In [1]:
```python
#Pandas library is imported to perform Data manipulation and analysis.
#Numpy library is used to do mathematical operations.
#Matplotlib library is used to plot graphs.
#First the excel data sheet is directly read into Jupyter notebook to get in detail information of dataset.


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
iris = pd.read_csv('C:\\Users\\harsh\\OneDrive\\Documents\\IRIS.csv')
iris
```

Out[1]:

	sepal_length	sepal_width	petal_length	petal_width	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
...	...	...	...	...	...
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 [7]:
```python
#Shape command is used for Exploration of data to know the number of rows and columns
#First value represents number of rows and second number represents column number.

iris.shape
```

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

In [14]:
```python
#The advantage of describe is it gives the data types of all columns at once and type of each column, that is when there are
#different types of data types.
#Observations made in this table are:
#Only column five, that is last column is Discrete and Categorical
#Remaining all other columns are Continuous and Float and also gives missing values which can be an eye opener
#The difference between .describe and .info is that describe gives the statistical values and .info gives number of
#non null columns and mainly data types.


iris.describe(include='all')
```

Out[14]:

	sepal_length	sepal_width	petal_length	petal_width	species
count	150.000000	150.000000	150.000000	150.000000	150
unique	NaN	NaN	NaN	NaN	3
top	NaN	NaN	NaN	NaN	Iris-setosa
freq	NaN	NaN	NaN	NaN	50
mean	5.843333	3.054000	3.758667	1.198667	NaN
std	0.828066	0.433594	1.764420	0.763161	NaN
min	4.300000	2.000000	1.000000	0.100000	NaN
25%	5.100000	2.800000	1.600000	0.300000	NaN
50%	5.800000	3.000000	4.350000	1.300000	NaN
75%	6.400000	3.300000	5.100000	1.800000	NaN
max	7.900000	4.400000	6.900000	2.500000	NaN

In [18]:
```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   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   species       150 non-null    object 
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

In [4]:
```python
#This is important method useful for this particular project.
#Because here iris flower has to be differentiated into seperate species based upon the measurements and knowing the
#unique possible species comes in handy.Output is an array.

iris['species'].unique()
```

Out[4]:
```
array(['Iris-setosa', 'Iris-versicolor', 'Iris-virginica'], dtype=object)
```

In [4]:
```python
#Data exploration that is to know if any missing values are present, and how to replace them

print(iris.isnull())
print(iris.isnull().sum())
#iris[iris['petal_width']=='.'].sum()
```

```
     sepal_length  sepal_width  petal_length  petal_width  species
0           False        False         False        False    False
1           False        False         False        False    False
2           False        False         False        False    False
3           False        False         False        False    False
4           False        False         False        False    False
..            ...          ...           ...          ...      ...
145         False        False         False        False    False
146         False        False         False        False    False
147         False        False         False        False    False
148         False        False         False        False    False
149         False        False         False        False    False

[150 rows x 5 columns]
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

In [9]:
```python
#The value count method gives the count of each indivisual variable present in particular column
#In this all remaining columns are measurements, and count method is only applicable to species column
iris["species"].value_counts()
```

Out[9]:
```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

In [17]:
```python
#Statistical Data
#MEAN, MEDIIAN, MODE can be calculated for each column, in this case since all are continuous variables all the
#statistical parameters can be applied

#mean = sum of total observations/number of odservations
mean_sepal_length = iris['sepal_length'].mean()
print('mean_sepal_length is: ',mean_sepal_length)
#mean = iris['sepal_width'].mean()
#mean_sepal width
```

```
mean_sepal_length is:  5.843333333333335
```

In [27]:
```python
#Mode is the frequency of particular data variable which occurs more frequently than others.
mode_petal_length = iris['petal_length'].mode()[0]
print('mode_petal_width is: ',mode_petal_length)
```

