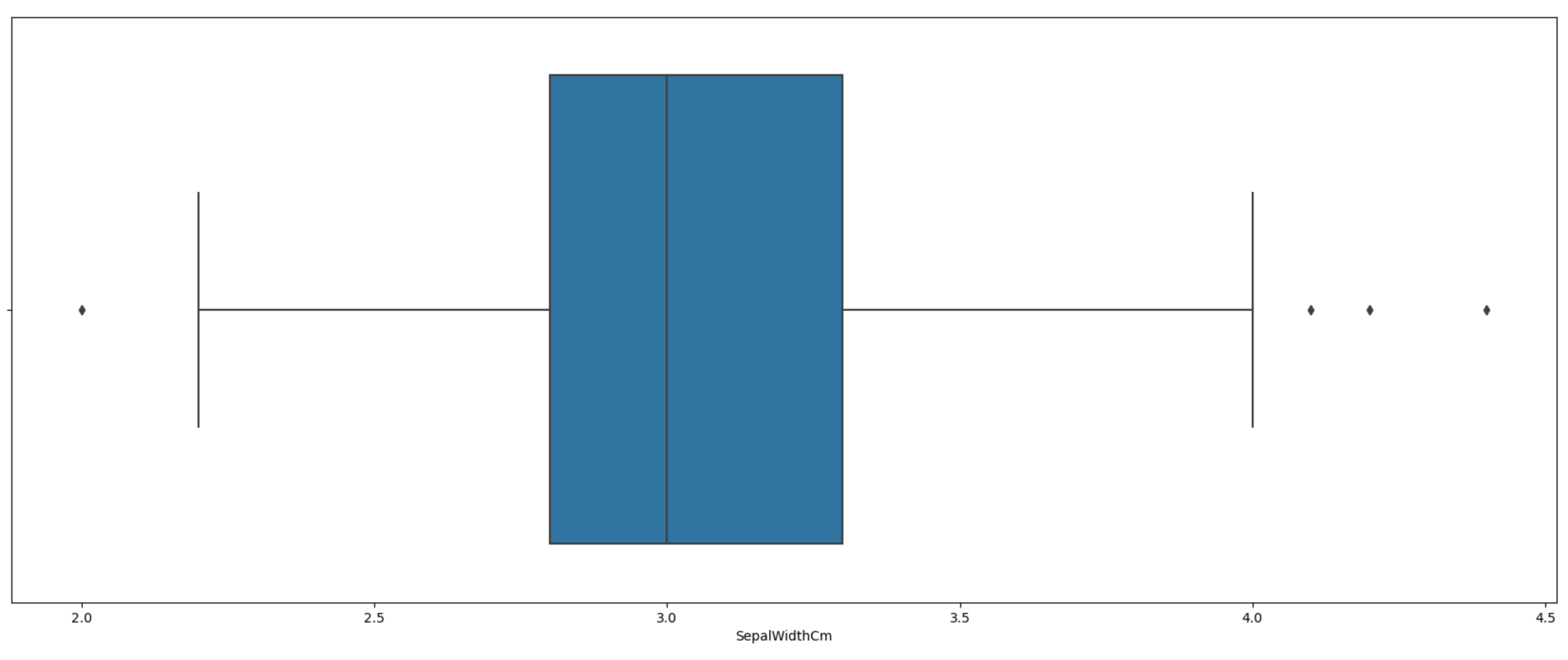
In [133…
```
sns.boxplot(iris_DF.SepalLengthCm)
```

