Part 1 ¶

Import all needed libraries

#This particular block of code is importing the necessary libraries for data analysis and visualization using pandas, numpy, and matplotlib. It is also importing specific functions for creating scatter plot matrices and reading CSV files into DataFrames.

In [1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt

from pandas.plotting import scatter_matrix
from pandas import DataFrame, read_csv
```

Load the data set into a pandas dataframe

Read the Iris data set and create the dataframe df

#This line of code specifies the file path of the Iris dataset and reads the CSV file into a pandas DataFrame called 'df'. It then displays the first 5 rows of the DataFrame to provide a preview of the data.

In [2]:
```
filepath = '../data/Iris.csv'
df = pd.read_csv ("C:\\Users\\vijay\\Downloads\\Iris.csv")
df.head(5)
```

Out[2]:

	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

Print the information about the dataset

#This line of code prints out the summary information about the dataset stored in the DataFrame 'df', including the number of rows, columns, data types of columns, and memory usage. It provides an overview of the dataset's structure and helps to identify any missing

values or potential issues with the data.

In [3]:
```
print(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
None
```

create a histogram

SepalWidth - normal dist, PetalLength - bimodal

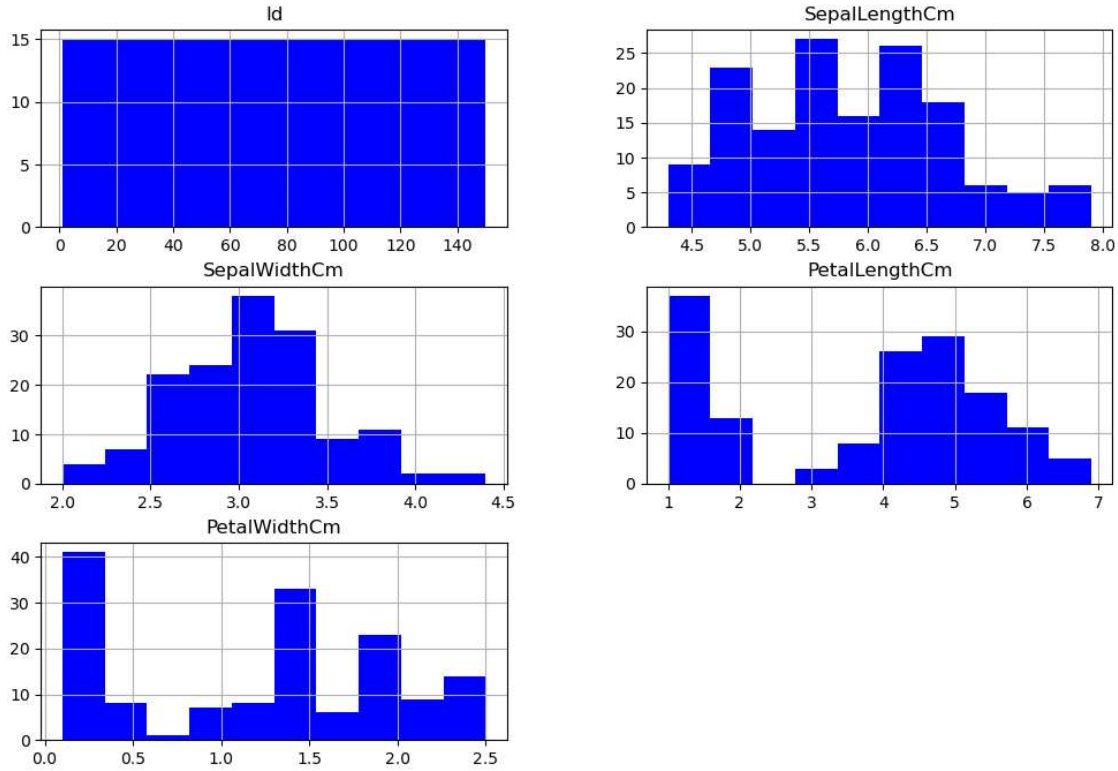
the normal distribution is so important easier for mathematical statisticians to work with.

many kinds of statistical tests can be derived from normal distributions.

#This code creates a histogram for each numeric column in the DataFrame 'df' and displays them in a figure with a size of 12x8 inches. The histograms provide a visual representation of the distribution of values in each column, helping to understand the data's range, central tendency, and spread.

In [4]:
```
df.hist(figsize=(12,8), color='blue')
plt.show
```

Out[4]: <function matplotlib.pyplot.show(close=None, block=None)>



here we want to see the different Species

#This code groups the data in the 'df' DataFrame by the 'Species' column and calculates the size of each group. It then prints the number of occurrences of each unique species, providing insights into the distribution and balance of the different species in the dataset.

In [5]:
```
print(df.groupby('Species').size())
```

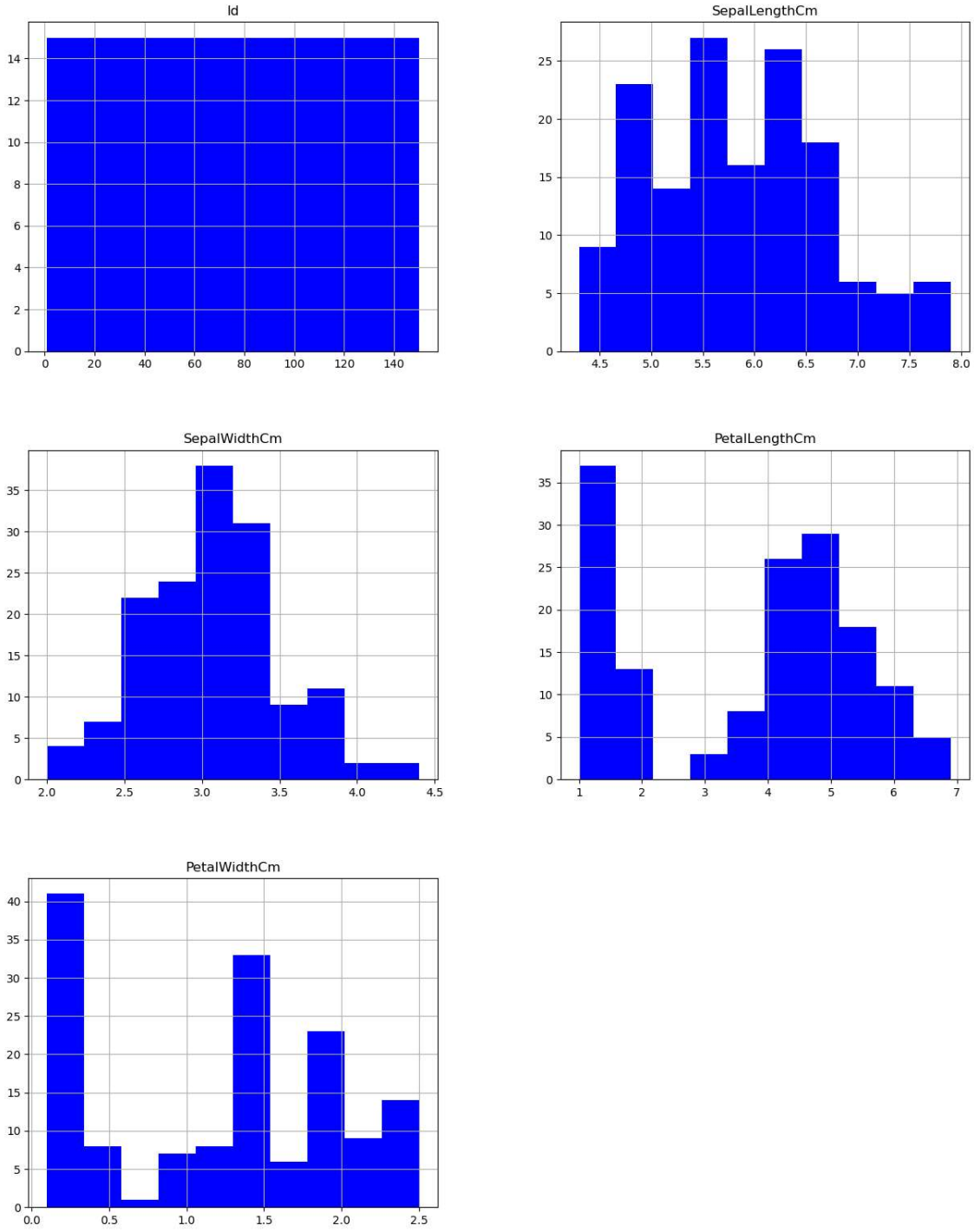
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Histograms are great when we would like to show the distribution of the data we are working with.

#This code creates multiple histograms for each numerical column in the DataFrame 'df' with a specified figure size and color. It visualizes the distribution of data for each column, helping to understand the range and shape of the data distribution.

In [6]:
```
df.hist(figsize=(15,19), color='blue')
plt.show
```

Out[6]: <function matplotlib.pyplot.show(close=None, block=None)>



In the above code, we see the variable "Id" is included in the analysis. In order to get rid of this variable we use the following code.

#This code removes the column named 'Id' from the DataFrame 'df'. It deletes the specified

In [ ]:
```
df.__delitem__('Id')
```

We are checking if the column 'Id' has been deleted from the DataFrame 'df'

#This code checks if 'Id' is present in the list of column names (df.columns). If it is not found, it indicates that the column has been deleted as shown below.

In [24]:
```
if 'Id' not in df.columns:
    print("Column 'Id' has been deleted.")
else:
    print("Column 'Id' is still present.")
```

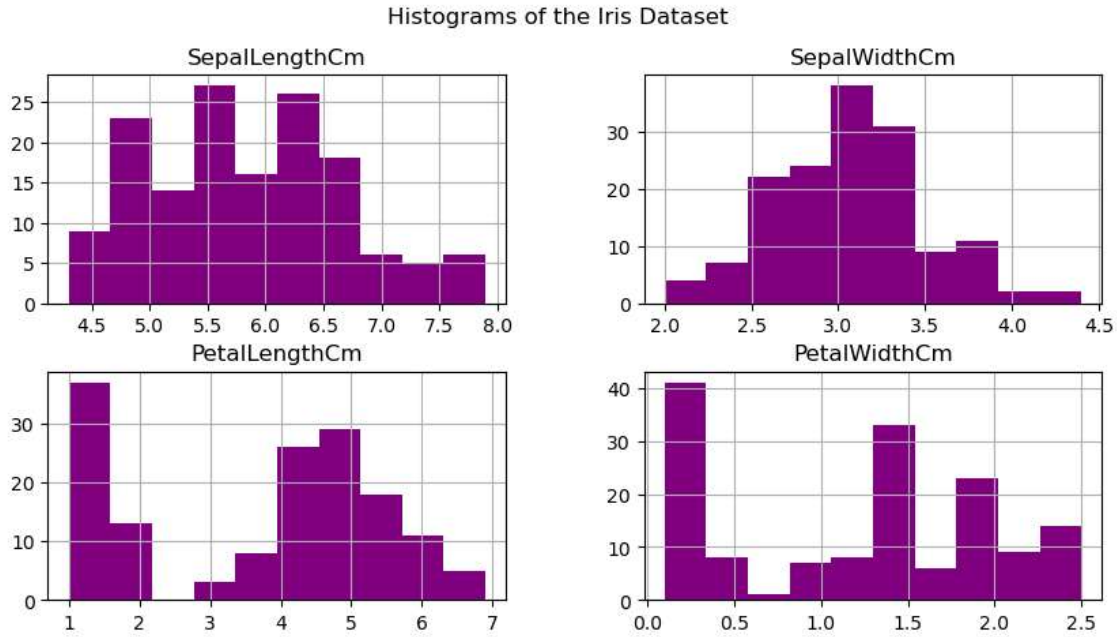
```
Column 'Id' has been deleted.
```

After this run, we see that "Id" is gone. you can also see we changed the color to purple.

#The DataFrame 'df"s columns' histograms are created by this code, and they are then plotted as subplots. The plotted histograms use a purple colour. The overall figure receives a title when the suptitle function is used. It highlights the visual impact of the choices to use purple for the histograms and visually depicts how the data is distributed among the columns.

In [25]:
```
df.hist(figsize=(10,5), color ="purple")
plt.suptitle("Histograms of the Iris Dataset")
plt.show
```

Out[25]: <function matplotlib.pyplot.show(close=None, block=None)>

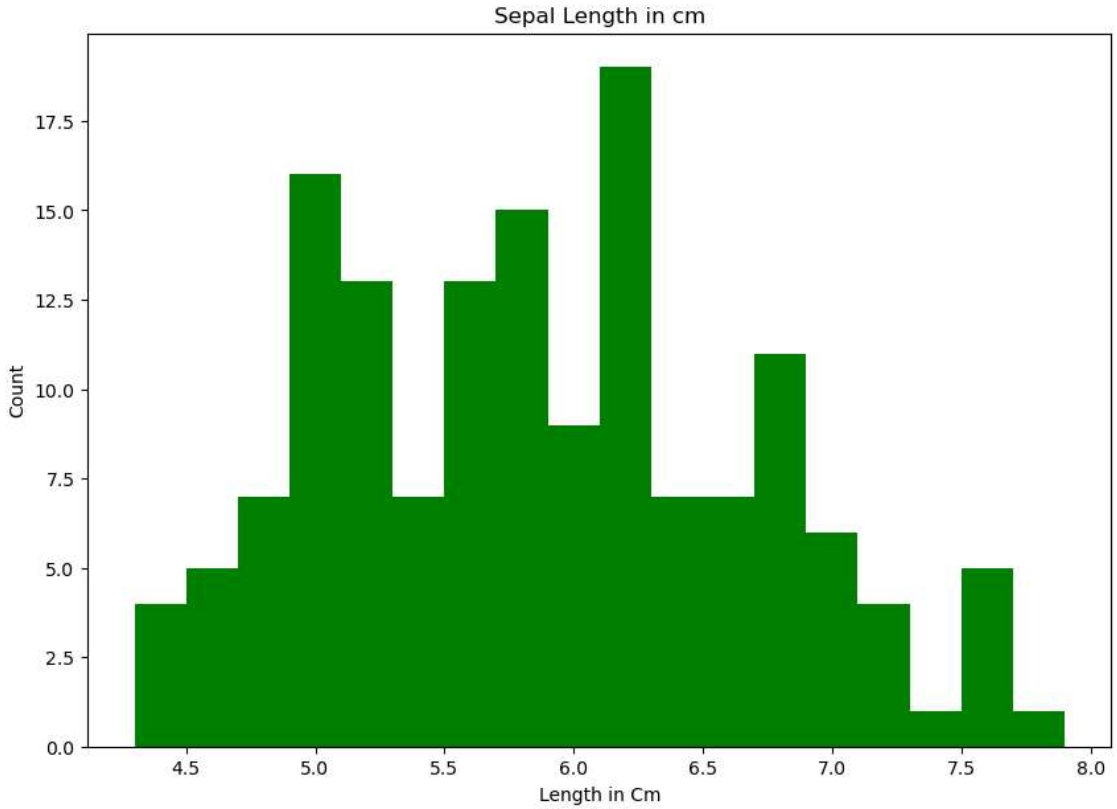


histogram with just one variable - Sepal Length. We need to isolate the one variable using square brackets [].

#This code creates a histogram for the "SepalLengthCm" variable in the DataFrame 'df'. It sets the figure size, specifies the number of bins, and chooses the color green for the histogram. The title, x-axis label, and y-axis label are added to provide a visual representation of the distribution of sepal lengths in centimeters and the count of data points in each bin.

In [26]:
```
plt.figure(figsize = (10, 7))
x = df["SepalLengthCm"]
#  bins is an integer, it defines the number of equal-width bins in the ra
plt.hist(x, bins = 18, color = "green")
plt.title("Sepal Length in cm")
plt.xlabel("Length in Cm")
plt.ylabel("Count")
```

Out[26]: Text(0, 0.5, 'Count')



here we are isolating the variable and giving each one a color and a specific number of bins.

grammar of graphics plot

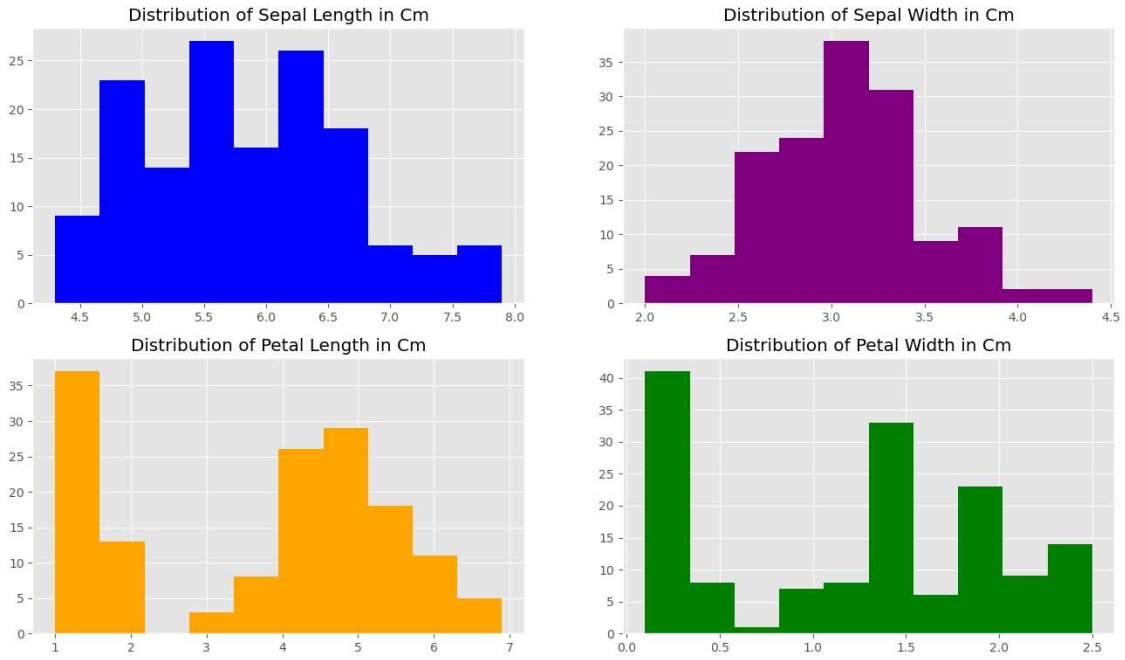
#This code creates a 2x2 grid of subplots, where in each subplots -- displaying the distribution of a specific variable from the 'df' DataFrame. The histograms are customized with titles, color, and the number of bins. The 'plt.style.use("ggplot")' line sets the plot style to "ggplot" for a consistent visual theme.

In [27]:

```
plt.style.use("ggplot")

fig, axes = plt.subplots(2, 2, figsize=(16,9))

axes[0,0].set_title("Distribution of Sepal Length in Cm")
axes[0,0].hist(df['SepalLengthCm'], bins=10, color ='blue');
axes[0,1].set_title("Distribution of Sepal Width in Cm")
axes[0,1].hist(df['SepalWidthCm'], bins=10, color ='purple');
axes[1,0].set_title("Distribution of Petal Length in Cm")
axes[1,0].hist(df['PetalLengthCm'], bins=10, color = 'orange');
axes[1,1].set_title("Distribution of Petal Width in Cm")
axes[1,1].hist(df['PetalWidthCm'], bins=10, color ='green');
```



Lastly, let's change the bin size

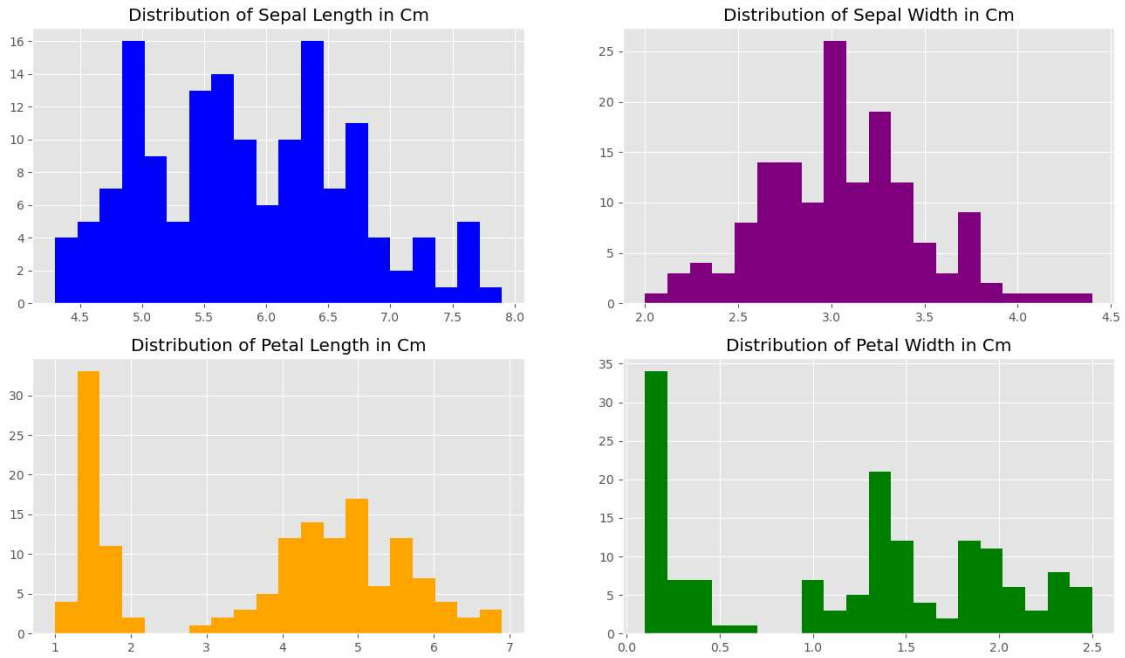
#This code creates a 2x2 grid of subplots, each displaying the distribution of a specific variable from the 'df' DataFrame. The histograms have increased bin size (20 bins) compared to the previous code, providing more detailed information about the distribution of each variable. The plot style is set to "ggplot" for a consistent visual theme.

In [28]:

```
plt.style.use("ggplot")

fig, axes = plt.subplots(2, 2, figsize=(16,9))