```
mode_petal_width is:  1.5
```

In [25]:
```python
species = iris['species'].mode()
print(species)
```

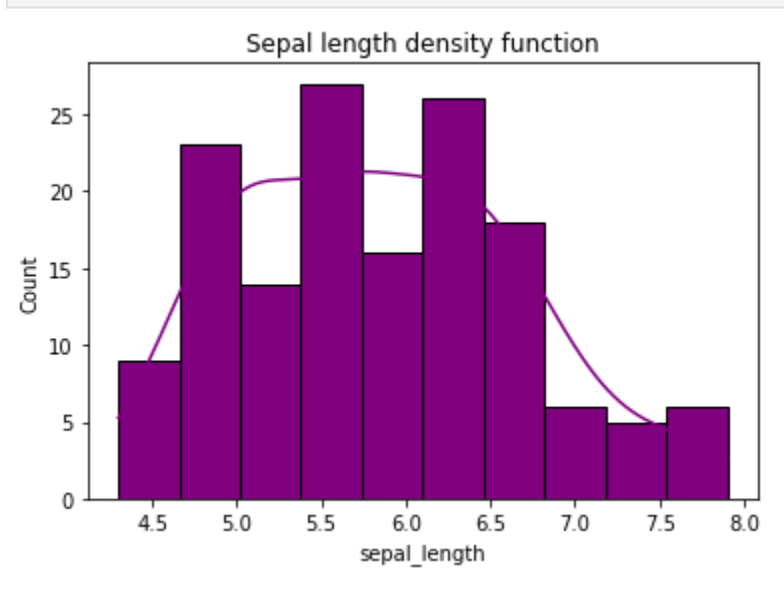
```
0        Iris-setosa
1    Iris-versicolor
2     Iris-virginica
dtype: object
```

In [26]:
```python
#Median in statistics, is the middle value of the given list of data when arranged in either ascending or descending order
median_petal_width = iris['petal_width'].median()
print('median_petal_width is: ',median_petal_width)
```

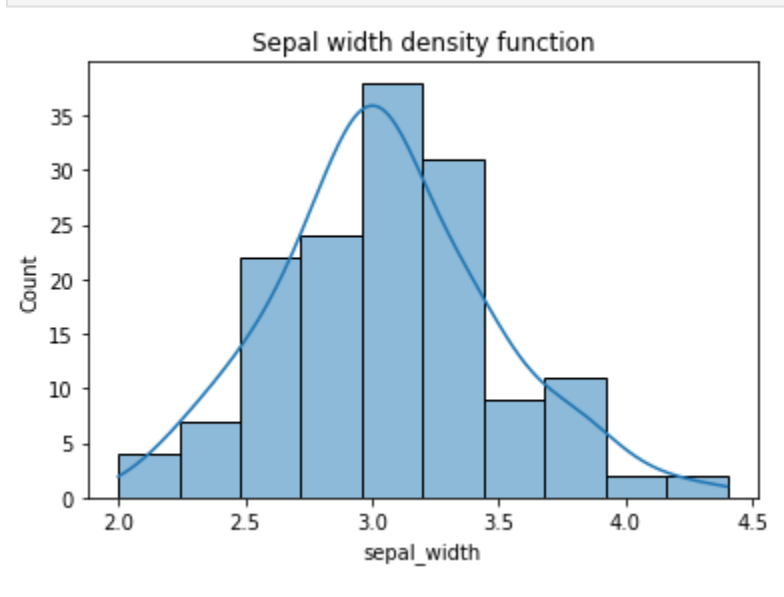
```
median_petal_width is:  1.3
```

In [6]:
```python
import seaborn as sns
```

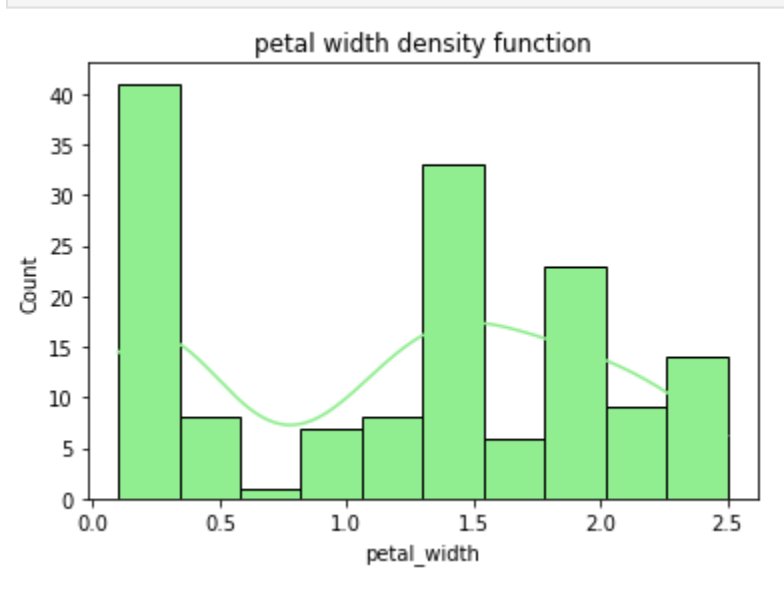
In [8]:
```python
#plt.scatter(iris['sepal_width'],iris['sepal_length'],marker='*',facecolor='green',edgecolor='Navy',s=40)
#kernel density estimate is kde, is used with histogram,it represents the line
#The probability density function (pdf) is used to describe probabilities for continuous random variables.In this case the
#sepal length

#Histogram is used to express the distribution of data particularly when the data is continuous.
sns.histplot(iris['sepal_length'],bins=10,kde=True,facecolor='purple')
plt.xlabel('sepal_length')
plt.title('Sepal length density function')
plt.show()
```



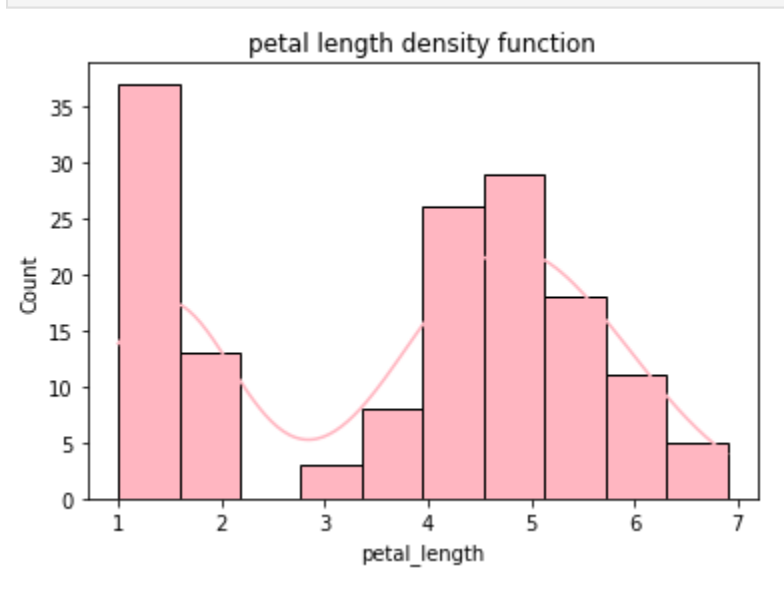
In [15]:
```python
sns.histplot(iris['sepal_width'],bins=10,kde=True)
plt.xlabel('sepal_width')
plt.title('Sepal width density function')
plt.show()
```



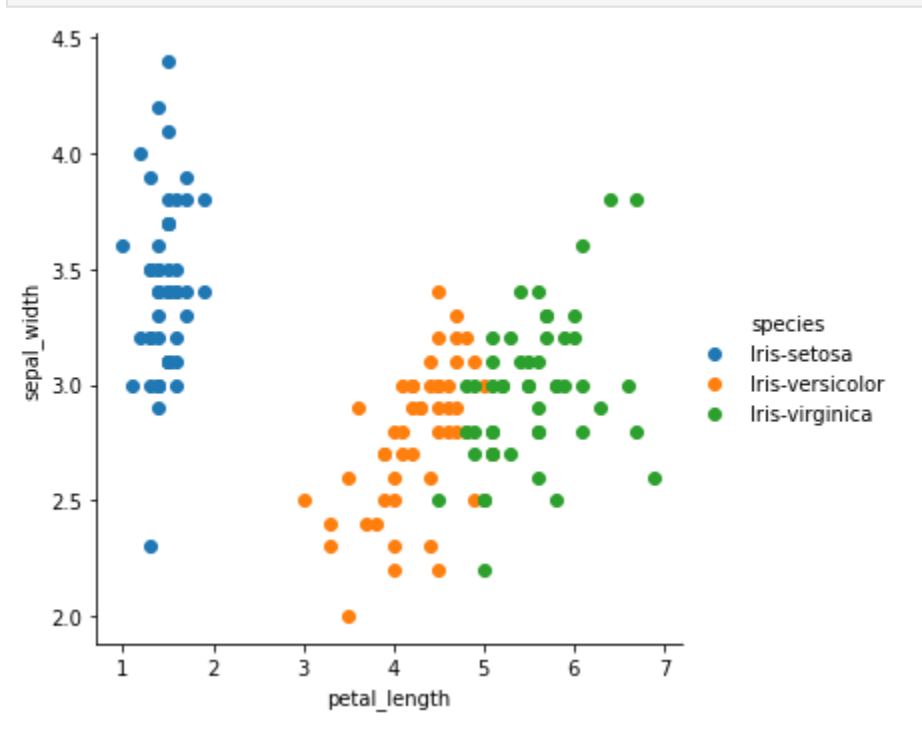
In [25]:
```python
sns.histplot(iris['petal_width'],bins=10,kde=True,facecolor='green')
plt.xlabel('petal_width')
plt.title('petal width density function')
plt.show()
```



In [26]:
```python
sns.histplot(iris['petal_length'],bins=10,kde=True,facecolor='lightpink')
plt.xlabel('petal_length')
plt.title('petal length density function')
plt.show()
```



In [13]:
```python
#The Facetgrid gives the different columns as subsets on the graph.
#The hue gives color plot aspects based on the values of a specific variable

sns.FacetGrid(iris,hue='species',height=5).map(plt.scatter,'petal_length','sepal_width').add_legend()
plt.show()
```

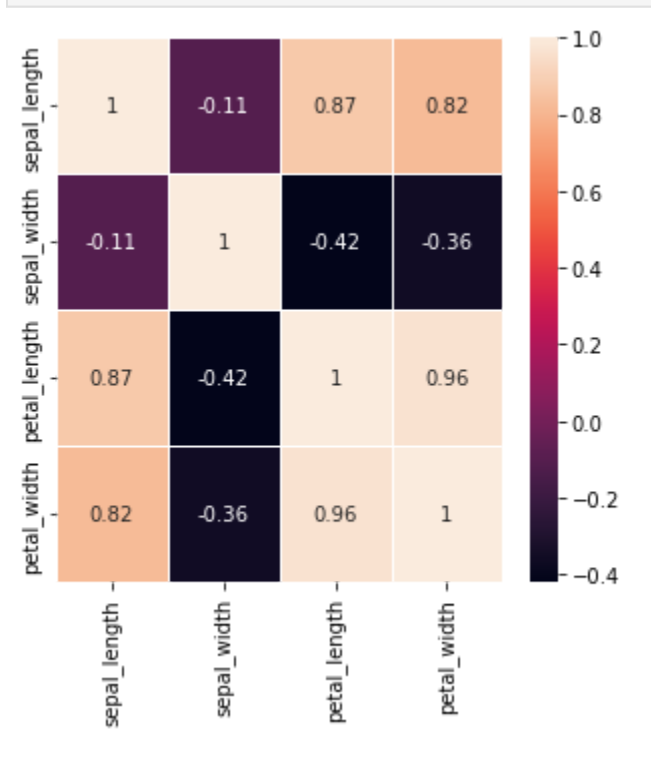


In [27]:
```python
#Correlation matrix, all the remaining graphs gives the distribution of each indivisual column value,
#but the correlation matrix alone gives the snapshot of relation of two variables, whether they are connected strongly,
#inversely or they are independent of each other.
#The heatmap basically it shows correlation between all numerical variables in the dataset.

#This command gives the matrix of relation, for better understanding it is converted to graph[heatmap]
iris.corr()
```

Out[27]:

	sepal_length	sepal_width	petal_length	petal_width
sepal_length	1.000000	-0.109369	0.871754	0.817954
sepal_width	-0.109369	1.000000	-0.420516	-0.356544
petal_length	0.871754	-0.420516	1.000000	0.962757
petal_width	0.817954	-0.356544	0.962757	1.000000

In [14]:
```python
#Heat map is designed using seaborn library
heat_map = iris.corr()
fig = plt.subplots(figsize=(5,5))
sns.heatmap(heat_map,annot=True,linewidths = 1)
plt.show()
```



LABEL ENCODING: This is done because the machine language algoritms do not understand labels, they have to be converted to nummerical values: Like three species are present in above dataset, they each can be represented using three numbers.

In [15]:
```python
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()#defining the function
iris['species']=le.fit_transform(iris['species'])#Fit label encoder and return encoded labels

#Now we can observe the species column instead of Iris-setosa it is 0, Iris-versicolor it is 1, Iris-virginica it is 2.
iris.head()
print(iris['species'])
```

```
0      0
1      0
2      0
3      0
4      0
      ..
145    2
146    2
147    2
148    2
149    2
Name: species, Length: 150, dtype: int32
```

MODEL TRAINING

In [16]:
```python
#Declaring 80% is used for training and 20% used for testing. The train_test_ split,
#Split arrays or matrices into random train and test subsets.
#If we observe the number of rows are 105 here, means out of total 150 rows, 20% are taken for model building
#Species column is the output and remaining columns are input

import numpy as np

from sklearn.model_selection import train_test_split
x = iris.drop(columns=['species'])
y = iris['species']

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.20)
print(x_train)
```

```
     sepal_length  sepal_width  petal_length  petal_width
101           5.8          2.7           5.1          1.9
34            4.9          3.1           1.5          0.1
0             5.1          3.5           1.4          0.2
92            5.8          2.6           4.0          1.2
45            4.8          3.0           1.4          0.3
..            ...          ...           ...          ...
82            5.8          2.7           3.9          1.2
136           6.3          3.4           5.6          2.4
104           6.5          3.0           5.8          2.2
128           6.4          2.8           5.6          2.1
144           6.7          3.3           5.7          2.5

[120 rows x 4 columns]
```

In [18]:
```python
#SVM
#Support Vector Machine is supervised classification algorithm, used to classify data into different classes even though
#its not visually differentiable

import numpy as np
import pandas as pd
from sklearn.svm import SVC
model_svc = SVC()
model_svc.fit(x_train,y_train)
```

Out[18]:
```
SVC()
```

In [19]:
```python
prediction1 = model_svc.predict(x_test)

from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,prediction1)*100)
```

```
100.0
```

LOGISTIC REGRESSION

In [20]:
```python
#Logistic regression is one of the most popular Machine Learning algorithms, which comes under the Supervised Learning technique.
#It is used for predicting the categorical dependent variable using a given set of
#independent variables. It's supervised because directions on how to work with data is specified.

from sklearn.linear_model import LogisticRegression
model = LogisticRegression(solver='lbfgs', max_iter=1000)

#lbfgs stand for: "Limited-memory Broyden–Fletcher–Goldfarb–Shanno Algorithm".
#It is one of the solvers' algorithms provided by Scikit-Learn Library.

model.fit(x_train,y_train)

print('Accuracy of linear regression model is: ',model.score(x_test,y_test)*100)

predicted = model.score(x_test,y_test)*100
#print(predicted)
```

```
Accuracy of linear regression model is:  100.0
```

In [ ]:
```python
#If we use the entire data for model building, we will not be left with any data for testing. So generally,
#we split the entire data set into two parts, say 70/30 percentage.
#We use 70% of the data for model building and the rest for testing the accuracy in prediction of our created model.
```

In [21]:
```python
#Prdicting using new input
x_new = np.array([[3.9,3.9,1.6,0.4],[4.5,2.6,3.7,1.8],[5.4,2.5,4.8,1.4]])

prediction = model_svc.predict(x_new)
print(prediction)
```

```
[0 1 1]
```

In [ ]: