LetsGrowMore

Iris Flowers Classification ML Project :

Classification using Supervised ML

Linear Regression with Python

In this section we will see how the Python library for machine learning can be used to implement regression functions. We will start with simple linear regression involving two variables.

Simple Linear Regression

In this regression task we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied. This is a simple linear regression task as it involves just two variables.

Import Libraries

In [2]:

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

Download data

In [3]:

```
path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
colname = ["sepal length in cm","sepal width in cm","petal length in cm","petal width in cm","class"
df = pd.read_csv(path, header=None, names=colname)
df
```

Out[3]:

	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm	class
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
...	...	...	...	...	...
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

150 rows × 5 columns

In [4]:

```
# Check the info of data
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 in cm  150 non-null    float64
 1   sepal width in cm   150 non-null    float64
 2   petal length in cm  150 non-null    float64
 3   petal width in cm   150 non-null    float64
 4   class               150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [5]:

```
# Check the discription of data
df.describe()
```

Out[5]:

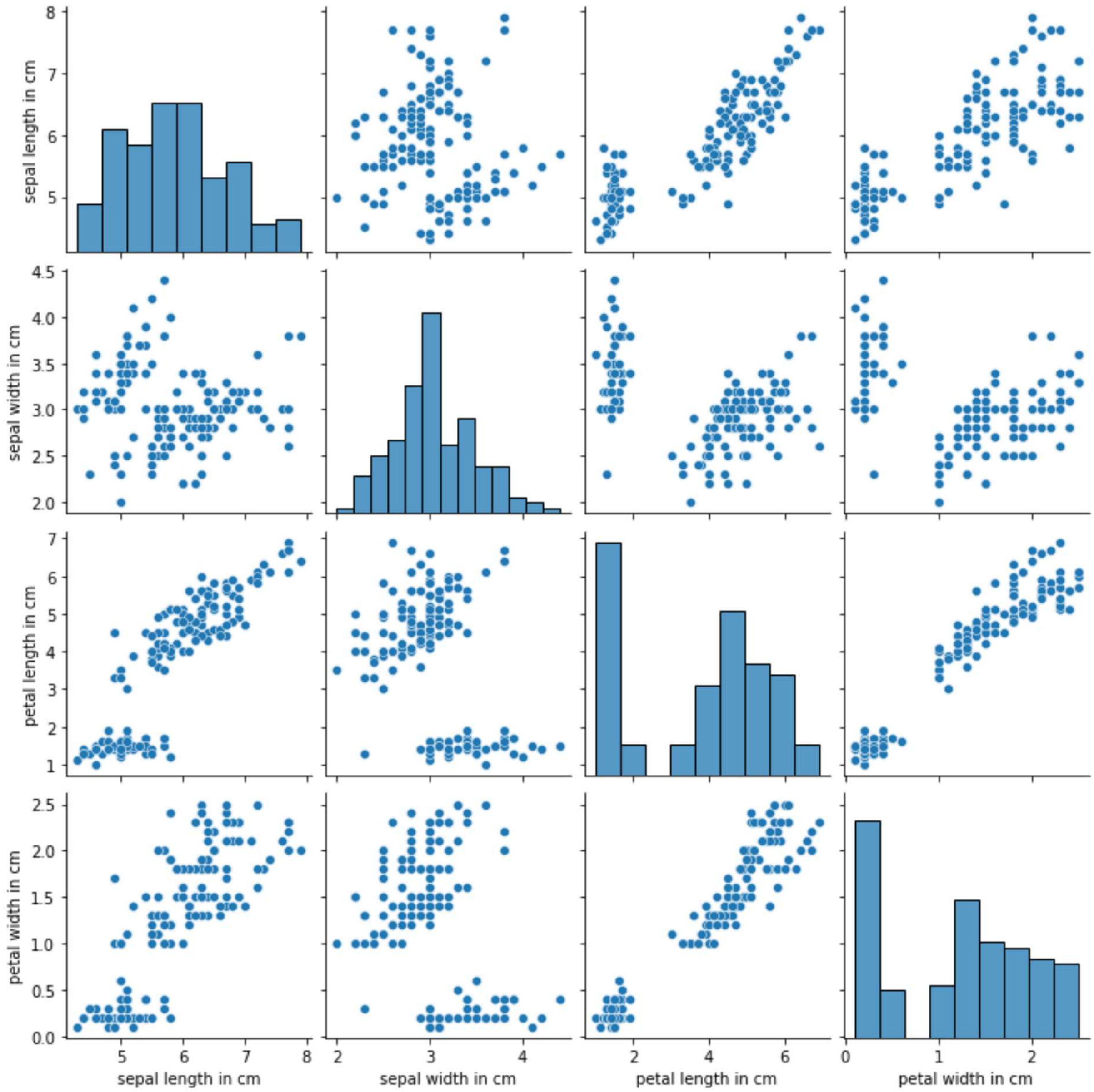
	sepal length in cm	sepal width in cm	petal length in cm	petal width in cm
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.054000	3.758667	1.198667
std	0.828066	0.433594	1.764420	0.763161
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

In [6]:

```
sns.pairplot(df)
```

Out[6]:

<seaborn.axisgrid.PairGrid at 0x2433f4b12b0>

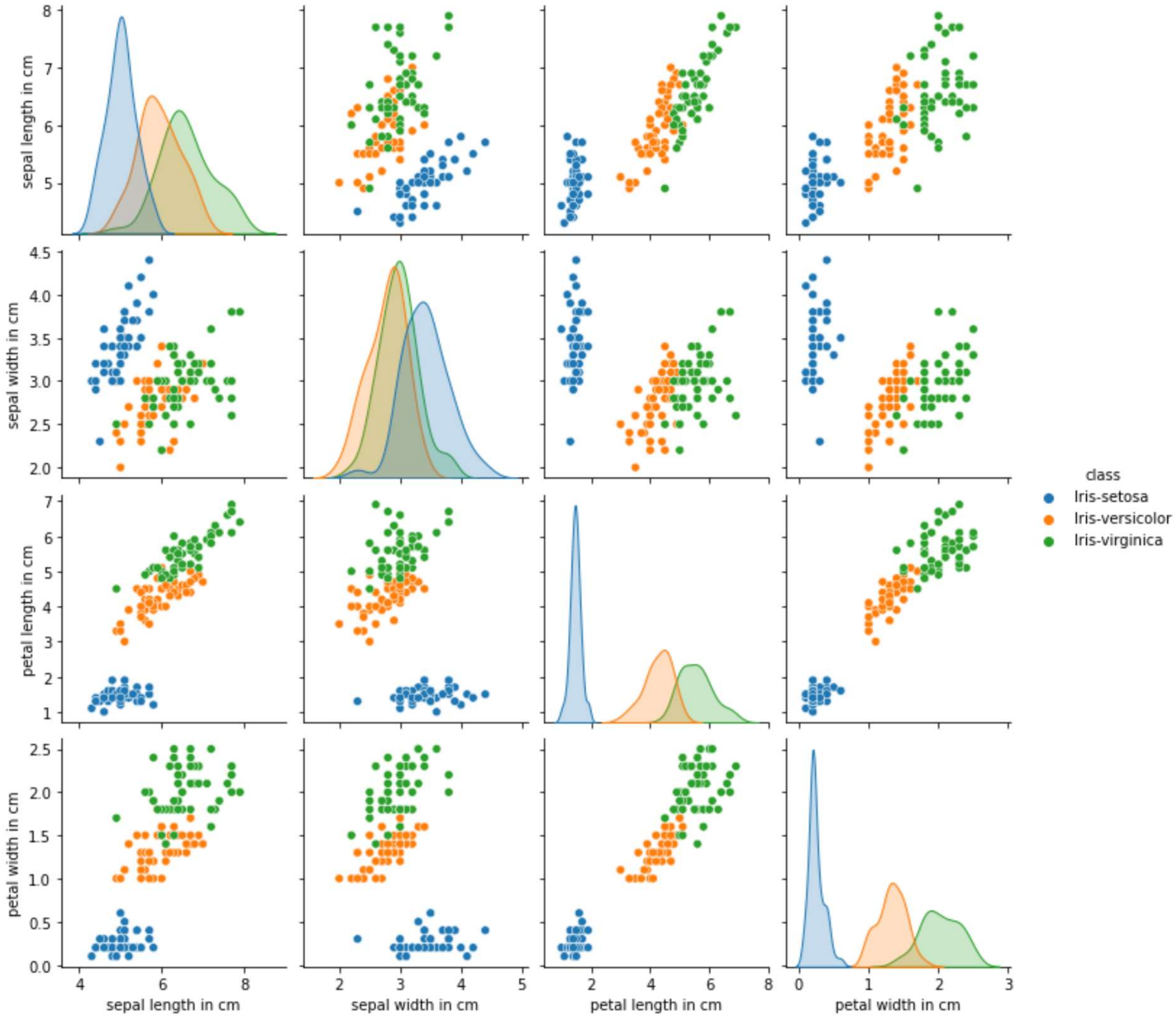


In [7]:

```
sns.pairplot(df, hue="class")
```

Out[7]:

```
<seaborn.axisgrid.PairGrid at 0x2433f31a760>
```



In [8]:

```
df["class"].value_counts()
```

Out[8]:

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: class, dtype: int64
```

We Split The Data Set Into X and Y

In [9]:

```
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

Train_Test_Split

In [10]:

```
from sklearn.model_selection import train_test_split
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
```

In [11]:

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y = le.fit_transform(y)
```

In [12]:

```
y
```

Out[12]:

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

SVM

In [13]:

```
from sklearn.svm import SVC
svm = SVC(kernel="rbf")
svm.fit(xtrain,ytrain)
ypred = svm.predict(xtest)
```

In [14]:

```
from sklearn.metrics import classification_report
print(classification_report(ytest,ypred))
```

```
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       1.00      0.94      0.97        18
 Iris-virginica       0.93      1.00      0.96        13

       accuracy                           0.98        45
      macro avg       0.98      0.98      0.98        45
   weighted avg       0.98      0.98      0.98        45
```

In [15]:

```
train = svm.score(xtrain,ytrain)
test = svm.score(xtest,ytest)

print(f"Training Accuracy:- {train}\n Testing Accuracy:- {test}")
```

```
Training Accuracy:- 0.9619047619047619
 Testing Accuracy:- 0.9777777777777777
```

In [16]:

```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred = model.predict(xtest)

    train = model.score(xtrain,ytrain)
    test = model.score(xtest,ytest)

    print(f"Training Accuracy:- {train}\n Testing Accuracy:- {test}")

    print(classification_report(ytest,ypred))
    return model
```

In [17]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import classification_report
```

In [18]:

```
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier
```

In [19]:

```
mymodel(AdaBoostClassifier())
```

```
Training Accuracy:- 0.9619047619047619
 Testing Accuracy:- 0.9555555555555556
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       0.94      0.94      0.94        18
 Iris-virginica       0.92      0.92      0.92        13

       accuracy                           0.96        45
      macro avg       0.96      0.96      0.96        45
   weighted avg       0.96      0.96      0.96        45
```

Out[19]:

```
AdaBoostClassifier()
```

In [20]:

```
logreg = mymodel(LogisticRegression())
```

```
Training Accuracy:- 0.9809523809523809
 Testing Accuracy:- 0.9777777777777777
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       1.00      0.94      0.97        18
 Iris-virginica       0.93      1.00      0.96        13

       accuracy                           0.98        45
      macro avg       0.98      0.98      0.98        45
   weighted avg       0.98      0.98      0.98        45
```

In [21]:

```
knn = mymodel(KNeighborsClassifier())
```

```
Training Accuracy:- 0.9523809523809523
 Testing Accuracy:- 0.9777777777777777
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       0.95      1.00      0.97        18
 Iris-virginica       1.00      0.92      0.96        13

       accuracy                           0.98        45
      macro avg       0.98      0.97      0.98        45
   weighted avg       0.98      0.98      0.98        45
```

In [22]:

```
mymodel(GradientBoostingClassifier())
```

```
Training Accuracy:- 1.0
 Testing Accuracy:- 0.9555555555555556
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       0.94      0.94      0.94        18
 Iris-virginica       0.92      0.92      0.92        13

       accuracy                           0.96        45
      macro avg       0.96      0.96      0.96        45
   weighted avg       0.96      0.96      0.96        45
```

Out[22]:

```
GradientBoostingClassifier()
```

In [25]:

```
dt = mymodel(DecisionTreeClassifier())
```

```
Training Accuracy:- 1.0
 Testing Accuracy:- 0.9555555555555556
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       0.94      0.94      0.94        18
 Iris-virginica       0.92      0.92      0.92        13

       accuracy                           0.96        45
      macro avg       0.96      0.96      0.96        45
   weighted avg       0.96      0.96      0.96        45
```

In [ ]:

```

```