axes[0,0].set_title("Distribution of Sepal Length in Cm")
axes[0,0].hist(df['SepalLengthCm'], bins=20, color ='blue');
axes[0,1].set_title("Distribution of Sepal Width in Cm")
axes[0,1].hist(df['SepalWidthCm'], bins=20, color ='purple');
axes[1,0].set_title("Distribution of Petal Length in Cm")
axes[1,0].hist(df['PetalLengthCm'], bins=20, color = 'orange');
axes[1,1].set_title("Distribution of Petal Width in Cm")
axes[1,1].hist(df['PetalWidthCm'], bins=20, color ='green');
```

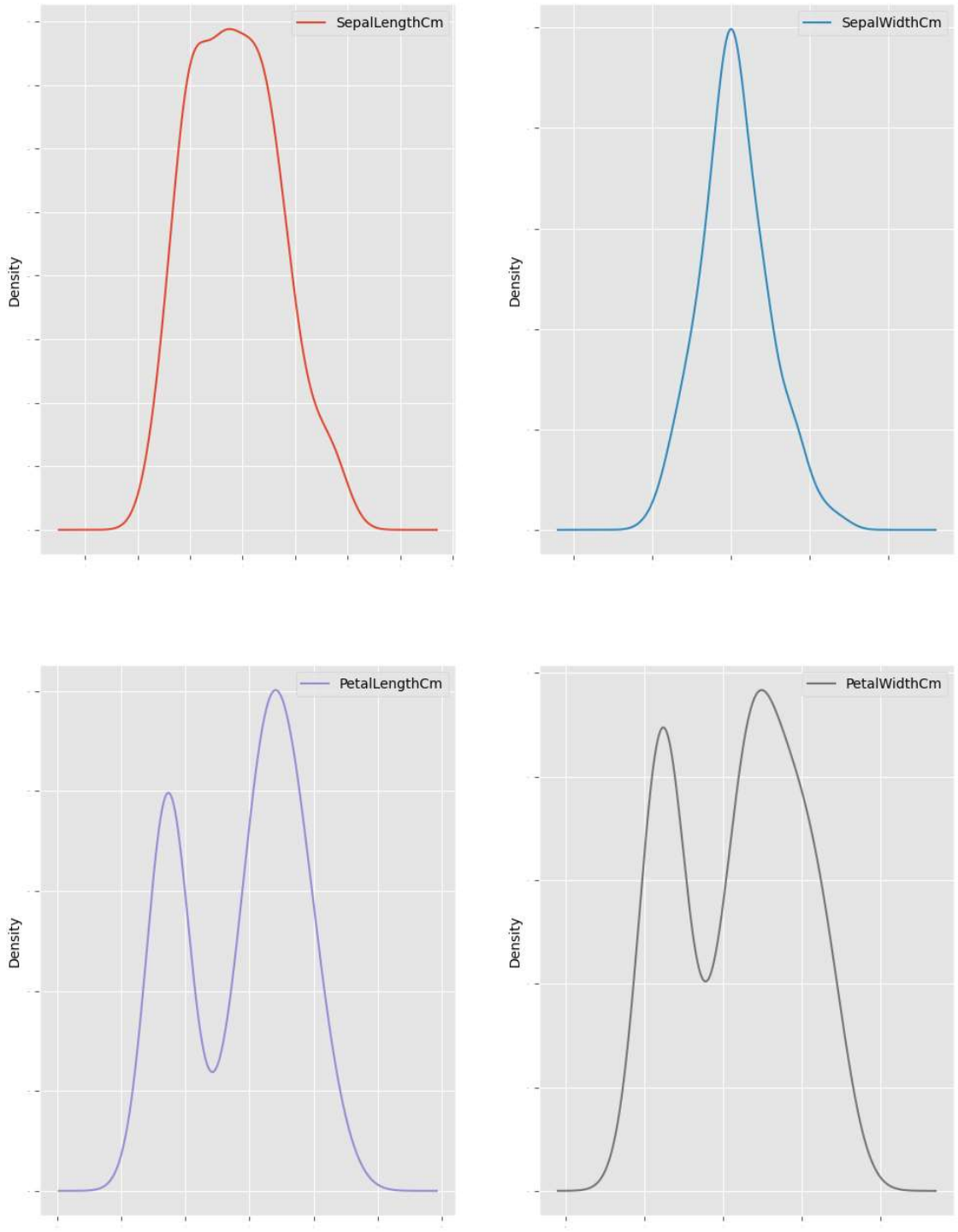


create the density plot

If subplots=True is specified, plots for each column are drawn as subplots

#This code creates a 2x2 grid of density plots for each column of the DataFrame 'df'. Each subplot represents the density distribution of a specific variable. The plots are displayed using a shared x-axis, with legends and font size settings. The overall figure size is set to (12,16) for better visibility.

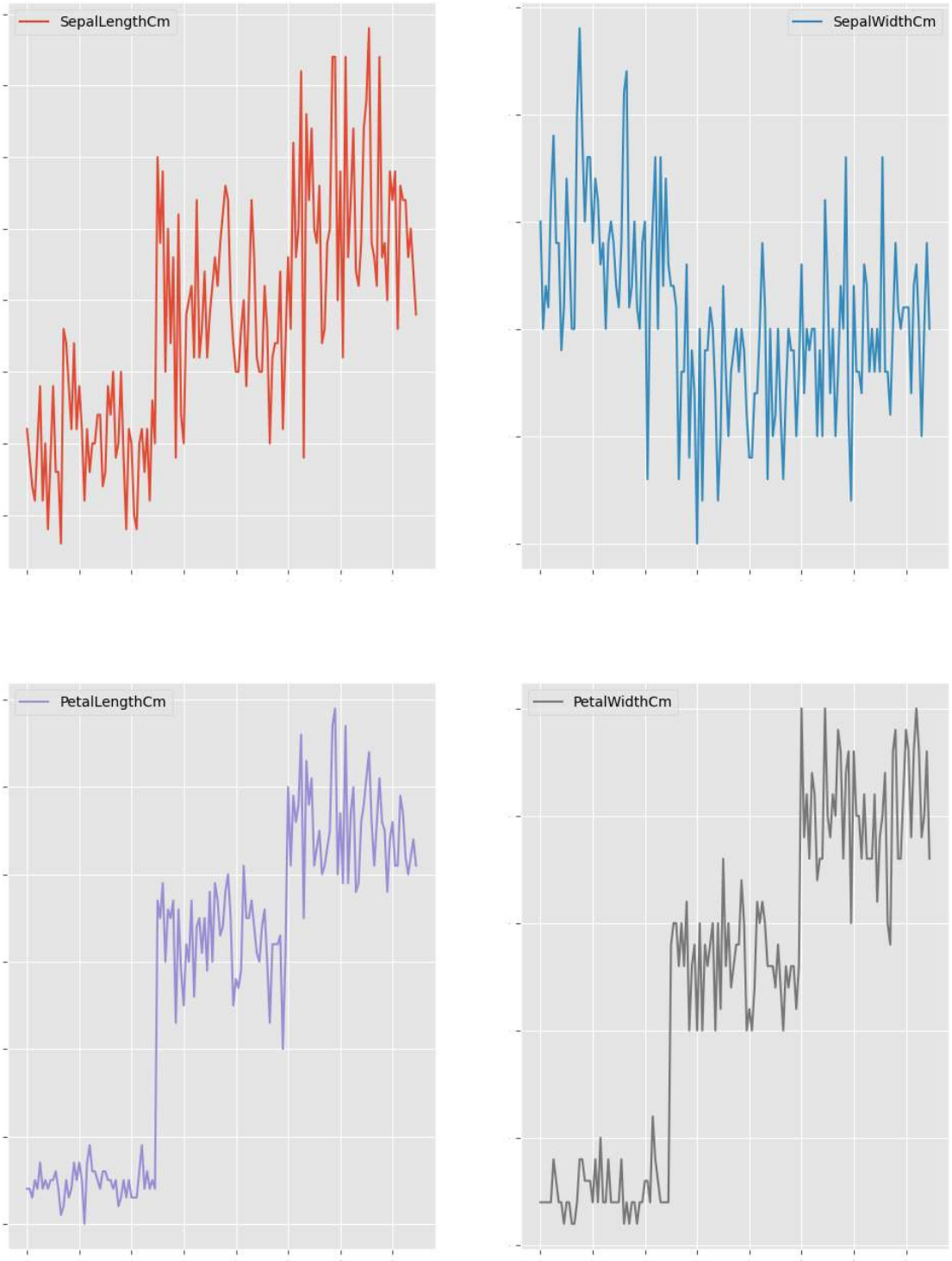
In [29]:
```
df.plot(kind='density', subplots=True, layout=(2,2), sharex=False, legend=
plt.show()
```



create a line graph

#This code creates a 2x2 grid of line graphs for each column of the DataFrame 'df'. Each subplot represents the trend of a specific variable over time or index. The plots are displayed using a shared x-axis, with legends and font size settings. The overall figure size is set to (12,16) for better visibility.

In [30]:
```
df. plot(kind='line', subplots=True, layout=(2,2), sharex=False, legend=Tr
plt.show()
```

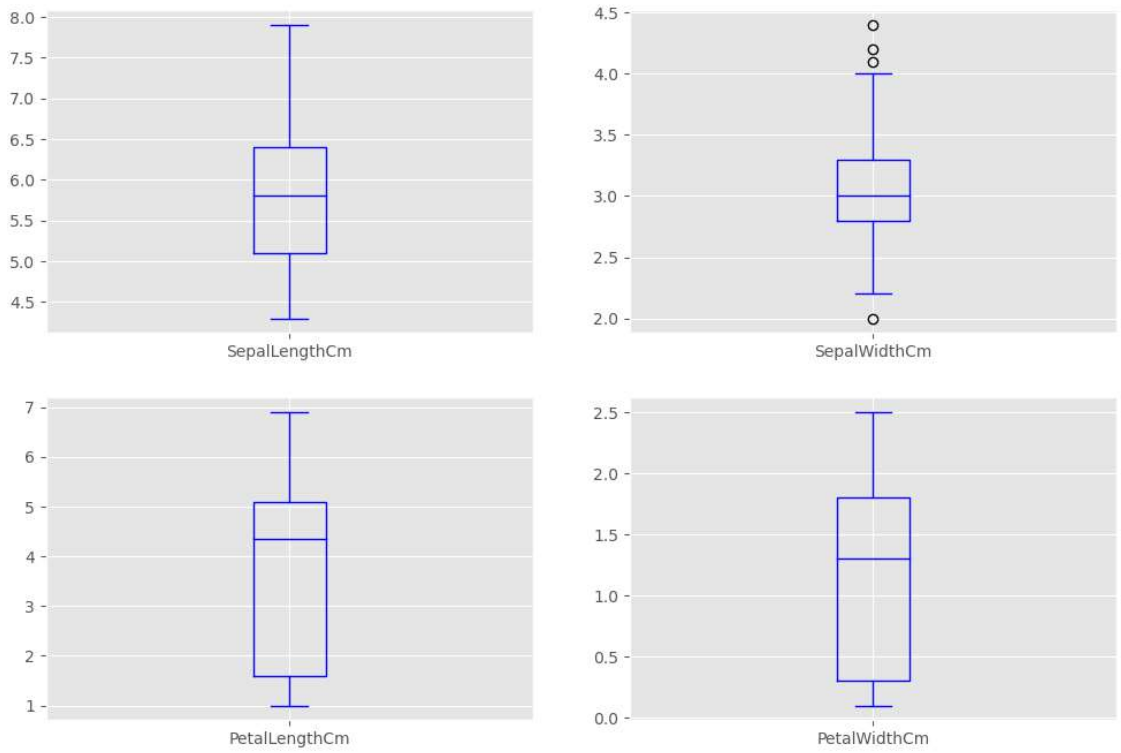


create a box plot

#This code creates a 2x2 grid of the box plot for each column of the DataFrame 'df'. Each subplot represents a statistical summary (such as median, quartiles, and outliers) of the distribution of a particular variable. Plots are shown by individual subplots, not by sharing an x-

axis or y-axis. The box image is shown in blue, and the overall size of the image is set to (12,8)

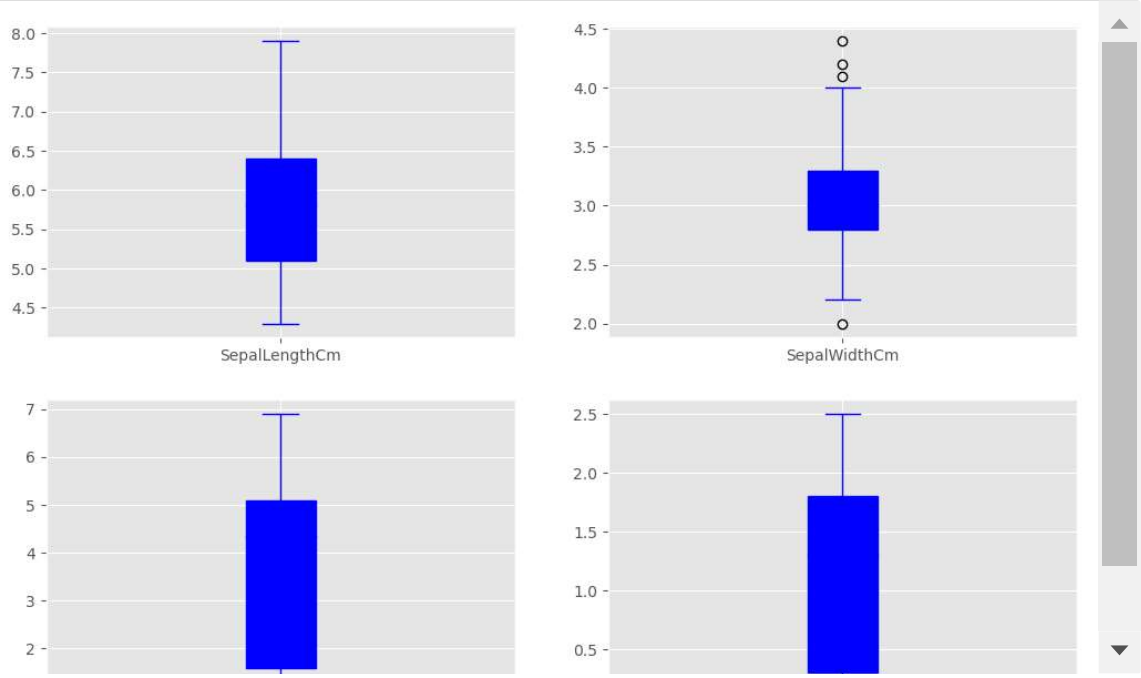
In [31]:
```
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=Fals
plt.show()
```



fill the boxes with color, using patch_artist

#This code creates a 2x2 grid of the box plot for each color of the DataFrame 'df'. Each subplot represents a statistical summary (such as median, quartiles, and outliers) of the distribution of a particular variable. Box plots are shown by individual subplots, not by sharing an x-axis or y-axis. Box plots are filled with color using the 'patch_artist' parameter, resulting in a visual representation of the data distribution. The overall size of the image is set to (12,8) for better visibility.

In [32]:
```
df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=Fals
plt.show()
```

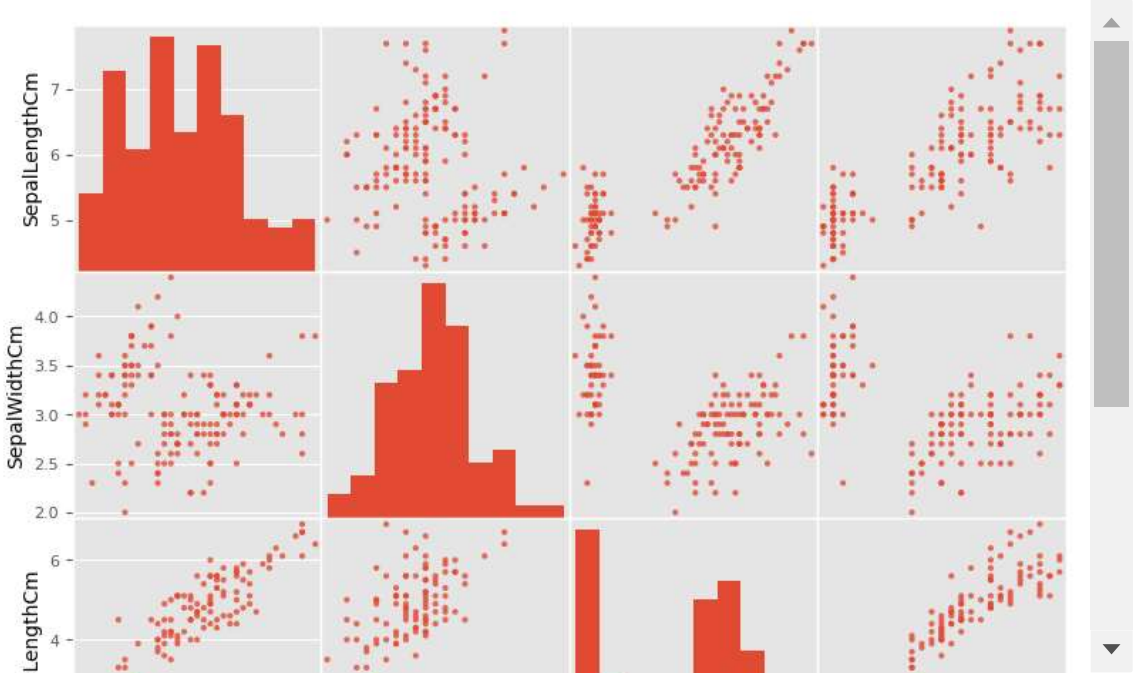


create a scatter matrix plot

alpha = Amount of transparency applied

#This code creates a scatter matrix plot, also known as a pair plot, for all variables in DataFrame 'df'. Each scatter plot represents the relationship between two variables, and the figure shows this plot for all combinations of variables. The 'alpha' parameter controls the clarity of the data points, and the overall image size is set to (9,9) for better visibility.

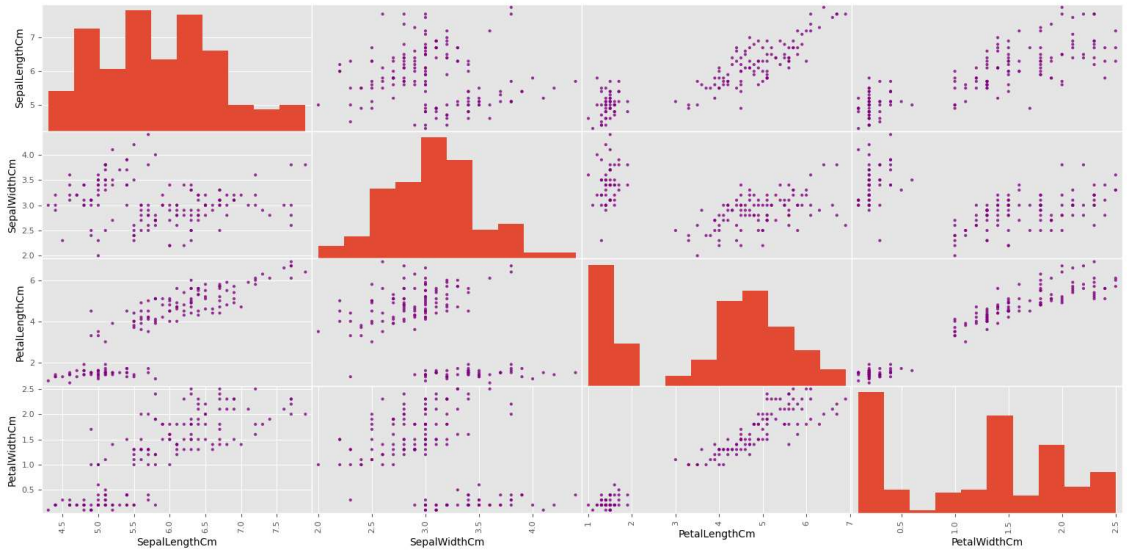
In [33]:
```
scatter_matrix (df, alpha=0.8, figsize=(9,9))
plt.show()
```



change the color to purple. Notice only one part of the plot changes.

#This code creates a scatter matrix plot, also known as a pair plot, for all variables in DataFrame 'df'. Each scatter plot represents the relationship between two variables, and the figure shows this plot for all combinations of variables. The ‘Alpha’ parameter controls the transparency of the data points, the overall image size is set to (19,9), and the data points are colored purple for better visualization.

In [35]:
```
scatter_matrix (df, alpha=0.8, figsize=(19,9), color = 'purple')
plt.show()
```



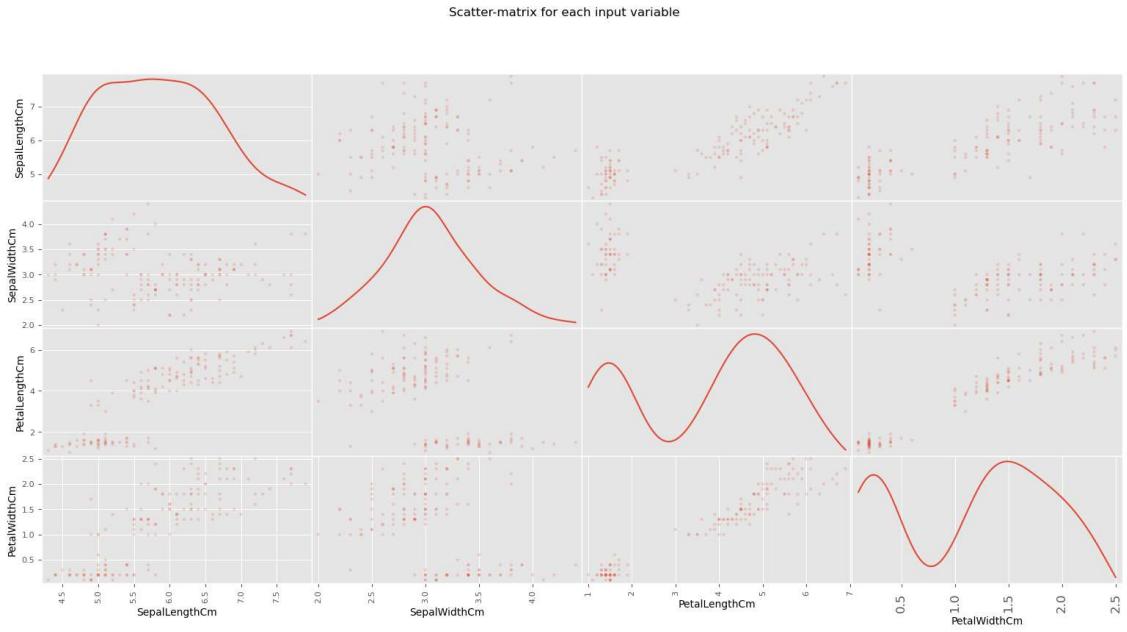
Here you are adding a suptitle.

Pick between ‘kde’ and ‘hist’ for either Kernel Density Estimation or Histogram plot in the diagonal

KDE = a density estimator is an algorithm which seeks to model the probability distribution that generated a dataset

#This code creates a scatter matrix plot with a kernel density estimation plot (KDE) on the diagonal. The ‘alpha’ parameter controls the transparency of the data points, the overall image size is set to (19,9), and a subtitle is added to the plot. Tick labels have also been adjusted for better readability.

In [37]:
```
scatter_matrix(df,alpha=0.2, diagonal = 'kde', figsize=(19,9))
plt.suptitle('Scatter-matrix for each input variable')
plt.tick_params(labelsize=12, pad=6)
```

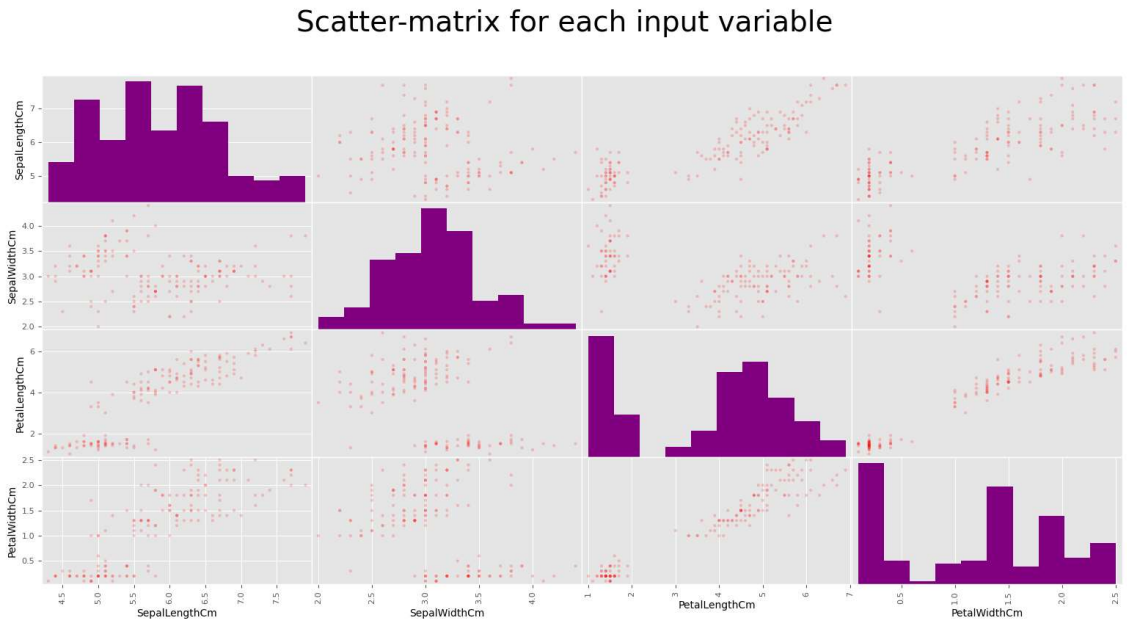


Lastly, you are changing the color to both parts of the plot.

#This code creates a scatter matrix plot with a customized figure size of (19,9). The data points are displayed with low transparency (alpha=0.2) and a red color. The histogram plots within the scatter matrix have a purple color. A suptitle is added with a larger font size of 28 for better visibility.

In [39]:
```
scatter_matrix(df, figsize= (19,9), alpha=0.2,
c='red', hist_kwds={'color':['purple']})
plt.suptitle('Scatter-matrix for each input variable', fontsize=28)
```

Out[39]: Text(0.5, 0.98, 'Scatter-matrix for each input variable')



Part 2

This code imports necessary libraries for data manipulation, visualization, and machine learning. It includes pandas for data handling, numpy for numerical computations, seaborn for enhanced visualizations, and sklearn for linear regression modeling and cross-validation. Matplotlib is imported for additional plotting capabilities.

In [42]:
```
import pandas as pd
import numpy as np

from pandas.plotting import scatter_matrix
from sklearn.linear_model import LinearRegression

from sklearn.model_selection import train_test_split

from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score



import seaborn as sns
sns.set(color_codes=True)

import matplotlib.pyplot as plt
```

The line of code is assigning the string "housing boston.csv" to the variable housingfile. It is used to load data into a DataFrame via pd.read_csv() and save the file name or file path of the data set for later use.

In [44]:
```
housingfile = ("housing boston.csv")
```

Load the data into a Pandas DataFrame

#The code is using the pd.read_csv() function to read the data from the file specified by the housingfile variable and store it in a DataFrame called df. The header=None parameter indicates that the CSV file does not have a header row, so the column names will be automatically assigned as integers.

In [47]:
```
df= pd.read_csv (housingfile, header=None)
```

Print the first few rows of the data

The code displays the first few lines of the DataFrame df using the head() function. It provides an overview of the data, showing the column values of the first five rows and their corresponding values

In [48]:
```
df.head()
```

Out[48]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

give names to the columns

#The code creates a list called col_names that contains the column names of the data set. It assigns a specific name to each column, which represents different features or variables in the dataset.

In [49]:
```
col_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX','RM', 'AGE', 'DIS', 'RAD
```

Let's check to see if the column names were added

#The code provides a list of column names called col_names for the columns of the DataFrame df. This successfully converts the name of the DataFrame's column to a specified name.

In [51]:
```
df.columns = col_names
```

Look at the first 5 rows of data

In [53]:
```
df.head()
```

Out[53]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	AA	I
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	

returns the number of missing values in the dataset.

We see there are no missing data points

#The code is calculating the sum of null values for each column in the DataFrame df. It provides an overview of missing values in each column of the dataset.

In [54]:
```
df.isnull().sum()
```

Out[54]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
DIS        0
RAD        0
TAX        0
PTRATIO    0
AA         0
LSTAT      0
MEDV       0
dtype: int64
```

Print the shape of the data.

Get the number of records/rows, and the number of variables/columns

#The code prints the size of the DataFrame df, which returns the number of rows and columns in the data set. It provides the dimensions of the data set.

In [55]:
```
print(df.shape)
```

```
(452, 14)
```

Print the data types of each column

Get the data types of all variables

#The code prints the data type of each column in the DataFrame df. Returns information about the type of data stored in each column, such as an integer, float, or object.

In [56]:
```
print(df.dtypes)
```

```
CRIM       float64
ZN         float64
INDUS      float64
CHAS         int64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD          int64
TAX          int64
PTRATIO    float64
AA         float64
LSTAT      float64
MEDV       float64
dtype: object
```

Obtain the summary statistics of the data

#The code provides descriptive statistics for the DataFrame df. It provides statistical summary measures such as mean, median, standard deviation, minimum, maximum, and quartile for each statistical column in the DataFrame.

In [57]:
```
print(df.describe())
```

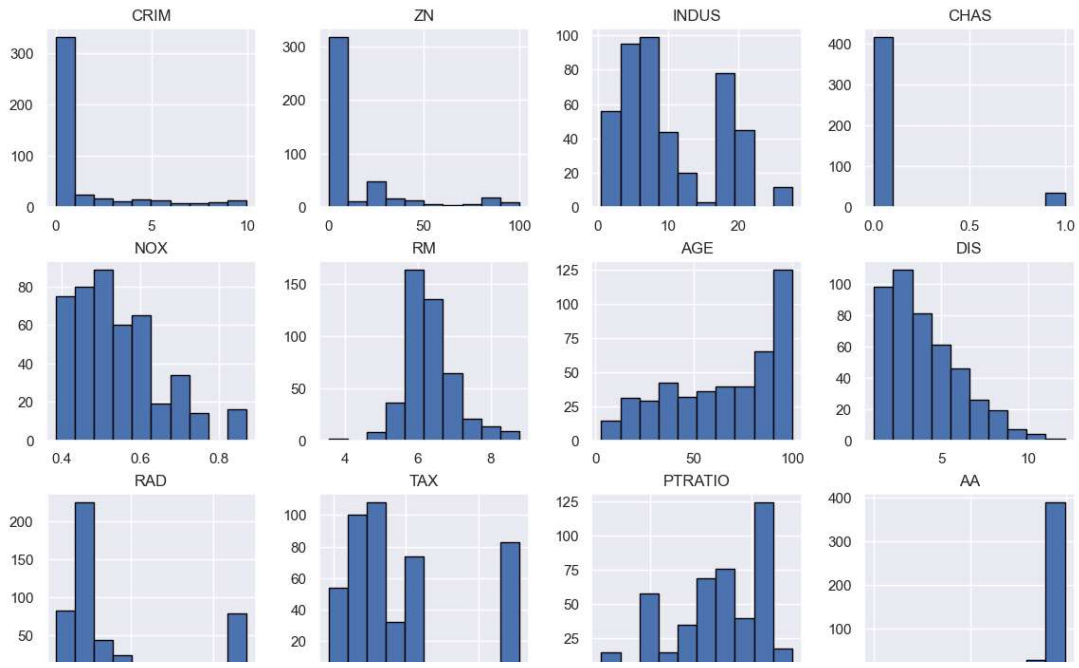
```
             CRIM          ZN       INDUS        CHAS         NOX
RM  \
count  452.000000  452.000000  452.000000  452.000000  452.000000  45
2.000000
mean     1.420825   12.721239   10.304889    0.077434    0.540816
6.343538
std      2.495894   24.326032    6.797103    0.267574    0.113816
0.666808
min      0.006320    0.000000    0.460000    0.000000    0.385000
3.561000
25%      0.069875    0.000000    4.930000    0.000000    0.447000
5.926750
50%      0.191030    0.000000    8.140000    0.000000    0.519000
6.229000
75%      1.211460   20.000000   18.100000    0.000000    0.605000
6.635000
max      9.966540  100.000000   27.740000    1.000000    0.871000
8.780000

             AGE         DIS         RAD         TAX     PTRATIO
```

Plot histogram for each variable

#The code generates histograms for each variable in the DataFrame df. It allows to see the distribution of objects in each color, and shows the frequency or count of values in certain areas.

In [58]:
```
df.hist(edgecolor= 'black',figsize=(14,12))
plt.show()
```



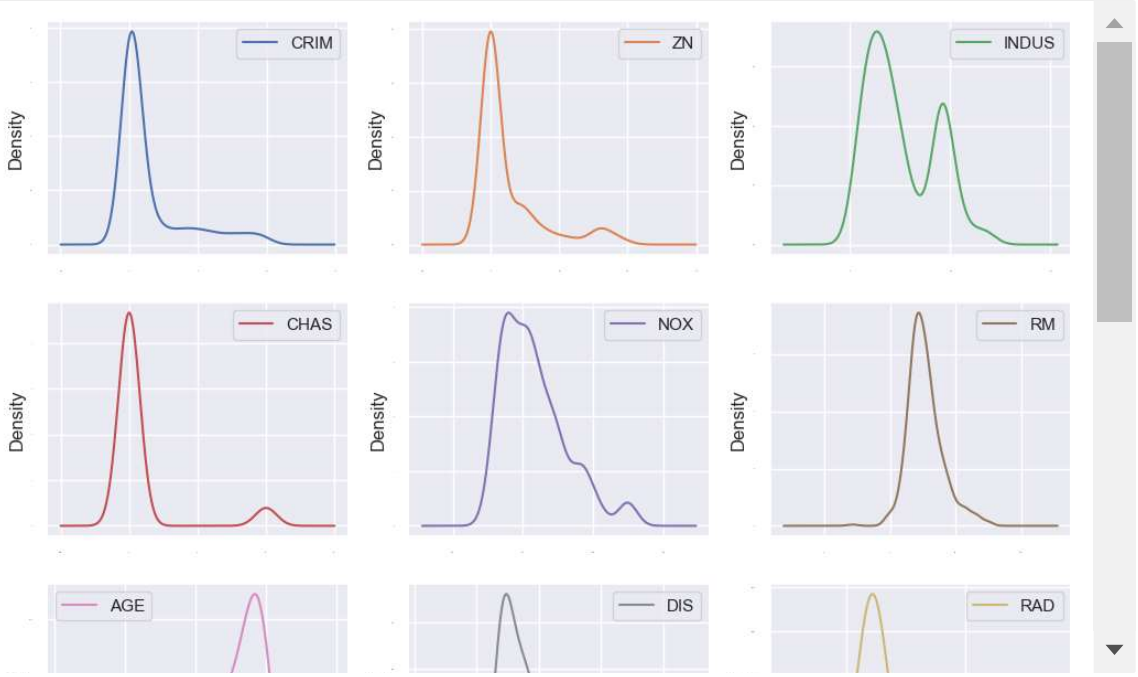
Density plots

Notes: 14 numeric variable, at least 14 plots, layout (5,3): 5 rows, each row with 3 plots

When subplots have a shared x-axis along a column, only the x tick labels of the bottom subplot are created

#The code creates a density plot for each variable in the DataFrame df. It displays the probability density function of each variable, and shows the distribution of values as a simple curve. The plots are arranged in a 5x3 grid format.

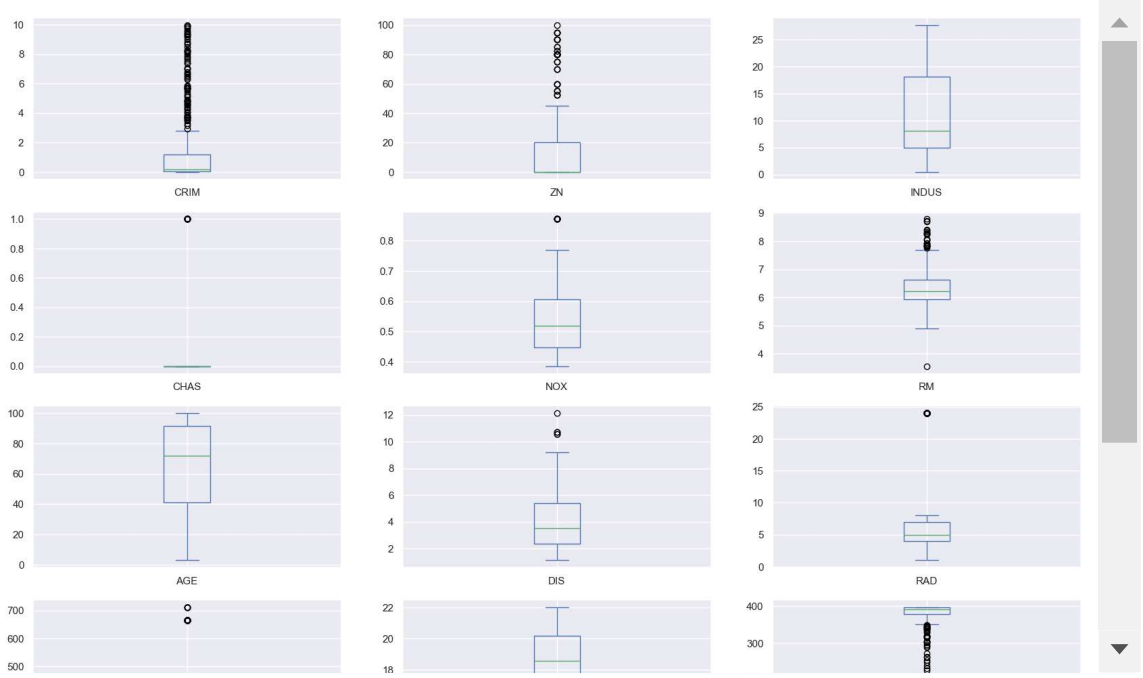
In [59]:
```
df.plot(kind='density', subplots=True, layout= (5,3), sharex=False, legend
plt.show()
```



Boxplots

#The code creates box plots for each variable in the DataFrame df. It identifies the distribution of values between variables, and identifies possible medians, quartiles, and outliers. The plots are arranged in a 5x3 grid format.

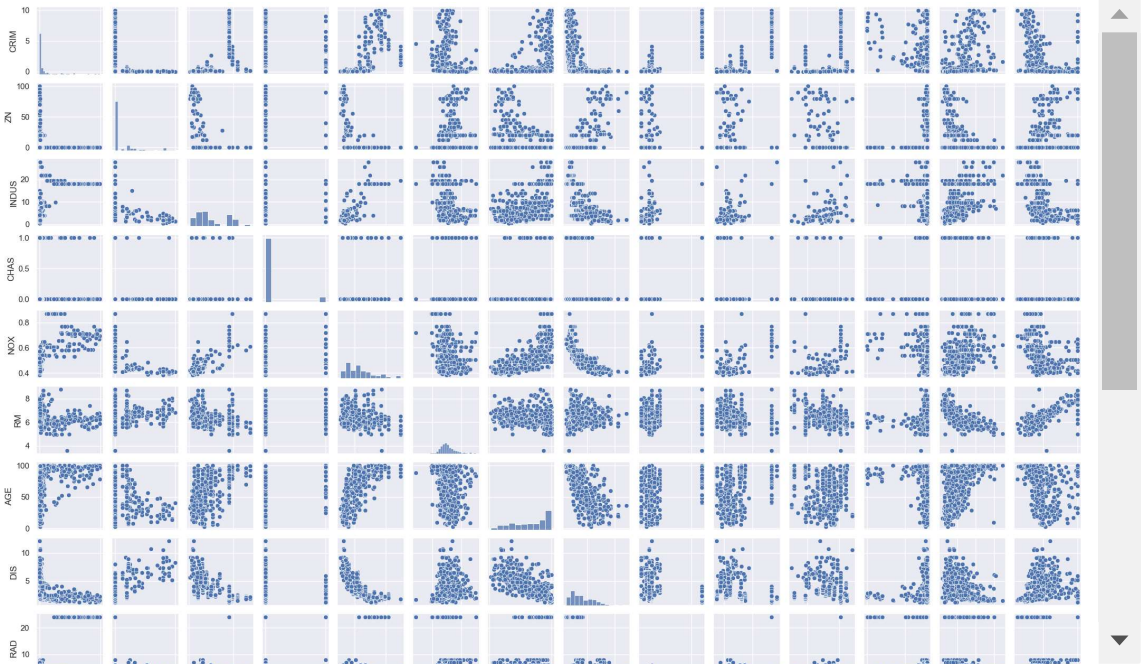
In [63]:
```
df.plot(kind="box", subplots=True, layout=(5,3), sharex=False, figsize=(20
plt.show()
```



Obtain pair plots of the data

#The code uses the Seaborn pairplot function to construct a pairwise scatter plot matrix. It reveals the relationship between pairs of variables in the DataFrame df. Each scatter plot represents the relationship between two variables, and the diagonal plot shows the distribution of the individual variables.

In [64]:
```
sns.pairplot(df, height=1.5);
plt.show()
```



We will decrease the number of decimal places with the format function.

#The code is setting the display format for floating-point numbers in the Pandas library. It formats the numbers to have three decimal places when they are displayed. This affects how the numbers are shown in the output of Pandas operations, such as printing DataFrames or summary statistics.

In [65]:
```
pd.options.display.float_format = '{:,.3f}'.format
```

Here we will get the correlations, with only 3 decimals.

#The code is calculating the correlation between the columns in the DataFrame df. It returns the correlation matrix, which shows the correlation coefficient between all pairs of columns in the DataFrame. This helps to determine the strength and direction of the linear relationship between the variables.

In [66]:
```
df.corr()
```

Out[66]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PT
CRIM	1.000	-0.281	0.574	0.050	0.637	-0.142	0.448	-0.462	0.898	0.826	
ZN	-0.281	1.000	-0.514	-0.060	-0.501	0.307	-0.556	0.656	-0.267	-0.269	
INDUS	0.574	-0.514	1.000	0.103	0.739	-0.365	0.606	-0.669	0.513	0.673	
CHAS	0.050	-0.060	0.103	1.000	0.134	0.077	0.123	-0.141	0.057	0.017	
NOX	0.637	-0.501	0.739	0.134	1.000	-0.265	0.707	-0.746	0.542	0.615	
RM	-0.142	0.307	-0.365	0.077	-0.265	1.000	-0.188	0.139	-0.096	-0.215	
AGE	0.448	-0.556	0.606	0.123	0.707	-0.188	1.000	-0.720	0.359	0.427	
DIS	-0.462	0.656	-0.669	-0.141	-0.746	0.139	-0.720	1.000	-0.388	-0.444	
RAD	0.898	-0.267	0.513	0.057	0.542	-0.096	0.359	-0.388	1.000	0.873	
TAX	0.826	-0.269	0.673	0.017	0.615	-0.215	0.427	-0.444	0.873	1.000	
PTRATIO	0.319	-0.364	0.317	-0.100	0.103	-0.334	0.193	-0.152	0.387	0.385	

We could simply look at the correlations but a heatmap is a great way to present.

In [67]:
```
plt.figure(figsize =(16,10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```

If you get stuck on what can be done with the heatmap, you can use the following code to get help

#sns.heat map ? code is used to obtain documentation or help information for the sns.heatmap() function in seaborn. It provides detailed descriptions of the application parameters, functions, and examples, helping users understand how to use the application properly and customize the heatmap diagram according to their needs

In [68]:
```
sns.heatmap?
```

Now let's say we want to decrease the amount of variables in our heatmap.

Remember how to make a subset. Try using different variables.

#The code df2 = df[['CRIM', 'INDUS', 'TAX', 'MEDV']] creates a new DataFrame df2 by selecting the exact column from the original DataFrame df. It simply provides a subset of the DataFrame with the characters 'CRIM', 'INDUS', 'TAX', and 'MEDV', and assigns it to the df2 variable.

In [69]:
```
df2= df[['CRIM','INDUS', 'TAX','MEDV']]
```

Here we will look at the correlations for only the variables in df2.

#The code df2.corr() calculates the correlation matrix of variables in DataFrame df2 calculates pairwise correlations between columns and examines the strength of correlations and correlations between variables.

In [70]:
```
df2.corr()
```

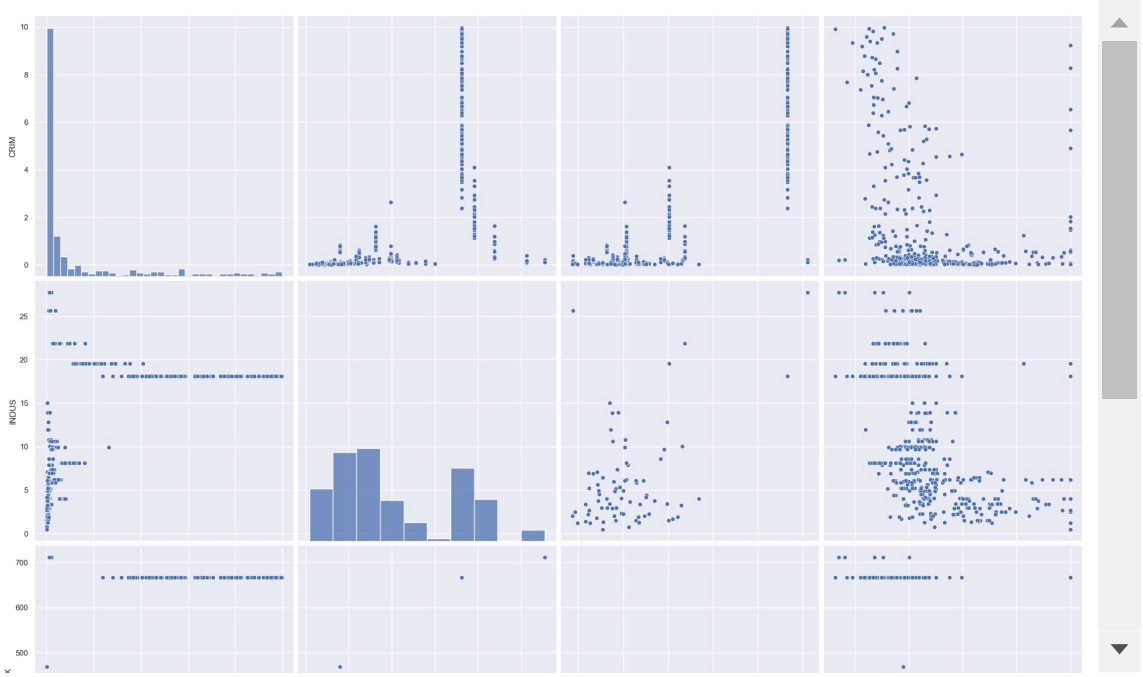
Out[70]:

	CRIM	INDUS	TAX	MEDV
CRIM	1.000	0.574	0.826	-0.286
INDUS	0.574	1.000	0.673	-0.412
TAX	0.826	0.673	1.000	-0.346
MEDV	-0.286	-0.412	-0.346	1.000

Let's try the pairplot with only the variables in df2

#The code sns.pairplot(df2, height=5.5) generates the pairwise scatter plot matrix for the variables in DataFrame df2. It plots each variable against each other and highlights the relationships between variables, allowing for a quick visualization of their distribution and correlations

In [72]:
```
sns.pairplot(df2, height=5.5);
plt.show()
```

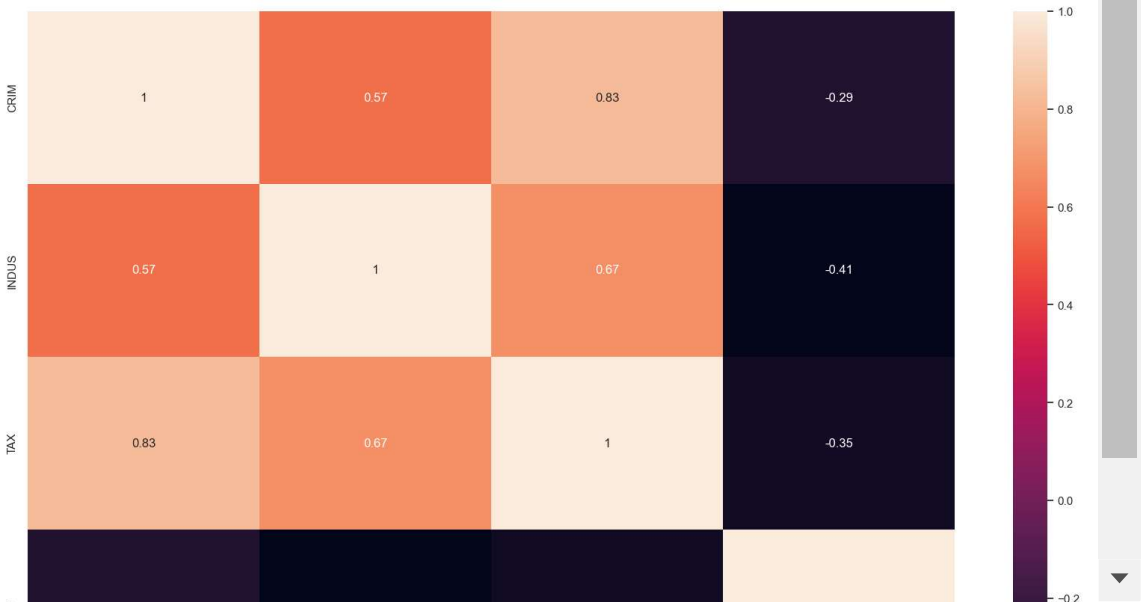


Now we will make a heatmap with only the variables in df2 subset. Again, it is very important to understand this for the final.

#The code plt.figure(figsize=(20,12)); sns.heatmap(df2.corr(), annot=True) creates a heatmap of the correlation matrix for the variables in DataFrame df2. It visualizes the strength and direction of the correlations between the variables using colors and annotated values, allowing for a clear understanding of the relationships between the variables.

In [74]:
```
plt.figure(figsize =(20,12))
plt.figure(figsize = (20,12))
sns.heatmap(df2.corr(), annot=True)
plt.show()
```

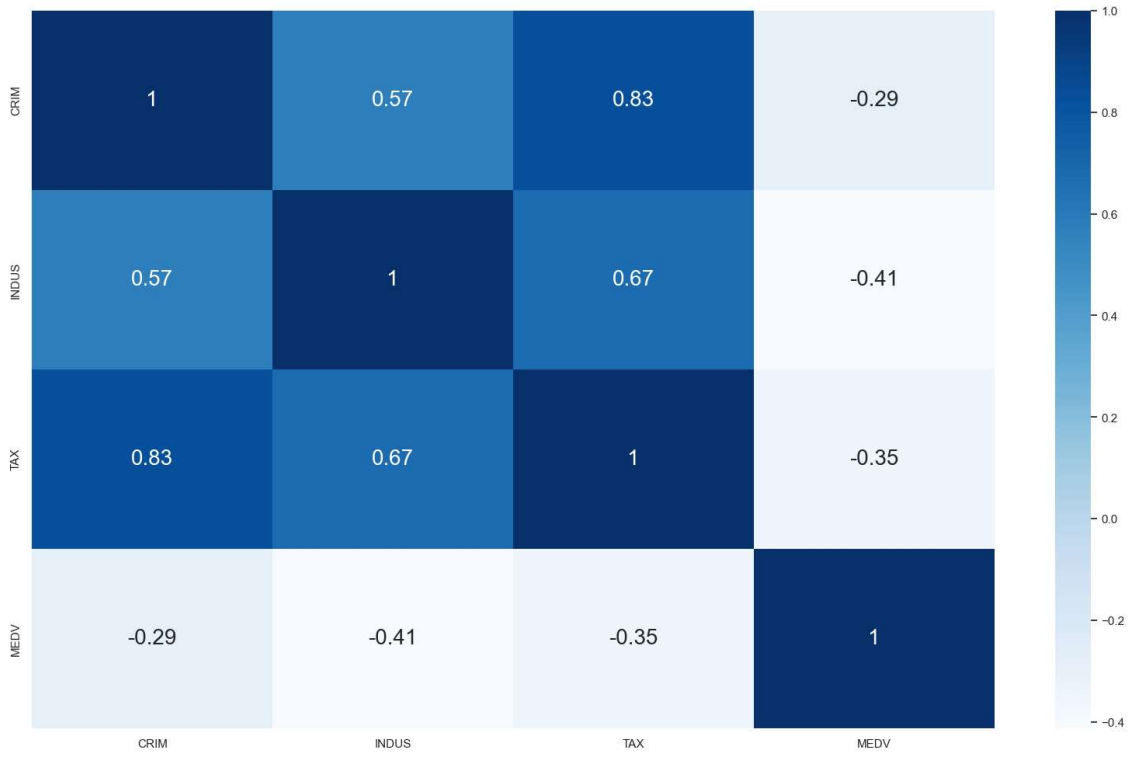
```
<Figure size 2000x1200 with 0 Axes>
```



If you want to change the color and font, to make the labels easier to read, use this code.

#The code plt.figure(figsize=(20,12)); sns.heatmap(df2.corr(), cmap="Blues", annot=True, annot_kws={"fontsize":20}) creates a heatmap of the correlation matrix for the variables in DataFrame df2. It uses a blue color map (cmap="Blues") and displays the correlation values as annotations with a font size of 20 (annot=True, annot_kws={"fontsize":20}). This code provides a visually appealing and informative representation of the variable correlations.

In [75]:
```
plt.figure(figsize =(20,12))
sns.heatmap(df2.corr(), cmap="Blues", annot=True, annot_kws={"fontsize":20
plt.show()
```



Importing train_test_split function for data splitting.

#The code from sklearn.model_selection import train_test_split imports the train_test_split function from the sklearn.model_selection module. This function is commonly used to split a dataset into training and testing subsets, allowing for the evaluation of machine learning models on unseen data.

In [76]:
```
from sklearn.model_selection import train_test_split
```

Store the dataframe values into a numPy array

#Converting the DataFrame values into a NumPy array and then separating the array into input (X) and output (Y) components. X contains all rows and columns from 0 to 2, while Y contains all rows in the last column (MEDV).

In [78]:
```
array= df2.values

# For X (input) [:,3] --> All the rows and columns from 0 up to 3

X = array [:, 0:3]

# For Y (output) [:3] --> All the rows in the last column (MEDV)

Y = array [:,3]
```

Split the dataset --> training sub-dataset: 67%, and test sub-dataset: 33%

#dividing the data set into smaller ones for training and testing. The training dataset contains 67% of the data, while the testing dataset contains 33%. The random seed is set to 7 in order to ensure consistent randomization and to ensure reproducibility.

In [79]:
```
test_size = 0.33

# Selection of records to include in which sub-dataset must be done random

seed = 7

# Split the dataset (both input & output) into training/testing datasets
# if random_state = None : Calling the function multiple times will produc
# if random_state = Integer : Will produce the same results across differe

X_train, X_test, Y_train, Y_test= train_test_split(X,Y, test_size=0.2, ran
```

#Creating a linear regression model object and fitting it to the training data. The model is then trained using the training subset. The intercept and coefficients of the linear regression model are printed, representing the estimated intercept and coefficients for the input variables.

In [80]:
```
# Build the model

model=LinearRegression()

# Train the model using the training sub-dataset

model.fit(X_train, Y_train)

#Print out the coefficients and the intercept
# Print intercept and coefficients
# are the variables statistically significant
# interdept = mean (average) value of Y
# if the value is less than 0.05: there is a strong relationship between t

print ("Intercept:", model.intercept_)
print ("Coefficients:", model.coef_)
```

```
Intercept: 31.393427670412983
Coefficients: [ 0.09859287 -0.42388844 -0.00931847]
```

#Creating a list of variable names corresponding to the coefficients of the linear regression model. The coefficients and variable names are paired together using the zip function. The resulting pairs are converted into a set and then printed out in a loop.

In [81]:
```
# If we want to print out the list of the coefficients with their correspo
# Pair the feature names with the coefficients
names_2 = ["CRIM", "INDUS", "TAX"]

coeffs_zip = zip(names_2, model.coef_)

# Convert iterator into set

coeffs = set(coeffs_zip)

# Print (coeffs)

for coef in coeffs:
    print (coef, "\n")
```

```
('INDUS', -0.4238884417716139)

('CRIM', 0.09859287239144014)

('TAX', -0.009318474474503551)
```

#The code initializes an instance of the LinearRegression class using the specified parameters: copy_X=True, fit_intercept=True, and n_jobs=1. It is a linear regression model that performs a linear regression analysis on the given data, where copy_X indicates whether the input data will be copied, fit_intercept indicates whether it will fit the intercept term, and n_jobs indicates the number of parallel jobs to use for the calculation.

In [82]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1)
```

Out[82]:
```
LinearRegression(n_jobs=1)
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

#The score method is used in this code to determine the R-squared value of the linear regression model. The percentage of the variance in the dependent variable (Y) that can be explained by the independent variables (X) is indicated by the R-squared value. After that, the calculated R-squared value is displayed.

In [83]:
```
R_squared = model.score(X_test, Y_test)
print("R-squared: ", R_squared)
```

```
R-squared:  0.15473313373590336
```

#The code is using the trained linear regression model to make a prediction on a new input data point with the features [12, 10, 450]. The predict method of the model is used to obtain the predicted output value based on the provided input features.

In [84]:
```
model.predict([[12,10,450]])
```

Out[84]:
```
array([24.14434421])
```

#The code is evaluating the performance of the linear regression algorithm using k-fold cross-validation. It splits the data into k subsets, trains the model on each subset, and evaluates the model using the negative mean squared error (MSE) as the scoring metric. The average of all the negative MSE values obtained from the k-fold cross-validation is then printed as the evaluation result.

In [85]:

```
# Evaluate the algorithm
# Specify the K-size

num_folds = 10

# Fix the random seed
# must use the same seed value so that the same subsets can be obtained
# for each time the process is repeated

seed = 7

# Split the whole data set into folds

kfold= KFold(n_splits=num_folds, random_state=seed, shuffle=True)

# For Linear regression, we can use MSE (mean squared error) value
# to evaluate the model/algorithm

scoring = 'neg_mean_squared_error'

# Train the model and run K-foLd cross-validation to validate/evaluate the

results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)

# Print out the evaluation results
# Result: the average of all the results obtained from the k-fold cross va

print("Average of all results from the K-fold Cross Validation, using nega
```

```
Average of all results from the K-fold Cross Validation, using negative m
ean squared error: -64.35862748210982
```

#The code is evaluating the performance of the linear regression algorithm using k-fold cross-validation. It splits the data into k subsets, trains the model on each subset, and evaluates the model using the explained variance as the scoring metric. The average of all the explained variance values obtained from the k-fold cross-validation is then printed as the evaluation result.

In [86]:
```
# Evaluate the algorithm
# Specify the K-size

num_folds = 10

# Fix the random seed must use the same seed value so that the same subset
# for each time the process is repeated

seed = 7

# Split the whole data set into folds

kfold= KFold(n_splits=num_folds, random_state=seed, shuffle=True)

# For Linear regression, we can use explained variance value to evaluate t

scoring = 'explained_variance'

# Train the model and run K-foLd cross-validation to validate/evaluate the

results = cross_val_score(model, X, Y, cv=kfold, scoring=scoring)

# Print out the evaluation results
# Result: the average of all the results obtained from the k-fold cross va

print("Average of all results from the K-fold Cross Validation, using exlp
```

```
Average of all results from the K-fold Cross Validation, using exlpained
variance: 0.19023822025958692
```

Part 3

Import Python Libraries: NumPy and Pandas

#The code is importing the Python libraries Pandas and NumPy. Pandas is used for data manipulation and analysis, while NumPy is used for numerical computations and array operations.

In [87]:
```
import pandas as pd
import numpy as np
```

Import Libraries & modules for data visualization

#The code is importing the necessary libraries for data visualization. scatter_matrix is used to create scatter plots, matplotlib.pyplot is used for plotting, and seaborn is used for enhancing the visual aesthetics of plots.

In [88]:
```
from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

Import scikit-Learn module for the algorithm/modeL: Logistic Regression

#The code is importing the Logistic Regression algorithm/model from the scikit-learn library. This algorithm/model can be used for classification tasks.

In [89]:
```
from sklearn.linear_model import LogisticRegression
```

Import scikit-Learn module to split the dataset into train/ test sub-datasets

#The code is importing the train_test_split function from the scikit-learn library. This function is used to split the dataset into training and testing subsets for model evaluation and validation purposes.

In [90]:
```
from sklearn.model_selection import train_test_split
```

Import scikit-Learn module for K-fold cross-validation - algorithm/modeL evaluation & validation

#The code is importing the KFold and cross_val_score functions from the scikit-learn library. These functions are used for K-fold cross-validation, which is a technique for evaluating and validating machine learning models by splitting the data into multiple subsets (folds) and iteratively training and testing the model on different combinations of these folds.

In [91]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

Import scikit-Learn module classification report to later use for information about how the system

#try to classify/lable each record #The code is importing the classification_report function from scikit-learn. This function is used to generate a classification report that provides information about the classification performance of a model, including metrics such as precision, recall, and F1 score.

In [92]:
```
from sklearn.metrics import classification_report
```

#The code below is specifying the location of the dataset file "iris.csv" and storing it in the variable "filename". Then, it uses the pandas library to read the CSV file and load the data into a DataFrame called "df".

In [93]:
```
# Specify location of the dataset

filename = '../data/iris.csv'

# Load the data into a Pandas DataFrame

df = pd.read_csv("C:\\Users\\vijay\\Downloads\\Iris.csv")
```

Print the first few rows of the data

#The code is displaying the first few rows of the DataFrame "df", providing a preview of the loaded dataset.

In [94]:
```
df.head()
```

Out[94]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.100	3.500	1.400	0.200	Iris-setosa
1	2	4.900	3.000	1.400	0.200	Iris-setosa
2	3	4.700	3.200	1.300	0.200	Iris-setosa
3	4	4.600	3.100	1.500	0.200	Iris-setosa
4	5	5.000	3.600	1.400	0.200	Iris-setosa

#The code replaces the zero values in the columns 'SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', and 'PetalWidthCm' with NaN (missing) values. Then, it counts the number of NaN values in each column and prints the result.

In [96]:
```
# mark zero values as missing or NaN

df[[ 'SepalLengthCm' , 'SepalWidthCm' , 'PetalLengthCm' ,'PetalWidthCm' ]]
= df[['SepalLengthCm' , 'SepalWidthCm' ,'PetalLengthCm' , 'PetalWidthCm' ]

# count the number of NaN values in each column
# returns the number of missing values in the dataset.
print (df.isnull().sum())
```

```
Id              0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

get the dimensions or shape of the dataset

i.e. number of records / rows X number of variables / columns

#The code prints the shape of the dataset, indicating the number of rows (records) and columns (variables) in the dataset.

In [97]:
```
print("Shape of the dataset(rows, columns):",df.shape)
```

```
Shape of the dataset(rows, columns): (150, 6)
```

get the data types of all the variables / attributes in the data set

#The code prints the data types of all the variables or attributes in the dataset, indicating the type of data stored in each column (e.g., integer, float, object).

In [100]:
```
print(df.dtypes)
```

```
Id                int64
SepalLengthCm    float64
SepalWidthCm     float64
PetalLengthCm    float64
PetalWidthCm     float64
Species           object
dtype: object
```

Print the summary statistics of the data

#return the summary statistics of the numeric variables/attributes in the data set #The code prints the data types of all the variables or attributes in the dataset, indicating the type of data stored in each column (e.g., integer, float, object)

In [101]:
```
print(df.describe())
```

```
              Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm
count 150.000        150.000       150.000        150.000       150.000
mean   75.500          5.843         3.054          3.759         1.199
std    43.445          0.828         0.434          1.764         0.763
min     1.000          4.300         2.000          1.000         0.100
25%    38.250          5.100         2.800          1.600         0.300
50%    75.500          5.800         3.000          4.350         1.300
75%   112.750          6.400         3.300          5.100         1.800
max   150.000          7.900         4.400          6.900         2.500
```

class distribution i.e. how many records are in each class

#The code groups the dataset by the 'Species' column and calculates the size of each group, indicating the number of records or instances belonging to each class or species. It provides the class distribution, showing the count of records in each class.

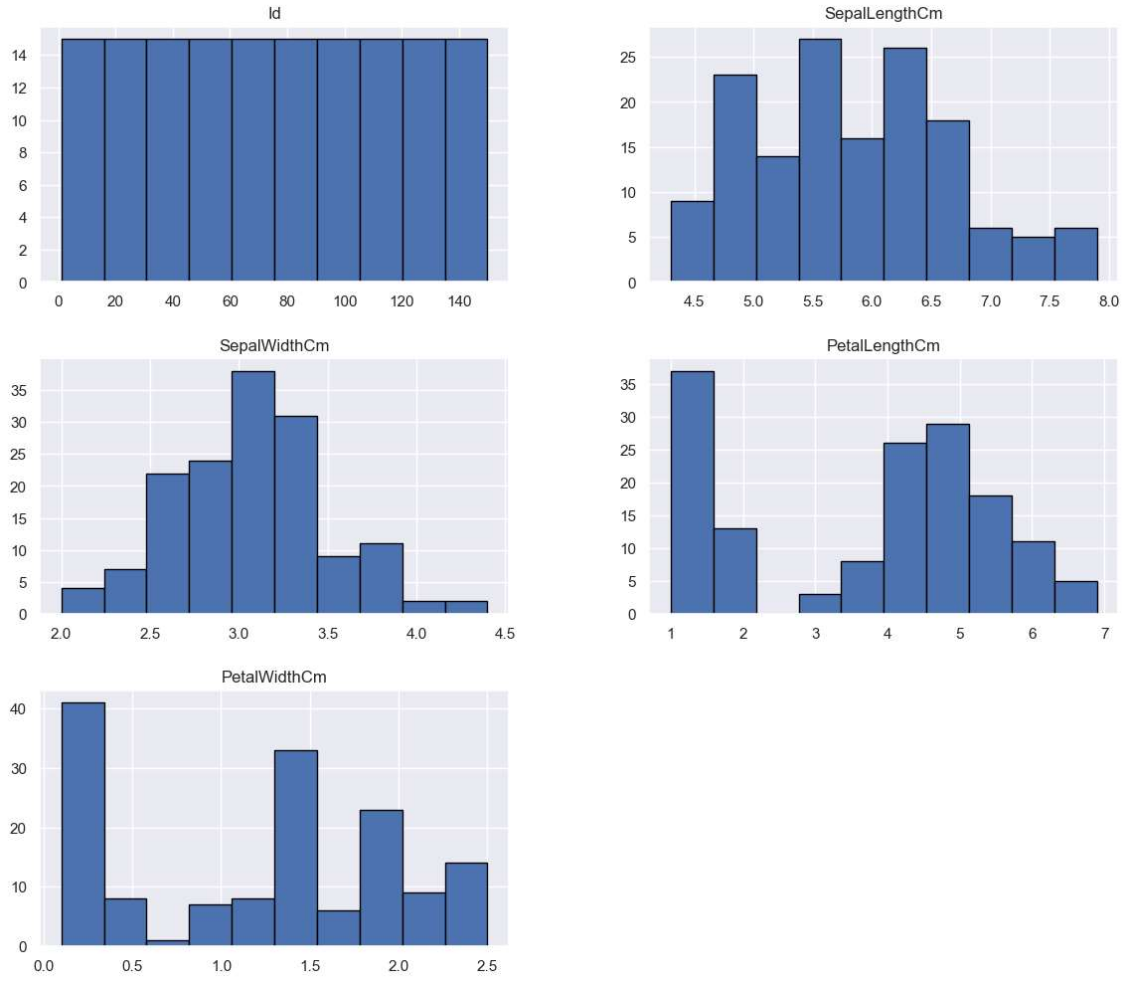
In [102]:
```
print(df.groupby('Species').size())
```

```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

Plot histogram for each variable.

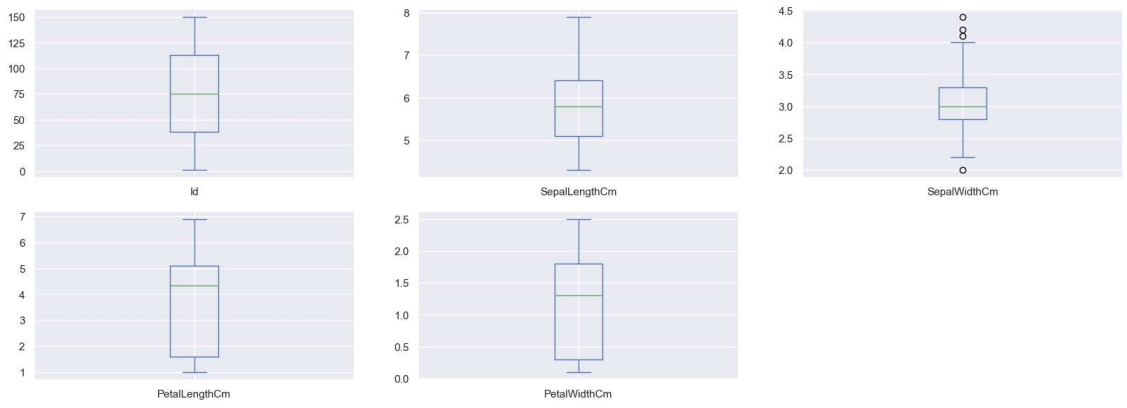
#Each variable in the data set has a histogram created by the code. It reveals how data are distributed across various variables, allowing the range, frequency, and size of the data to be understood. The histograms that generated are shown in the figure with the chosen edge colour and thickness.

In [104]:
```
df.hist(edgecolor= 'black',figsize=(14,12))
plt.show()
```



In [66]:
```
# Boxplots

df.plot(kind="box", subplots=True, layout=(5,3), sharex=False, figsize=(20
plt.show()
```

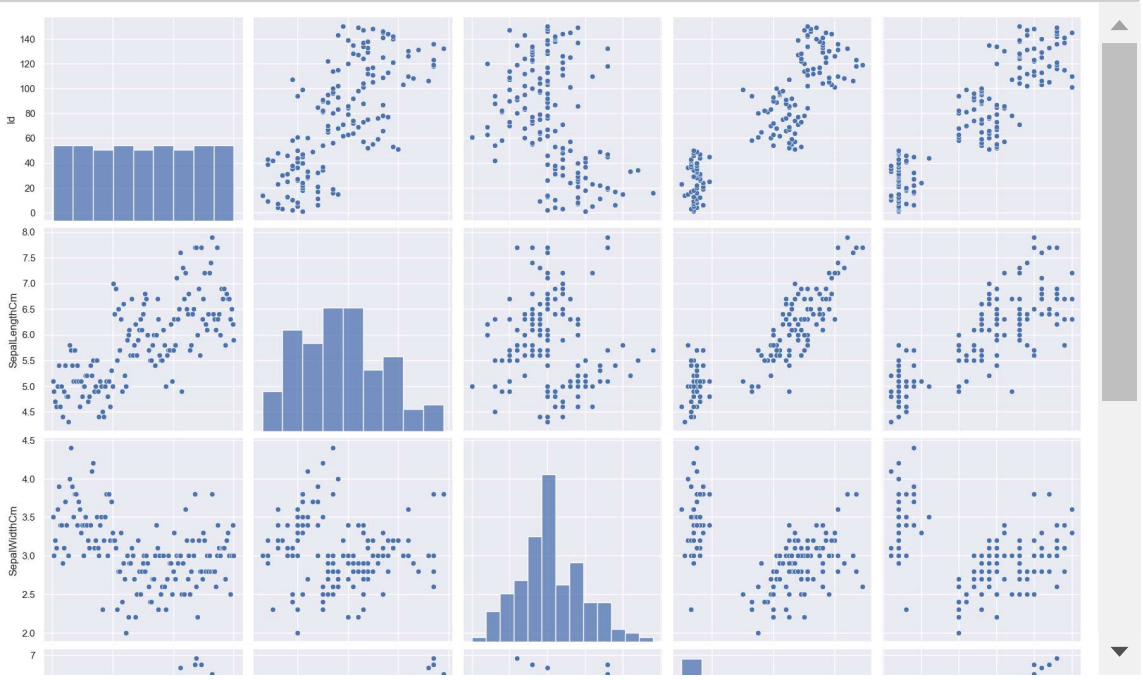


Please click on the above URL to learn more about Pair Plots

I know this is a lot of information but I wanted you to see what is possible with Seaborn library.

#The code is creating a pair plot using Seaborn library. It visualizes the pairwise relationships between variables in the dataset, showing scatter plots for each pair of variables along with histograms for each variable. The resulting plot helps us understand the correlations and distributions between different variables in the dataset.

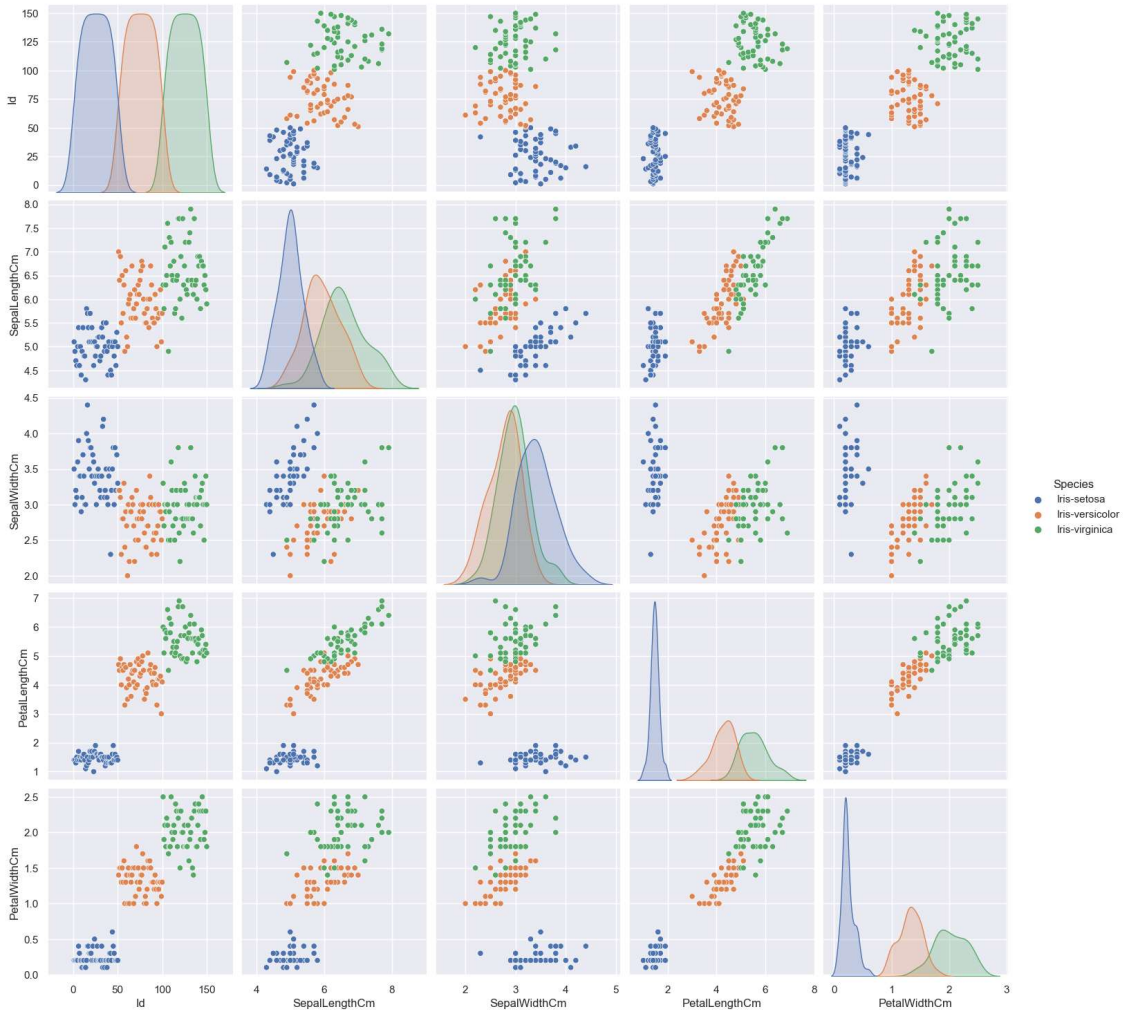
In [105]:
```
sns.pairplot(df, height=3.5);
plt.show()
```



Let's try that again using color.

#The code is creating a pair plot using Seaborn library with color-coded data points based on the 'Species' variable. It visualizes the pairwise relationships between variables, where each point represents an instance in the dataset, and the hue variable adds a semantic mapping to differentiate species. This plot helps us understand the relationships and distributions between variables for different species.

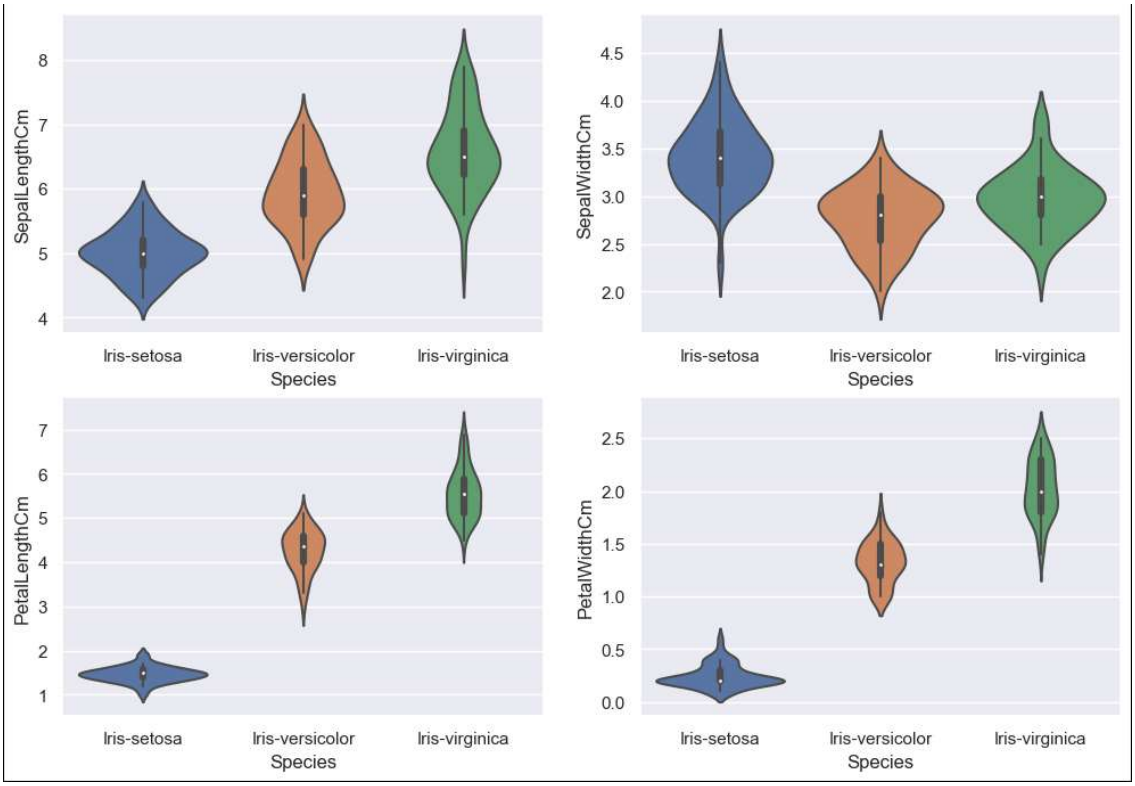
In [107]:
```
sns.pairplot(df, hue='Species', height=3, aspect= 1);
```



Please click on the URL above to learn more about Violin Plots

#The code is creating a 2x2 grid of violin plots using Seaborn library. Each subplot represents a different variable ('SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm') plotted against the 'Species' variable. Violin plots display the distribution of the data and provide insights into the distribution shape, median, and quartiles for each species.

In [108]:
```
plt.figure(edgecolor="black", linewidth= 1.2,figsize=(12,8));
plt.subplot(2,2,1)
sns.violinplot(x='Species', y = 'SepalLengthCm', data=df)
plt.subplot(2,2,2)
sns.violinplot(x='Species', y = 'SepalWidthCm', data=df)
plt.subplot(2,2,3)
sns.violinplot(x='Species', y = 'PetalLengthCm', data=df)
plt.subplot(2,2,4)
sns.violinplot(x='Species', y = 'PetalWidthCm', data=df);
```



#These lines of code are converting a DataFrame into a NumPy array and then separating the array into input (X) and output (Y) components. X contains all the rows and columns 1 to 4, representing the independent variables or predictors, while Y contains all the rows in column 5, representing the value we are trying to predict.

In [118]:
```
# store dataframe values into a numpy array

array = df.values

# separate array into input and output by slicing
# for X(input) [:, 1:5] --> all the rows, columns from 1 - 5
# these are the independent variables or predictors

X = array[:,1:5]

# for Y(input) [:, 5] --> all the rows, column 5
# this is the value we are trying to predict

Y = array[:,5]
```

#Here, these labels separate the training and test data sets from the input (X) and output (Y) data sets. 33 percent of the data will be used for the test set, and the remaining 67 percent will be used to train the model. The divisions are generated randomly in this instance using randomly chosen fertility seeds.

In [119]:
```
# split the dataset --> training sub-dataset: 67%; test sub-dataset: 33%

test_size = 0.33

#selection of records to include in each data sub-dataset must be done ran

seed = 7

#split the dataset (input and output) into training / test datasets

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=test_s
random_state=seed)
```

#These lines of code are building a logistic regression model using the LogisticRegression class from scikit-learn. The model is then trained using the training dataset (X_train and Y_train). Finally, the model's performance is evaluated by predicting the output values for the test dataset (X_test) and generating a classification report that includes metrics such as precision, recall, and F1-score.

In [120]:
```
#build the model

model = LogisticRegression(random_state=seed, max_iter=1000)

# train the model using the training sub-dataset

model.fit(X_train, Y_train)

#print the classification report

predicted = model.predict(X_test)
report = classification_report(Y_test, predicted)
print("Classification Report: ", "\n", "\n",report)
```

```
Classification Report:

                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        14
Iris-versicolor       0.89      0.89      0.89        18
 Iris-virginica       0.89      0.89      0.89        18

       accuracy                           0.92        50
      macro avg       0.93      0.93      0.93        50
   weighted avg       0.92      0.92      0.92        50
```

#The code is calculating the accuracy score of the trained logistic regression model on the test dataset. It then prints the accuracy percentage to evaluate the performance of the model in predicting the target variable.

In [122]:
```
#score the accuracy level

result = model.score(X_test, Y_test)

#print out the results

print(("Accuracy: %.3f%%") % (result*100.0))
```

```
Accuracy: 92.000%
```

#The code is using the trained logistic regression model to make predictions on a new input data point. It predicts the target class for the given input features [5.3, 3.0, 4.5, 1.5].

In [123]:
```
model.predict([[5.3, 3.0, 4.5, 1.5]])
```

Out[123]:
```
array(['Iris-versicolor'], dtype=object)
```

#The code is using the trained logistic regression model to predict the target class for a new input data point. It predicts the target class for the given input features [5, 3.6, 1.4, 1.5].

In [124]:
```
model.predict([[5, 3.6, 1.4, 1.5]])
```

Out[124]:
```
array(['Iris-setosa'], dtype=object)
```

The code is performing K-fold cross-validation on the logistic regression model using the specified number of splits and random seed. It evaluates the model's accuracy using the scoring metric and prints the average accuracy and standard deviation of the results obtained from the cross-validation process.

In [125]:
```
# Evaluate the algorithm and specify the number of times of repeated split

n_splits=10

#Fix the random seed.  You must use the same seed value so that the same s

seed=7

kfold=KFold(n_splits, random_state=seed, shuffle=True)

# for logistic regression, we can use the accuracy level to evaluate the m

scoring="accuracy"

#train the model and run K-fold cross validation to validate / evaluate th

results=cross_val_score (model, X, Y, cv=kfold, scoring=scoring)

# print the evaluation results.  The result is the average of all the resu

print("Accuracy: %.3f (%.3f)"% (results.mean(), results.std()))
```

```
Accuracy: 0.967 (0.054)
```

Type *Markdown* and LaTeX: α^2