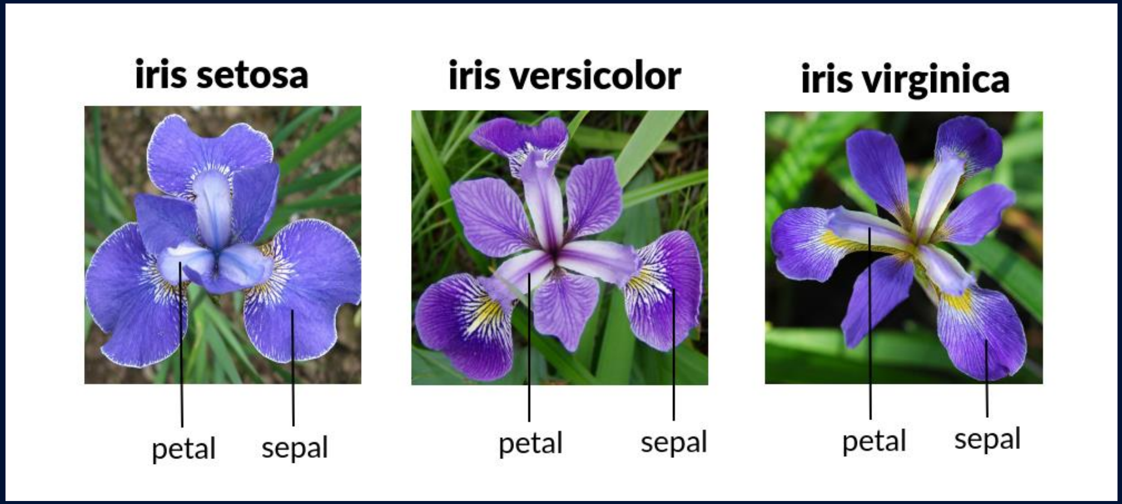
Project -1

Iris Flower Classification

Introduction

- Iris flower dataset is a classic dataset in the field of machine learning and data science.



- This dataset, first introduced by biologist Ronald Fisher in 1936.

Objective

- Our main objective with this project is to develop a machine learning model that can accurately classify iris flowers into their respective species based on these measurements. By doing so, we aim to showcase the power of machine learning algorithms in solving real-world classification tasks.

Methodology

"To achieve our objective, we will follow a systematic approach:

- **Data Exploration**
- **Data Preprocessing**
- **Model Development**
- **Model Evaluation**
- **Model Selection**

Data Exploration

1. Data Exploration

In [2]:

```python
import os
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')
```

Loading dataset

In [3]:

```python
iris=pd.read_csv('Iris.csv')
```

In [4]:

```python
iris.head(5)
```

Out[4]:

	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

Missing value

1. Removal of unwanted column

In [5]:
```
iris.drop(columns='Id', inplace=True)
```

2. Missing Value

In [6]:
```
iris.isnull().sum()
```

Out[6]:
```
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

No missing value here.

Statistical Summary

In [7]:

```
iris.describe()
```

Out[7]:

	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
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

Datatypes

In [8]:

```python
iris.info()
```

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

Correlation Analysis

In [9]:

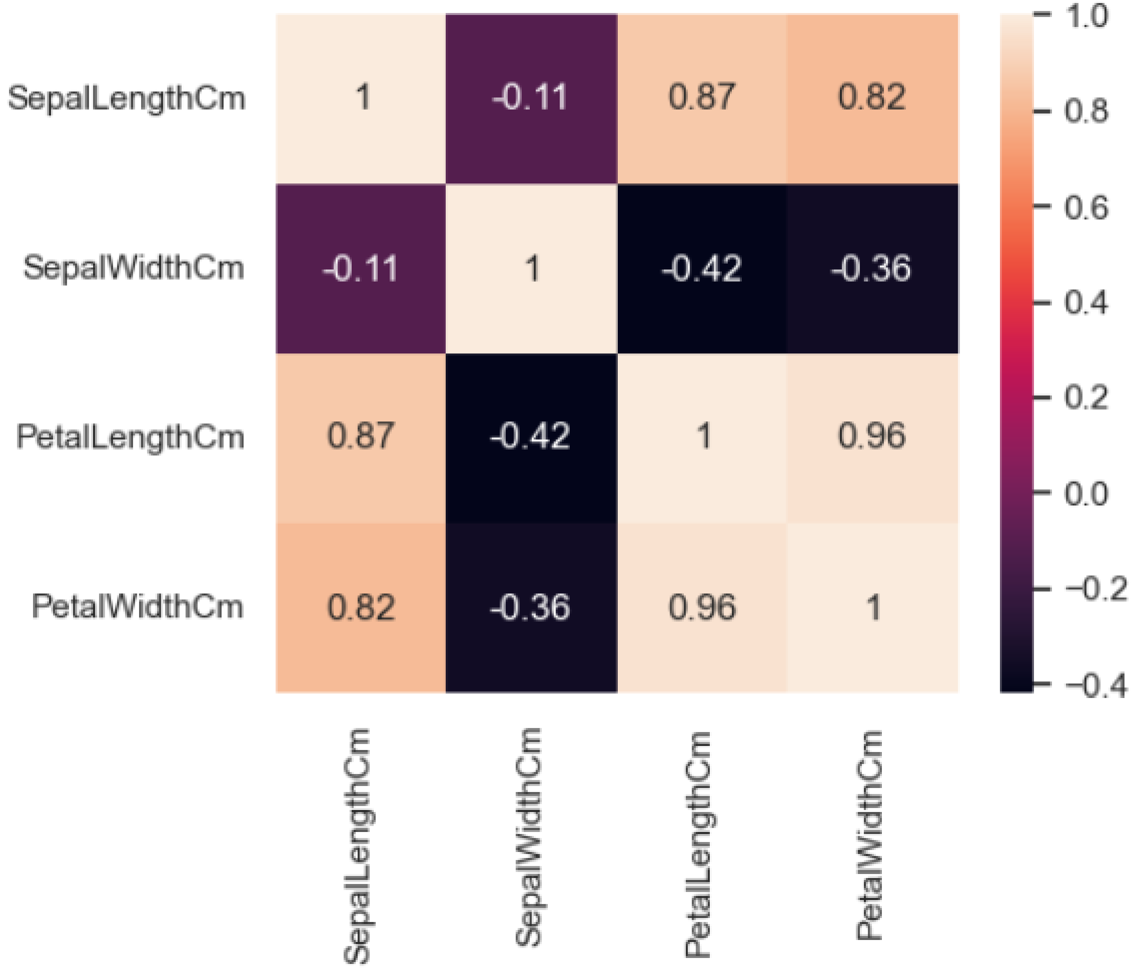
```python
corr = iris.corr()
print(corr)
```

```
               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
```

Heatmap

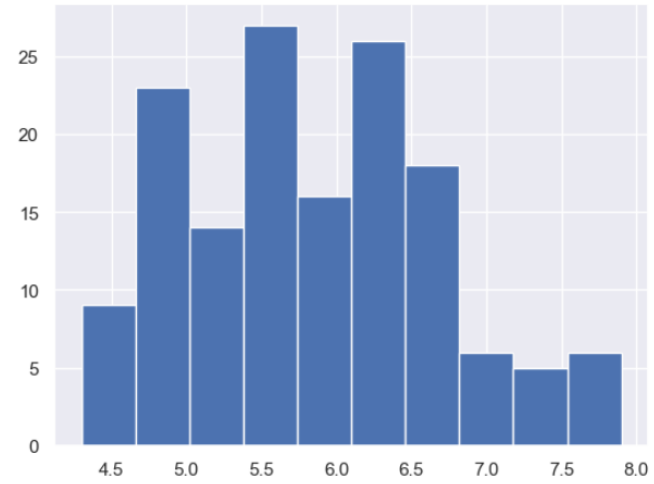
In [15]:
```
fig,ax=plt.subplots(figsize=(5,4))
sns.heatmap(corr,annot=True,ax=ax)
```

Out[15]: <Axes: >

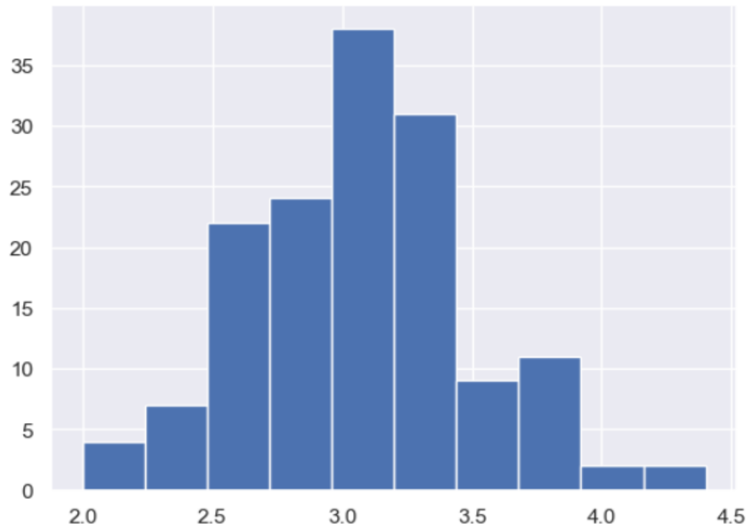


Distribution of numerical features

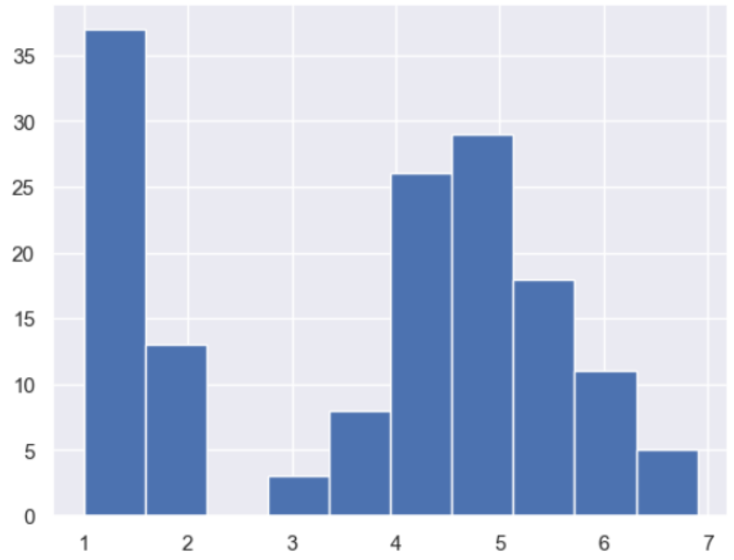
In [16]:
```
iris['SepalLengthCm'].hist()
plt.show()
```



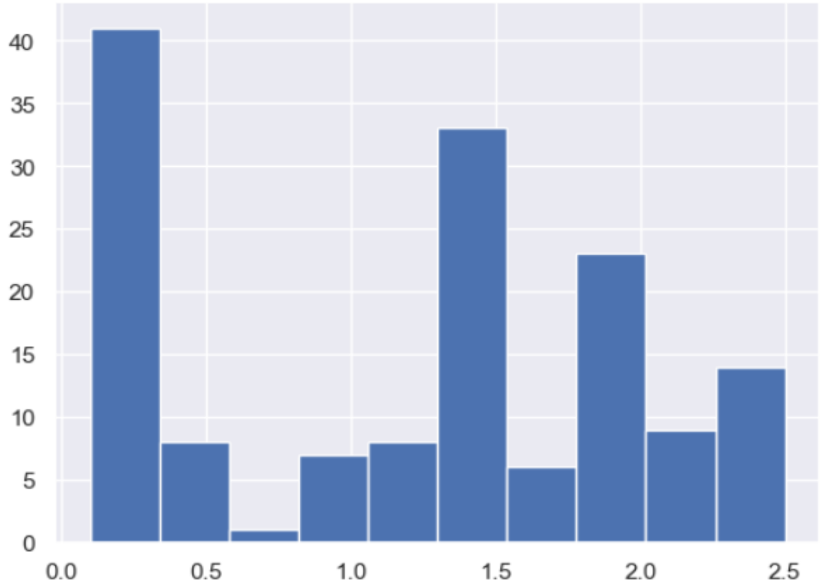
In [17]:
```
iris['SepalWidthCm'].hist()
plt.show()
```



]:
```
iris['PetalLengthCm'].hist()
plt.show()
```




```
iris['PetalWidthCm'].hist()
plt.show()
```



Label Encoder

```
from sklearn.preprocessing import LabelEncoder

```

```
la_en = LabelEncoder()
iris['Species']=la_en.fit_transform(iris['Species'])

```

```
iris.head(3)
```

	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

Splitting into independent and dependent variable

```
x=iris.drop(columns=['Species'])
y=iris['Species']
```

```
x.shape
```

```
(150, 4)
```

```
y.shape
```

```
(150,)
```

```
y.head()
```

```
0    0
1    0
2    0
3    0
4    0
Name: Species, dtype: int32
```

Splitting into Train and Test

```
from sklearn.model_selection import train_test_split
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=12)
```

```
print("x_train shape:", x_train.shape)
print("x_test shape:", x_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

```
x_train shape: (120, 4)
x_test shape: (30, 4)
y_train shape: (120,)
y_test shape: (30,)
```

Model Building

```
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold, cross_val_score
from sklearn.ensemble import RandomForestClassifier
```

```
# Importing dependencies

from sklearn.model_selection import cross_val_score
from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV
```

```
# Accuracy score
```

```
models = [LogisticRegression(max_iter=1000),DecisionTreeClassifier(),RandomForestClassifier(),KNeighborsClassifier()]
```

```
def compare_models_train_test():
    for model in models:
        model.fit(x_train,y_train)
        y_predicted = model.predict(x_test)
        accuracy = accuracy_score(y_test,y_predicted)
        print("Accuracy of the ",model,"=",accuracy)
        print("="*100)
```

```
compare_models_train_test()

```

```
Accuracy of the  LogisticRegression(max_iter=1000) = 0.9666666666666667
====================================================================================================
Accuracy of the  DecisionTreeClassifier() = 0.9666666666666667
====================================================================================================
Accuracy of the  RandomForestClassifier() = 0.9333333333333333
====================================================================================================
Accuracy of the  KNeighborsClassifier() = 0.9666666666666667
====================================================================================================
```

Cross Validation

1.Model Performance Estimation: Cross-validation provides a more reliable estimate of how well a model will perform on unseen data compared to a simple train/test split. By using multiple subsets of the data for training and testing, cross-validation provides a more robust estimate of model performance.

2. Model Selection: Cross-validation helps in comparing different models or different hyperparameter settings for the same model. By performing cross-validation on each candidate model, you can select the one that performs the best on average across different subsets of the data.

Cross Validation

```python
models = [LogisticRegression(max_iter=1000),DecisionTreeClassifier(),RandomForestClassifier(),KNeighborsClassifier()]
```

```python
def compare_models_cv():
    for model in models:
        cv_score =cross_val_score(model,x,y,cv=5)
        mean_accuracy = sum(cv_score)/len(cv_score)
        mean_accuracy= mean_accuracy*100
        mean_accuracy = round(mean_accuracy,2)
        print("cv_score of the",model,"=",cv_score)
        print("mean_accuracy % of the",model,"=",mean_accuracy,"%")
        print("="*100)
```

```python
compare_models_cv()

```

```
cv_score of the LogisticRegression(max_iter=1000) = [0.96666667 1.         0.93333333 0.96666667 1.        ]
mean_accuracy % of the LogisticRegression(max_iter=1000) = 97.33 %
====================================================================================================
cv_score of the DecisionTreeClassifier() = [0.96666667 0.96666667 0.9        0.96666667 1.        ]
mean_accuracy % of the DecisionTreeClassifier() = 96.0 %
====================================================================================================
cv_score of the RandomForestClassifier() = [0.96666667 0.96666667 0.93333333 0.93333333 1.        ]
mean_accuracy % of the RandomForestClassifier() = 96.0 %
====================================================================================================
cv_score of the KNeighborsClassifier() = [0.96666667 1.         0.93333333 0.96666667 1.        ]
mean_accuracy % of the KNeighborsClassifier() = 97.33 %
====================================================================================================
```

```python
columns = ['Models', 'Accuracy', 'Precision', 'Recall', 'F1 Score', 'ROC AUC']
```

```python
tuned_results_df = pd.DataFrame(tuned_results, columns=columns)

print(tuned_results_df)
```

```
    Models  Accuracy  Precision    Recall  F1 Score   ROC AUC
0  Model_0  0.966667   0.966667  0.966667  0.966667  1.000000
1  Model_1  0.933333   0.933333  0.933333  0.933333  0.947090
2  Model_2  0.966667   0.966667  0.966667  0.966667  0.994709
3  Model_3  0.966667   0.966667  0.966667  0.966667  1.000000
```

"Among the models evaluated for iris flower classification, Logistic Regression outperforms others based on cross-validation score, accuracy, precision, recall, F1 score, and ROC AUC. Its superior performance indicates Logistic Regression as the most suitable choice for accurate classification of iris flower species."