Out[133]:
```
<AxesSubplot:xlabel='SepalLengthCm'>
```



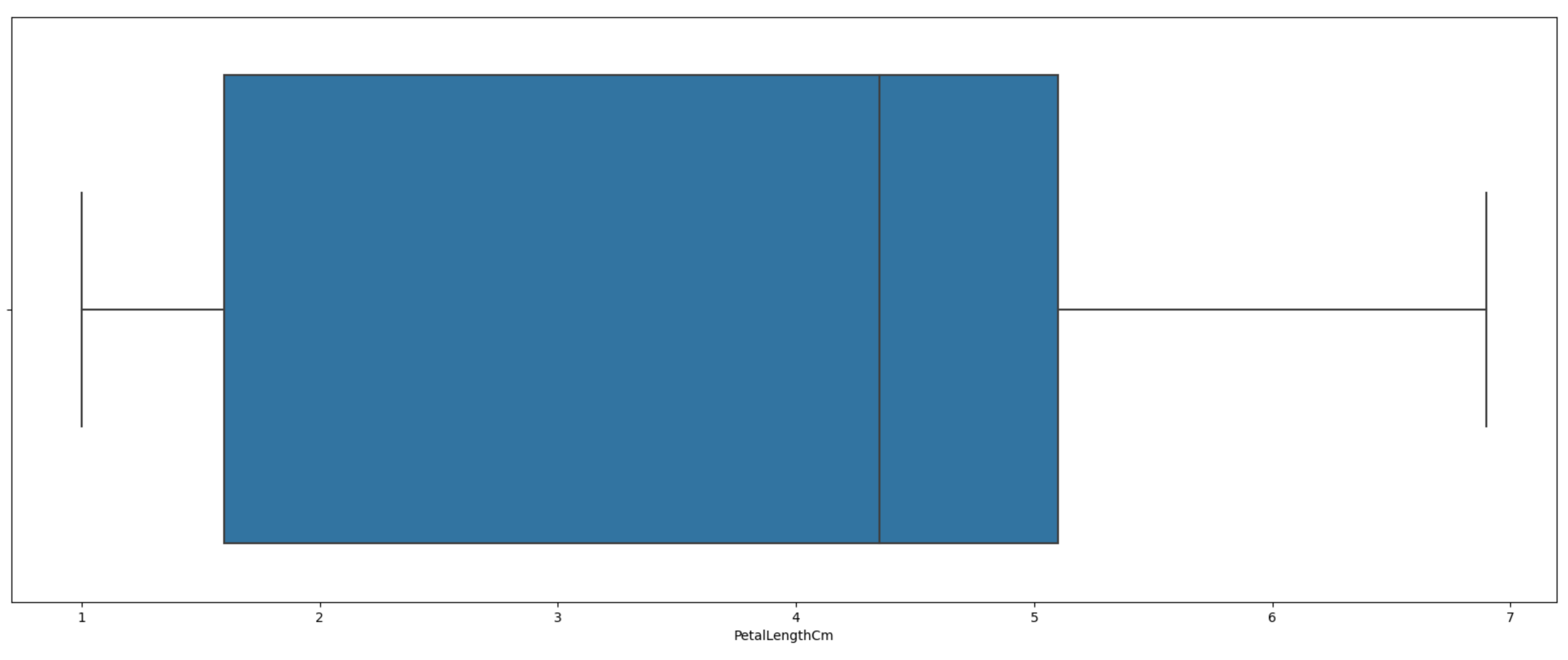
In [134…
```
sns.boxplot(iris_DF.SepalWidthCm)
```

Out[134]:
```
<AxesSubplot:xlabel='SepalWidthCm'>
```



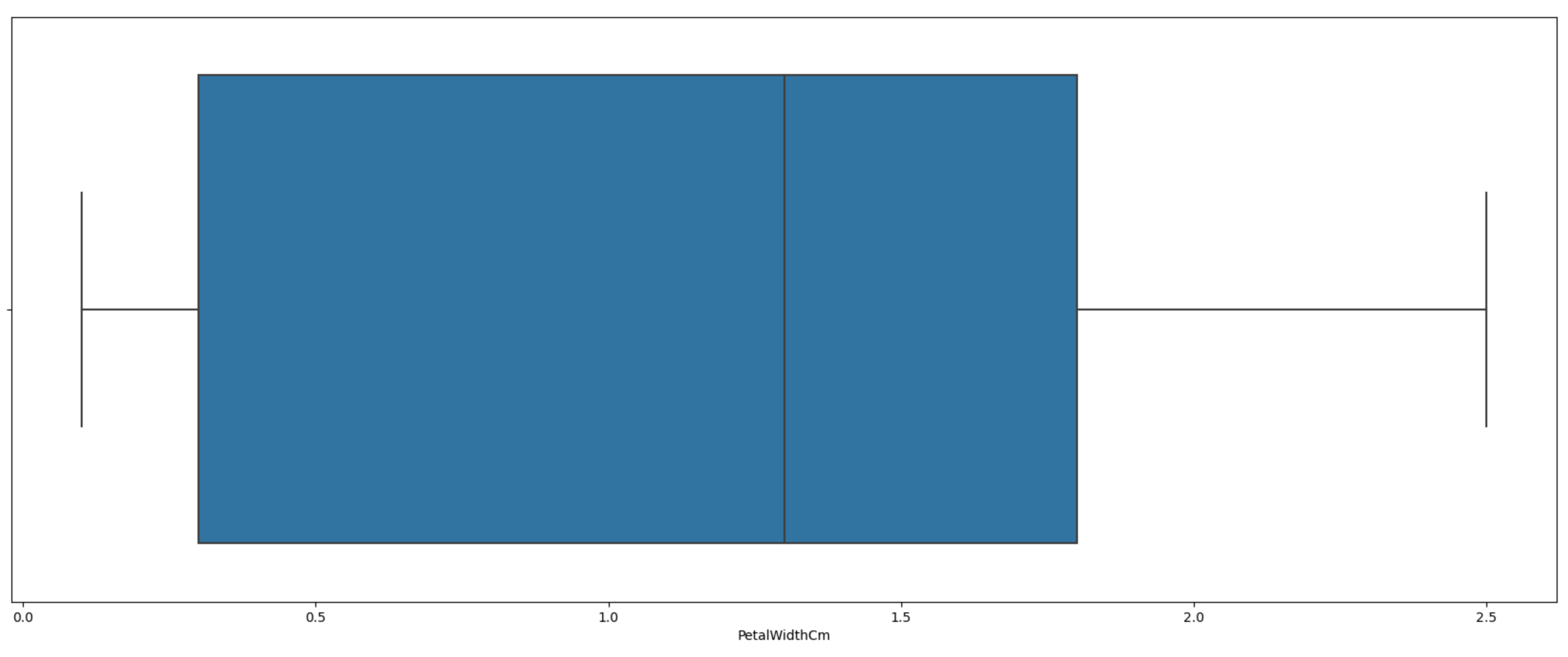
In [135…
```
sns.boxplot(iris_DF.PetalLengthCm)
```

Out[135]:
```
<AxesSubplot:xlabel='PetalLengthCm'>
```



In [136…
```
sns.boxplot(iris_DF.PetalWidthCm)
```

Out[136]:
```
<AxesSubplot:xlabel='PetalWidthCm'>
```



In [137…
```
iris_DF.Species.unique()
```

Out[137]:
```
['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
Categories (3, object): ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
```

In [138…
```
iris_DF.drop("Id",inplace = True , axis=1)
```

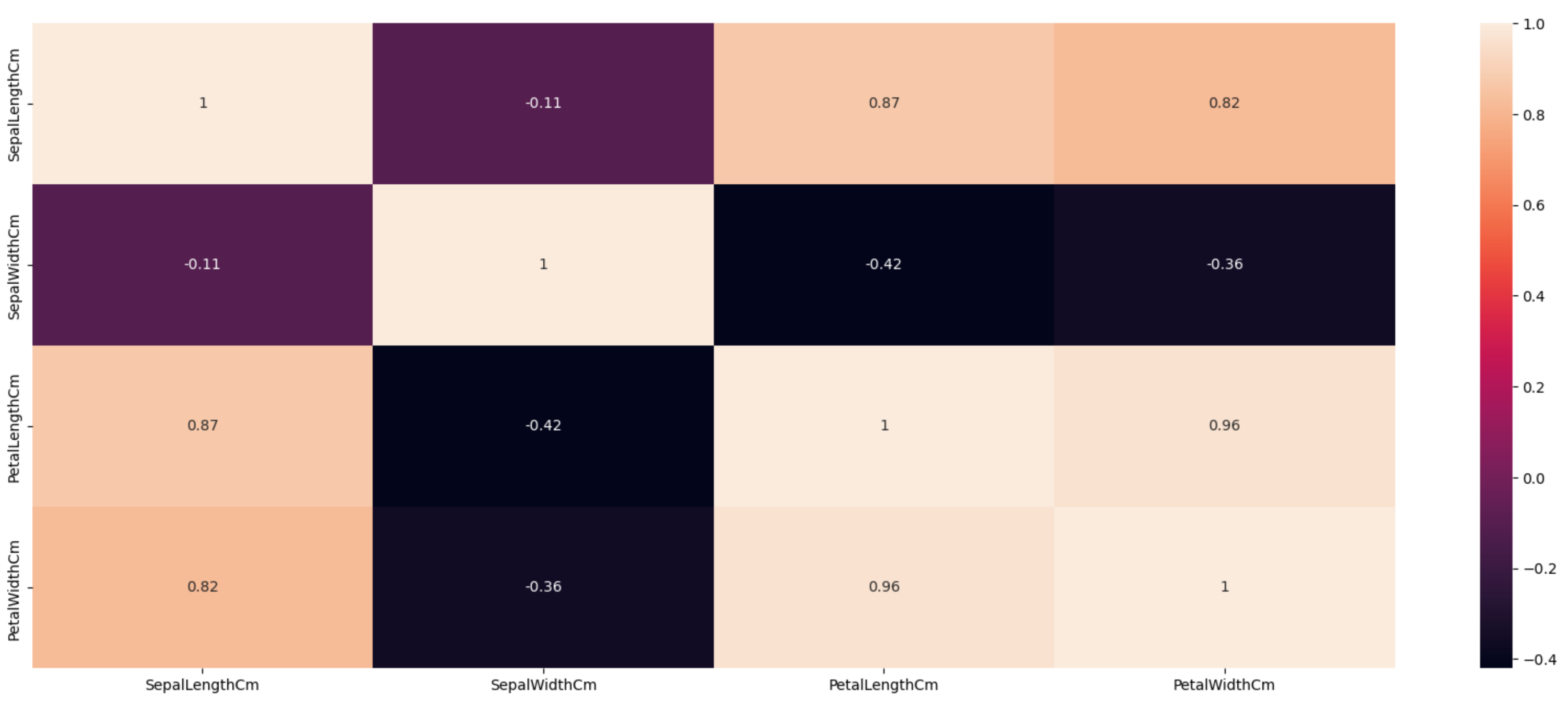
In [139…
```
iris_DF.head(10)
```

Out[139]:

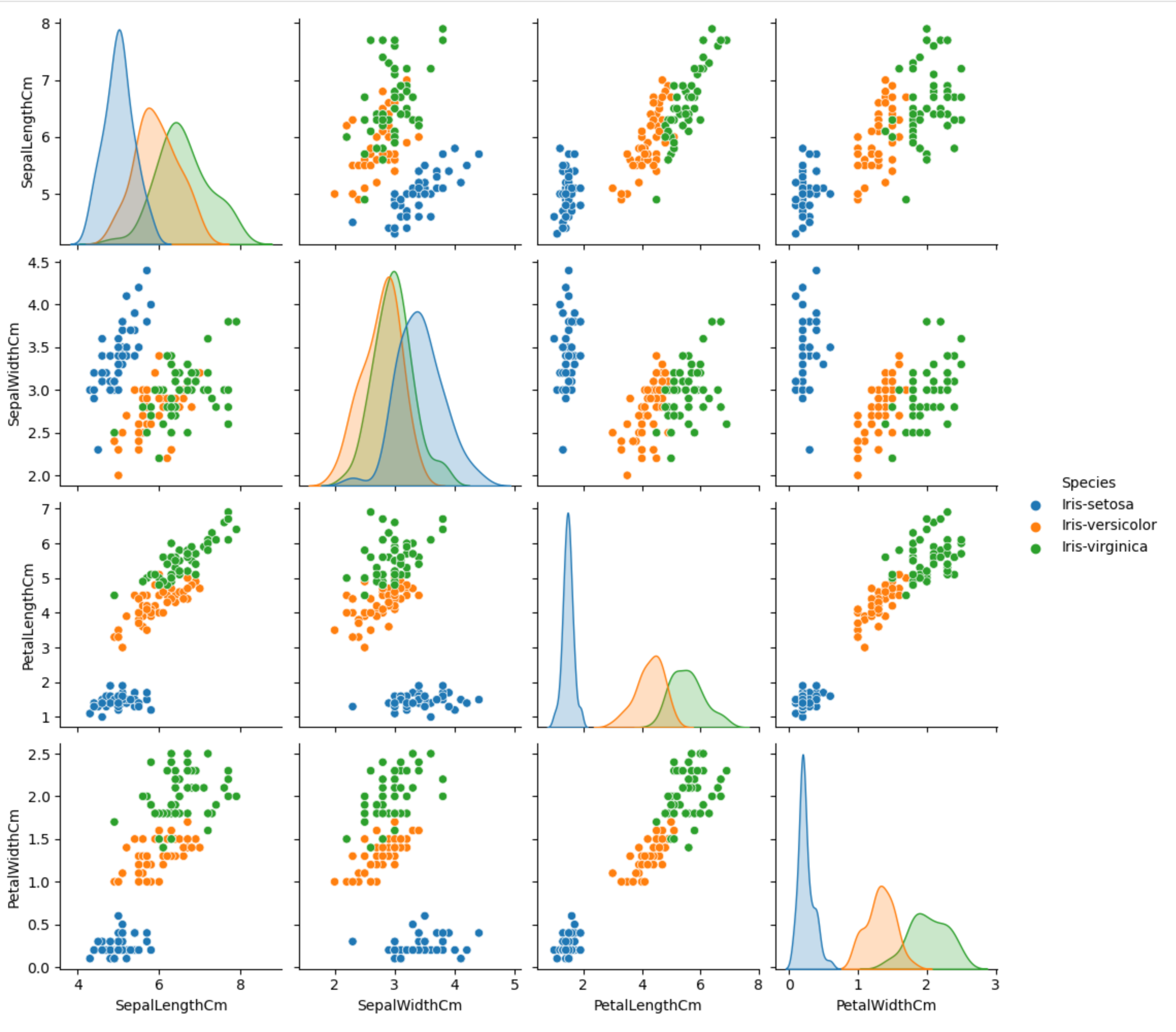
	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa

In [140…
```
sns.heatmap(iris_DF.corr(),annot = True)
```

Out[140]:
```
<AxesSubplot:>
```



In [141…
```
sns.pairplot(iris_DF , hue = "Species")
plt.show()
```



In [142…
```
iris_DF.corr()
```

Out[142]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

In [143…
```
iris_DF.corr().style.background_gradient(cmap = 'cool')
```

Out[143]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
SepalLengthCm	1.000000	-0.109369	0.871754	0.817954
SepalWidthCm	-0.109369	1.000000	-0.420516	-0.356544
PetalLengthCm	0.871754	-0.420516	1.000000	0.962757
PetalWidthCm	0.817954	-0.356544	0.962757	1.000000

Selected_features = ['SepalLengthCm', 'SepalWidthCm'] input_Parameter = iris_DF[Selected_features] target_Parameter = iris_DF['Species']

In [144…
```
input_Parameter = iris_DF.drop("Species",axis=1)
target_Parameter = iris_DF.Species
```

In [145…
```
input_Parameter.shape
```

Out[145]:
```
(150, 4)
```

In [146…
```
target_Parameter.shape
```

Out[146]:
```
(150,)
```

#Standard scaling test

In [147…
```
sc = StandardScaler()
input_Parameter = sc.fit_transform(input_Parameter)
```

In [148…
```
#spliting data to train and test
input_train,input_test,target_train,target_test= train_test_split(input_Parameter,target_Parameter,test_size=0.5,random_state=0)
```

In [149…
```
#algorithm-decision tree
DT = DecisionTreeClassifier()
DT.fit(input_train,target_train)
DTP=DT.predict(input_test)
DTA=accuracy_score(DTP,target_test)
print("percent accuracy = ",round(DTA*100,2),"%")
```

```
percent accuracy =  96.0 %
```

In [150…
```
#algorithm KNeighbors
KN=KNeighborsClassifier(n_neighbors=9)
KN.fit(input_train,target_train)
KNP=KN.predict(input_test)
KNA=accuracy_score(KNP,target_test)
print("percent accuracy = ",round(KNA*100,2),"%")
```

```
percent accuracy =  90.67 %
```

In [151…
```
#Algorthim-Random forest
RF=RandomForestClassifier(n_estimators=500)
RF.fit(input_train,target_train)
RFP=RF.predict(input_test)
RFA=accuracy_score(RFP,target_test)
print("percent accuracy = ",round(RFA*100,2),"%")
```

```
percent accuracy =  94.67 %
```

In [152…
```
#algorithm-Logistic Regression
LR=LogisticRegression()
LR.fit(input_train,target_train)
LRP=LR.predict(input_test)
LRA=accuracy_score(LRP,target_test)
print("percent accuracy = ",round(LRA*100,2),"%")
```

```
percent accuracy =  92.0 %
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: