Assignments

Rajesh Nemani(11509393)

EDA: Python Data Visualization with Matplotlib, Pandas, and NumPy

Imports

In [51]:
```
# Import all needed libraries

import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns

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

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

# Import scikit-Learn module for the algorithm/modeL: Logistic Regress
from sklearn.linear_model import LogisticRegression

# Import scikit-Learn module to split the dataset into train/ test sub
from sklearn.model_selection import train_test_split

# Import scikit-Learn module for K-fold cross-validation - algorithm/m

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

# Import scikit-Learn module classification report to later use for in
#try to classify/lable each record

from sklearn.metrics import classification_report

from sklearn.linear_model import LinearRegression
```

In [31]:
```
# Load the data set into a pandas dataframe
# Read the Iris data set and create the dataframe df

location = (r'/Users/laptopcheckout/Downloads/Iris.csv')
df = pd.read_csv (location)
df.head(5)
```

Out[31]:

	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

In [3]:
```
#print the information about the dataset
print(df.info())
```

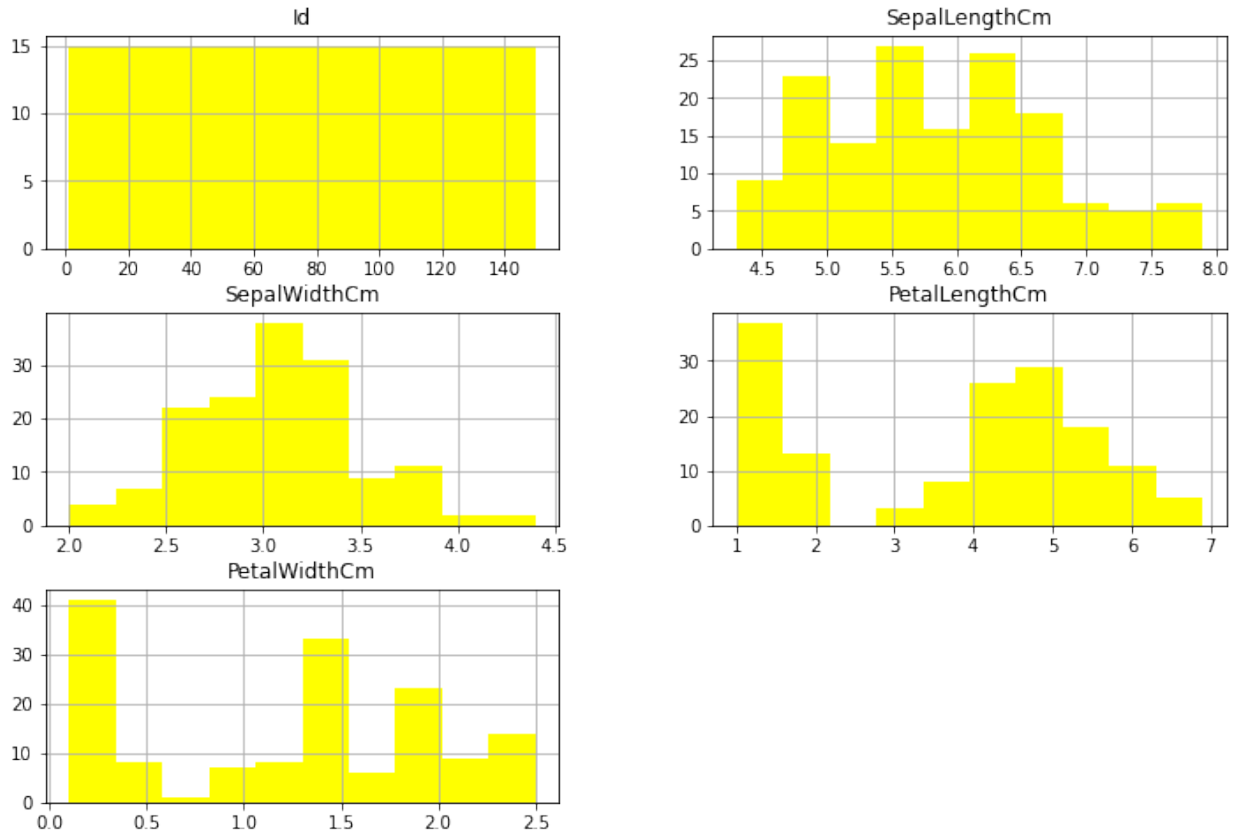
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 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
```

Univariate Data Visualization

Histograms

In [9]:
```
#create a histogram

df.hist(figsize=(12,8), color='yellow')
plt.show
```

Out[9]: <function matplotlib.pyplot.show(close=None, block=None)>

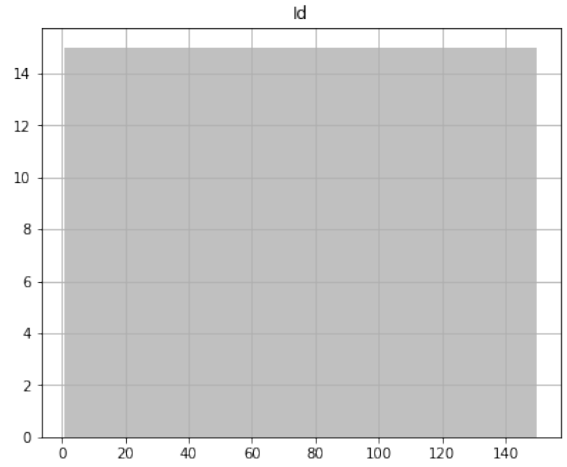


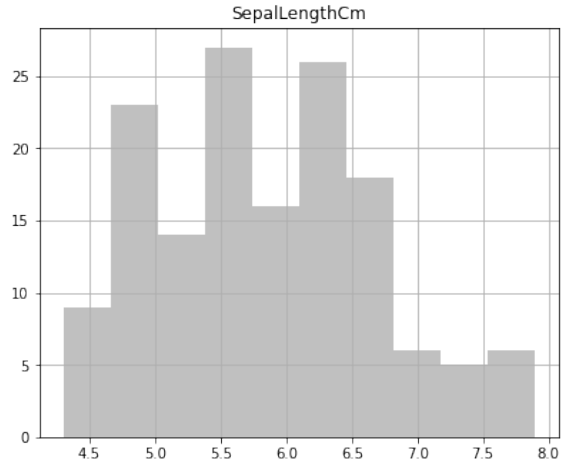
In [5]:
```
# here we want to see the different Species

print(df.groupby('Species').size())
```

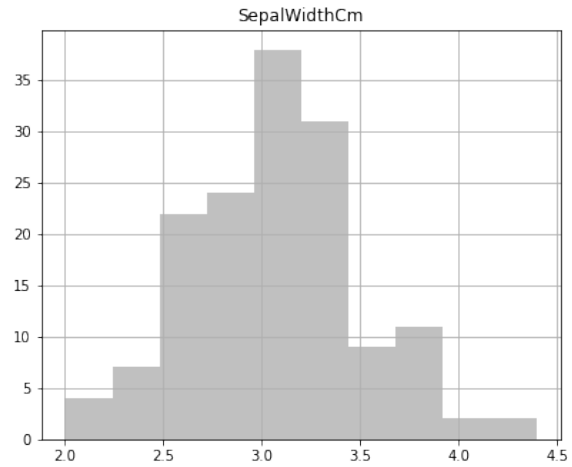
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

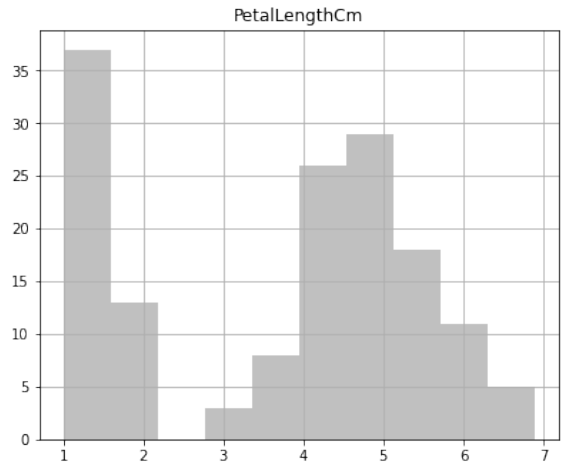
In [7]:
```
# Histograms are great when we would like to show the distribution of

df.hist(figsize=(15,19), color='silver')
plt.show
```

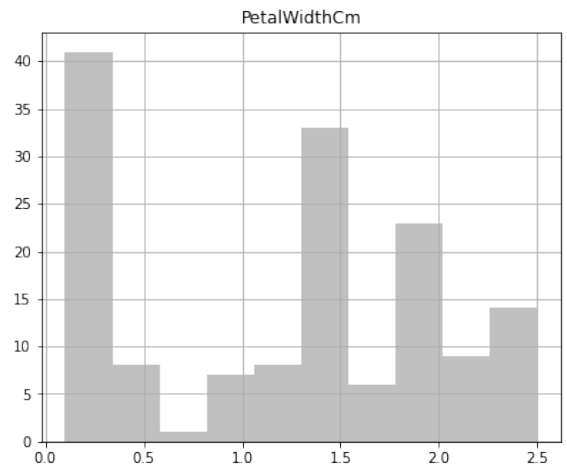
Out[7]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```







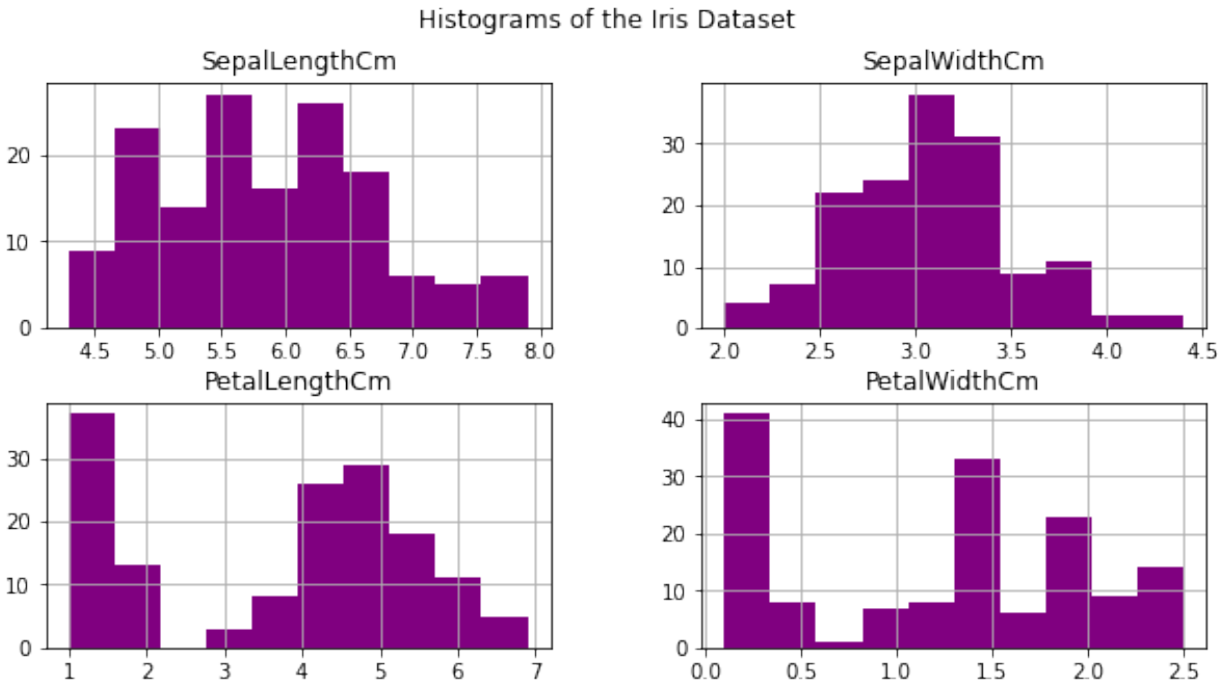




In [10]:
```
# in the above code, we see the variable "Id" is included in the analy

df.__delitem__('Id')
```

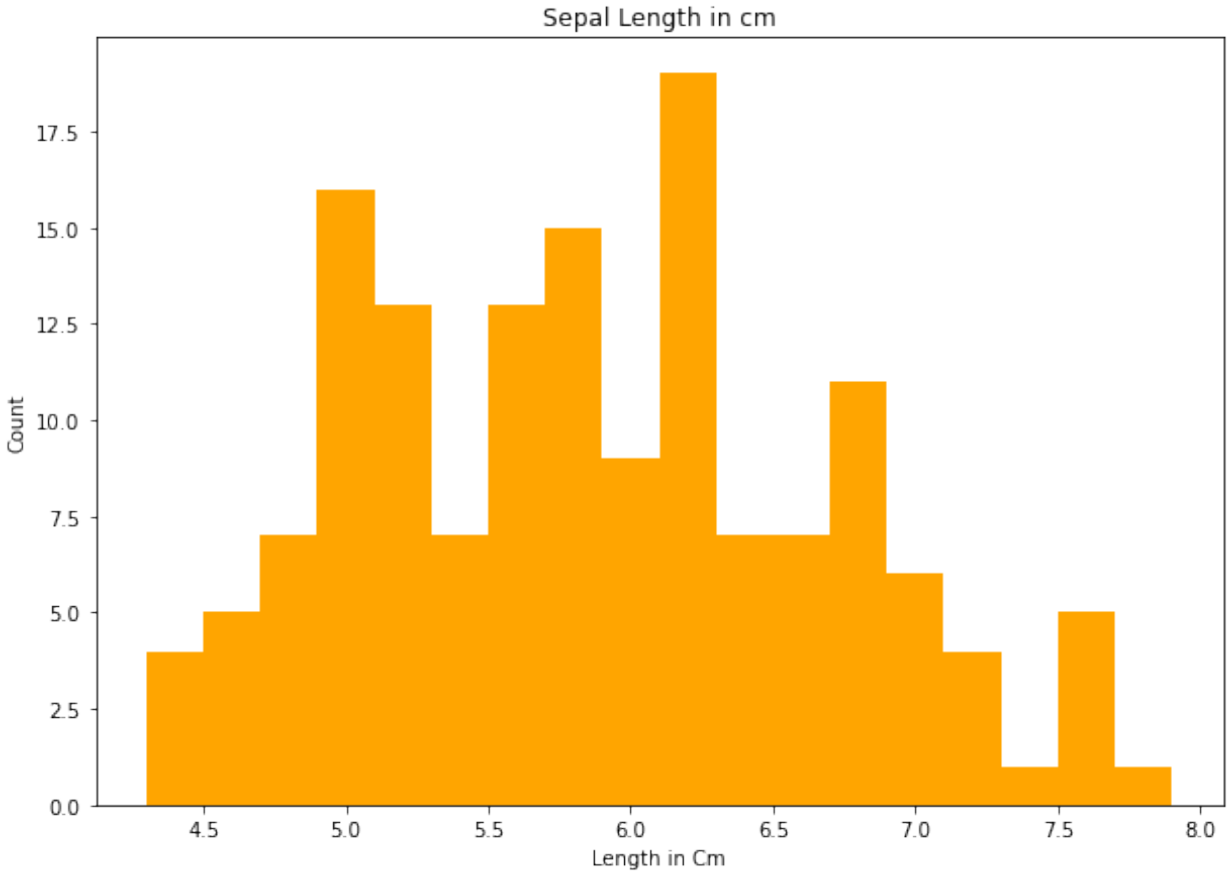
In [11]:
```
df.hist(figsize=(10,5), color ="purple")
plt.suptitle("Histograms of the Iris Dataset")
plt.show

# After this run, we see that "Id" is gone. you can also see we change
```

Out[11]: <function matplotlib.pyplot.show(close=None, block=None)>

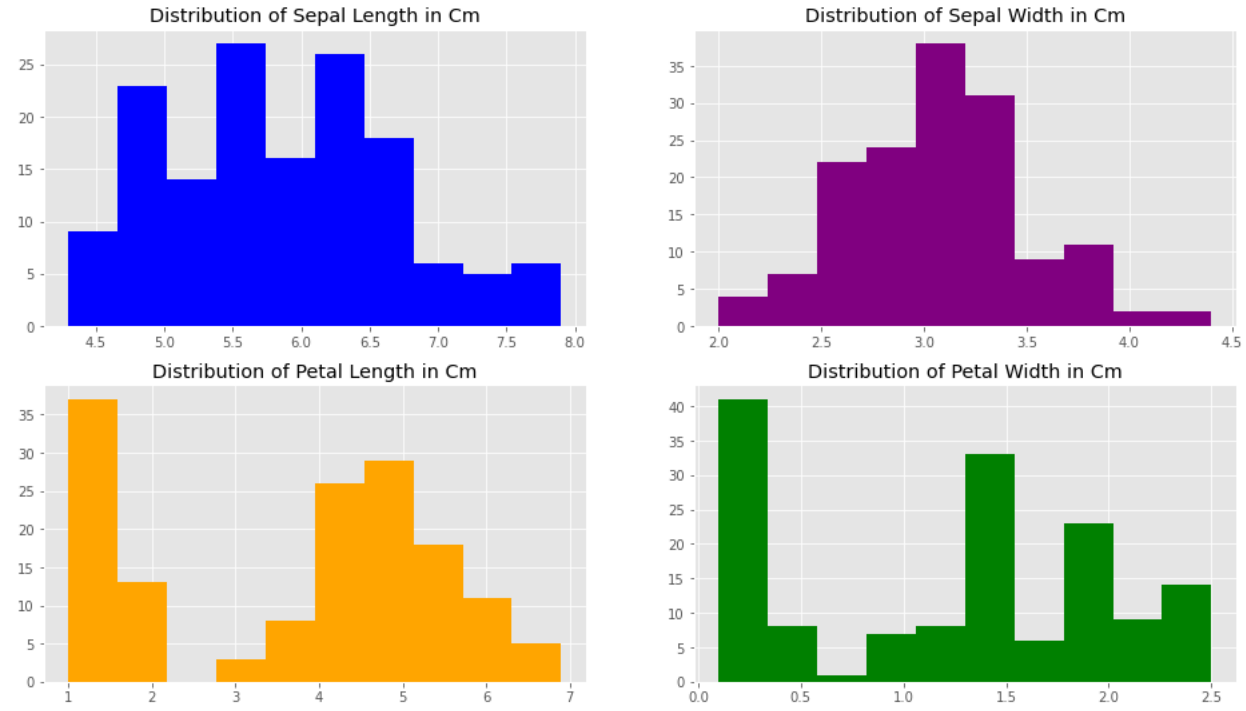


In [13]:
```
# histogram with just one variable - Sepal Length.  We need to isolate

plt.figure(figsize = (10, 7))
x = df["SepalLengthCm"]
plt.hist(x, bins = 18, color = "orange")
plt.title("Sepal Length in cm")
plt.xlabel("Length in Cm")
plt.ylabel("Count")
```

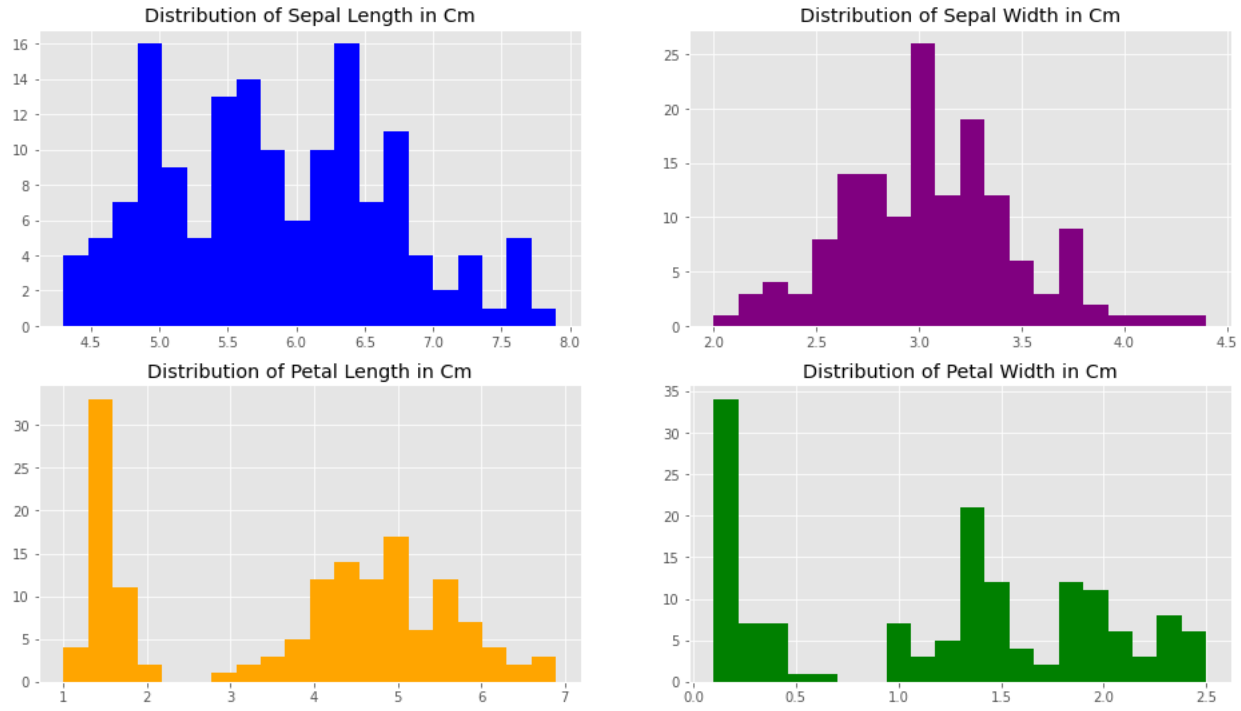
Out[13]: Text(0, 0.5, 'Count')



In [14]:
```
# here we are isolating the variable and giving each one a color and a

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');
```



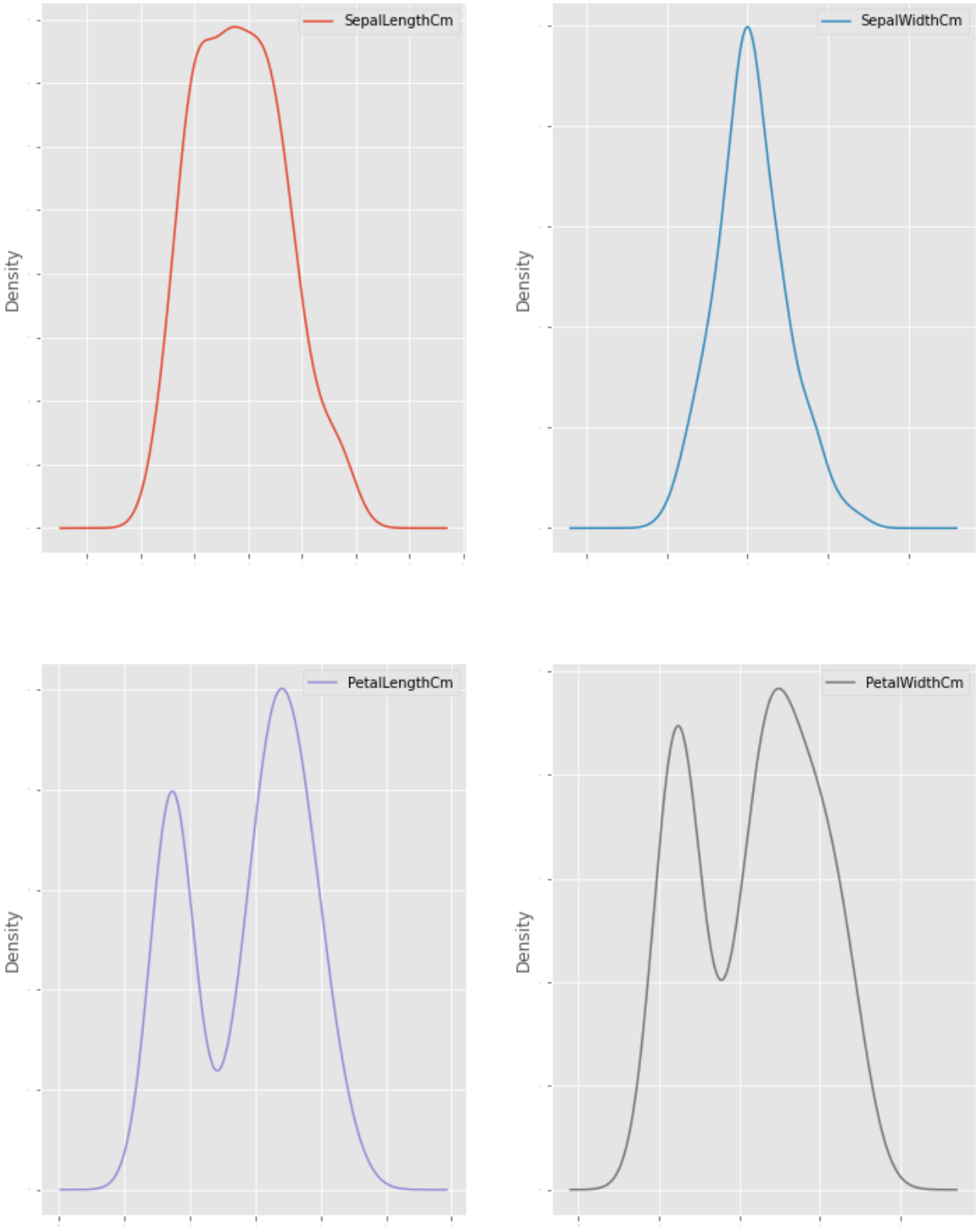
In [15]:
```
# Lastly, let's change the bin size

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');
```



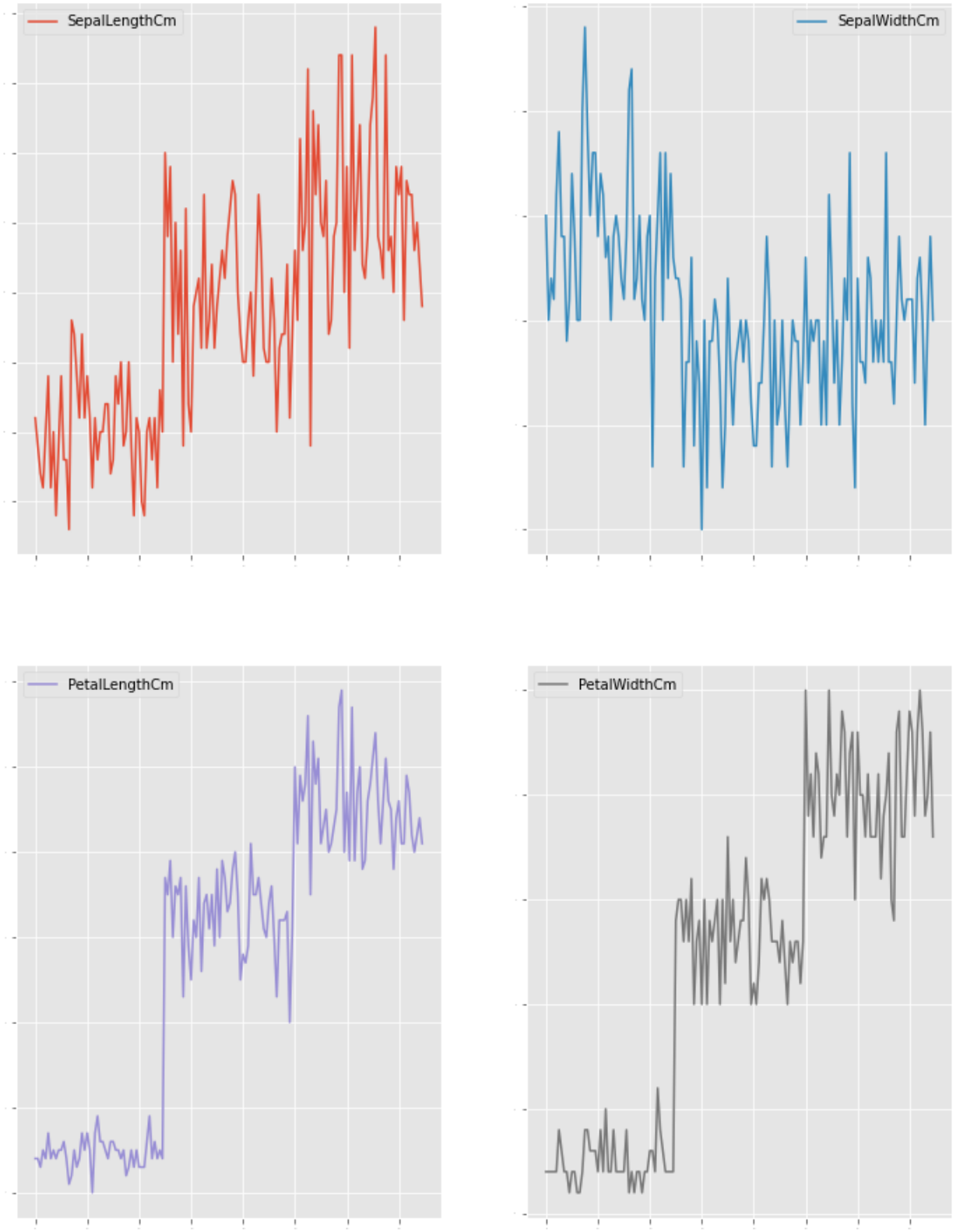
Density Plots

- A Density Plot visualizes the distribution of data over a time period or a continuous interval. This chart is a variation of a Histogram but it smooths out the noise made by binning.
- Density Plots have a slight advantage over Histograms since they're better at determining the distribution shape and, as mentioned above, they are not affected by the number of bins used (each bar used in a typical histogram).
- As we saw above a Histogram with only 10 bins wouldn't produce a distinguishable enough shape of distribution as a 20-bin Histogram would. With Density Plots, this isn't an issue.

In [16]:
```
# create the density plot

df. plot(kind='density', subplots=True, layout=(2,2), sharex=False, le
plt.show()
```

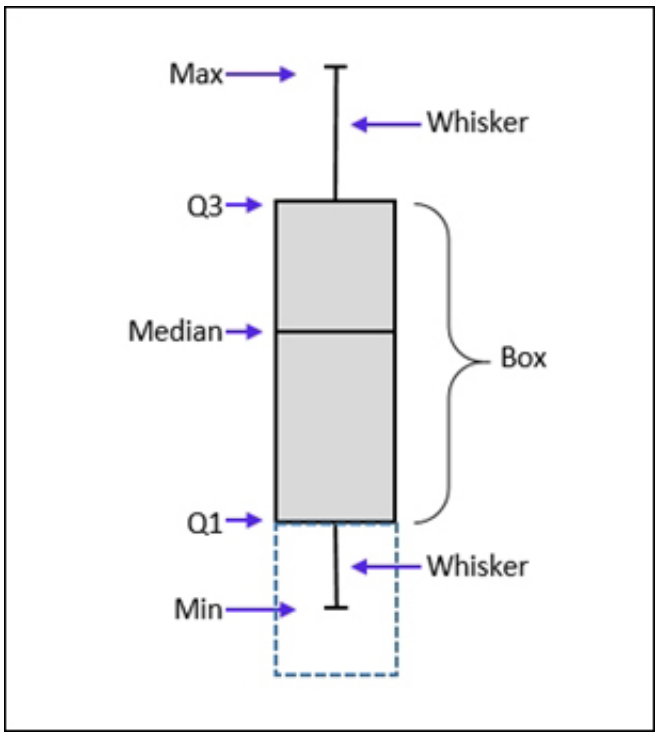


In [17]:
```
# create a line graph

df. plot(kind='line', subplots=True, layout=(2,2), sharex=False, leger
plt.show()
```



Boxplots

- A box plot is a very good plot to understand the spread, median, and outliers of data



1.Q3: This is the 75th percentile value of the data. It's also called the upper hinge.

2.Q1: This is the 25th percentile value of the data. It's also called the lower hinge.

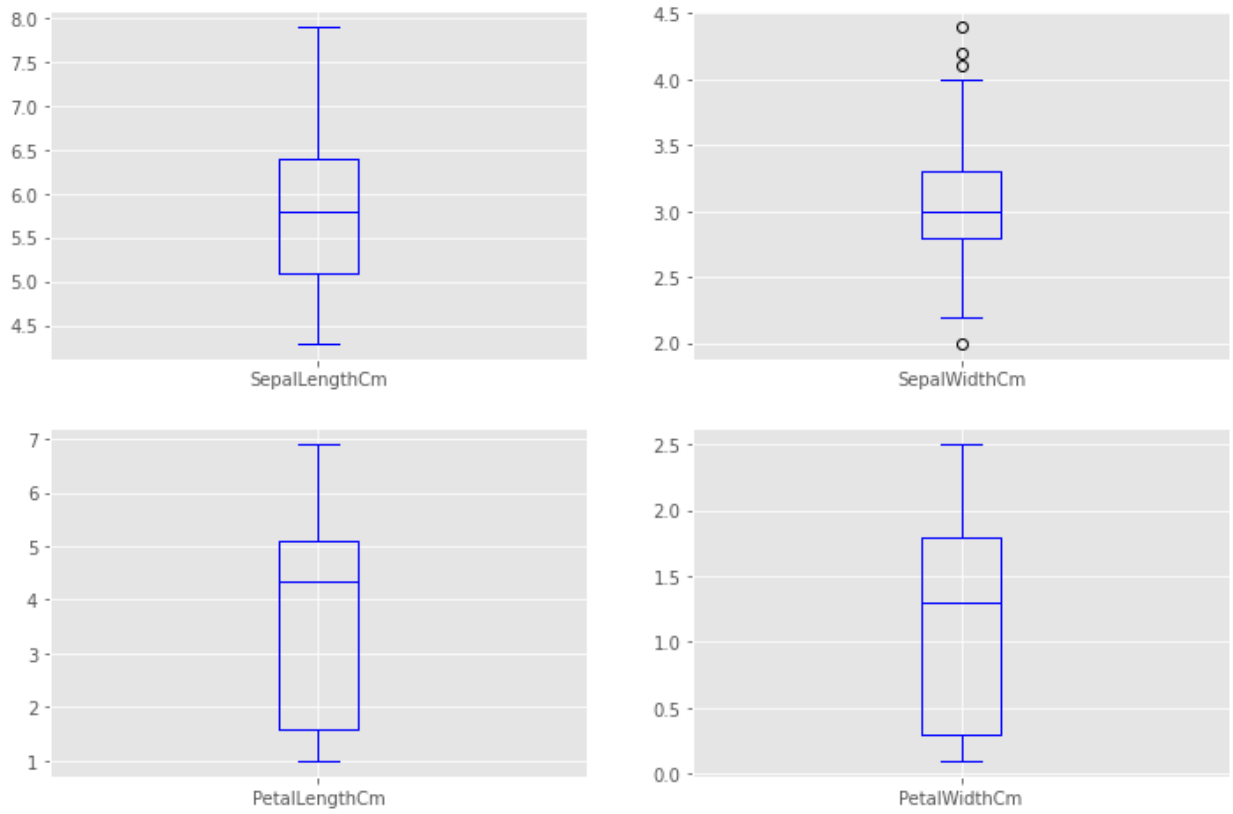
3.Box: This is also called a step. It's the difference between the upper hinge and the lower hinge.

4.Median: This is the midpoint of the data.

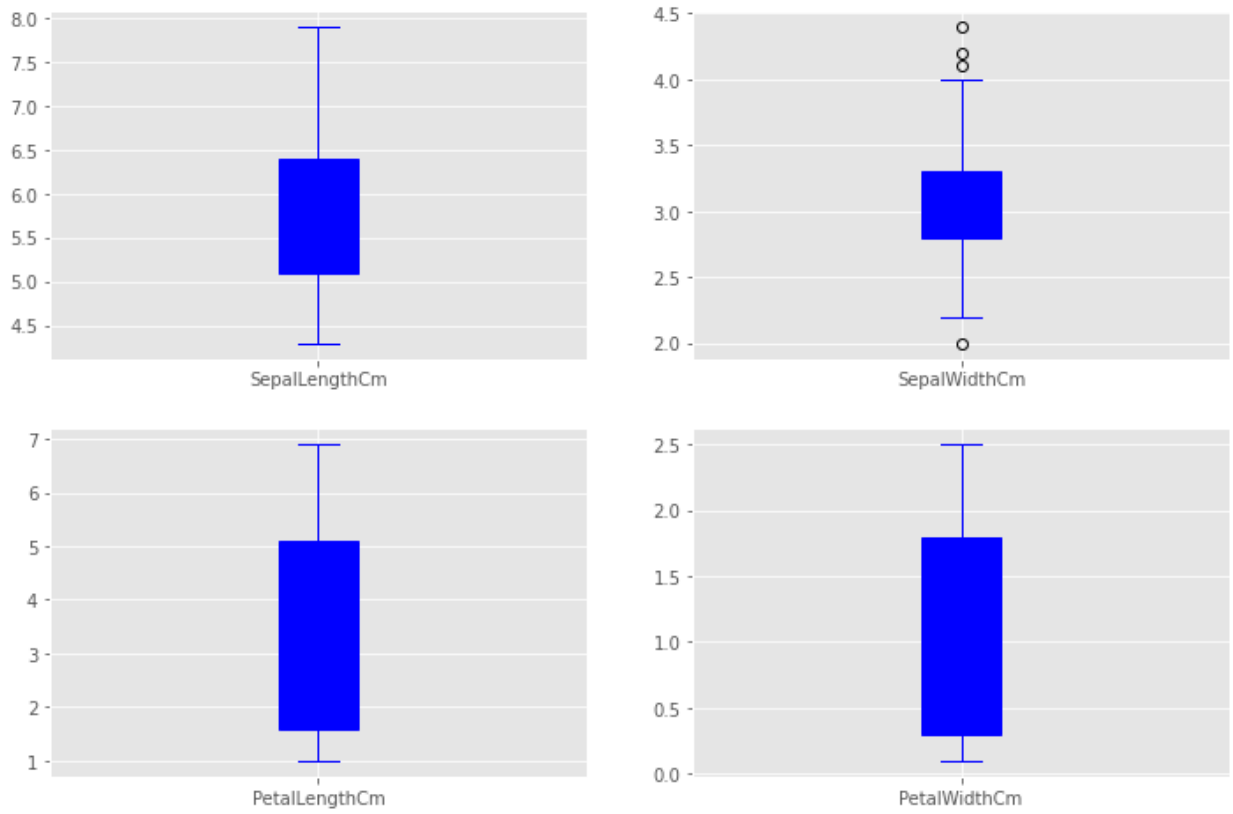
5.Max: This is the upper inner fence. It is 1.5 times the step above Q3.

6.Min: This is the lower inner fence. It is 1.5 times the step below Q1.

In [18]:
```
# create a box plot

df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=
plt.show()
```



In [19]:
```
# fill the boxes with color, using patch_artist

df.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=
plt.show()
```

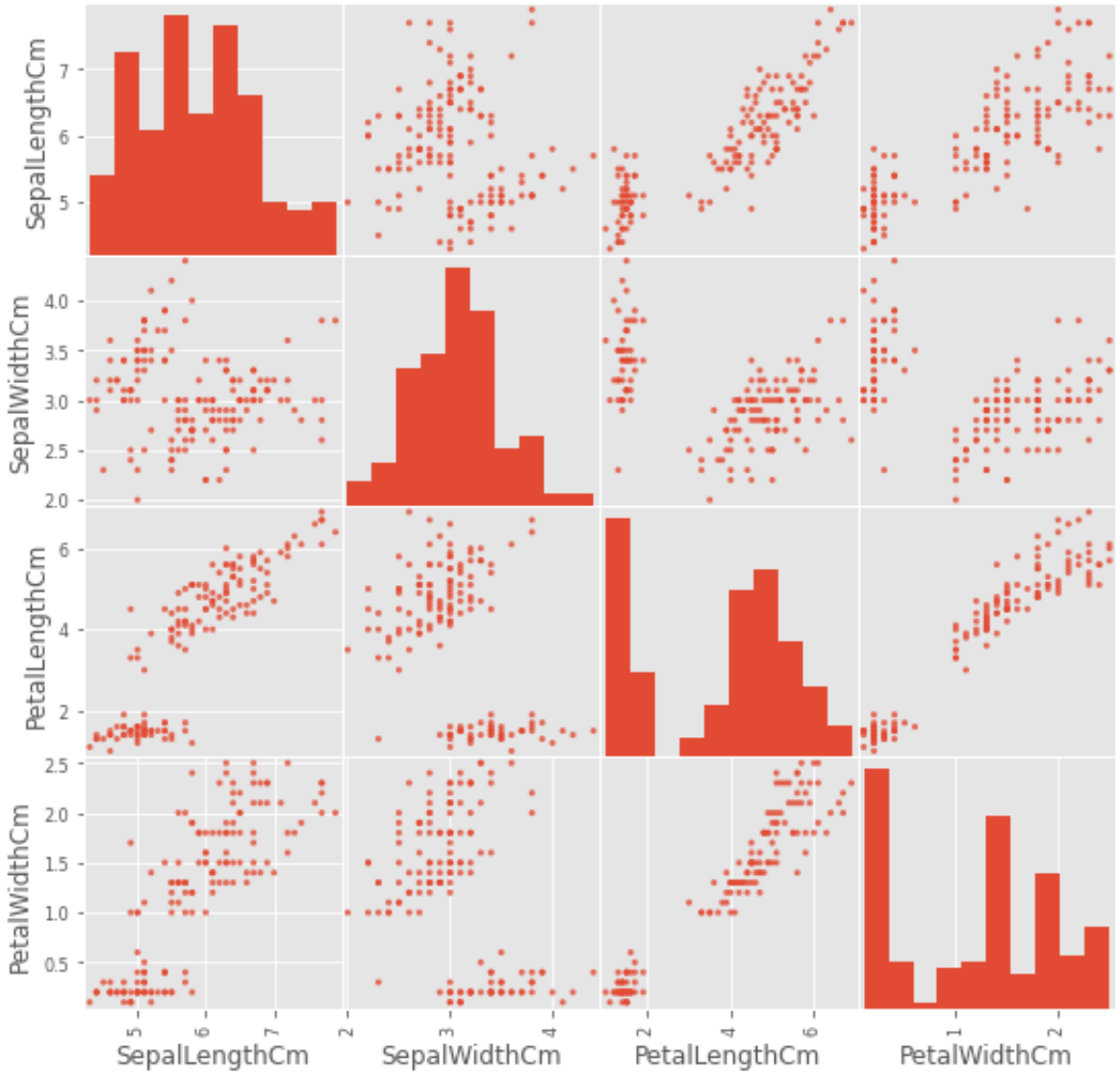


Multivariate Data Visualization

Scatter Matrix Plot

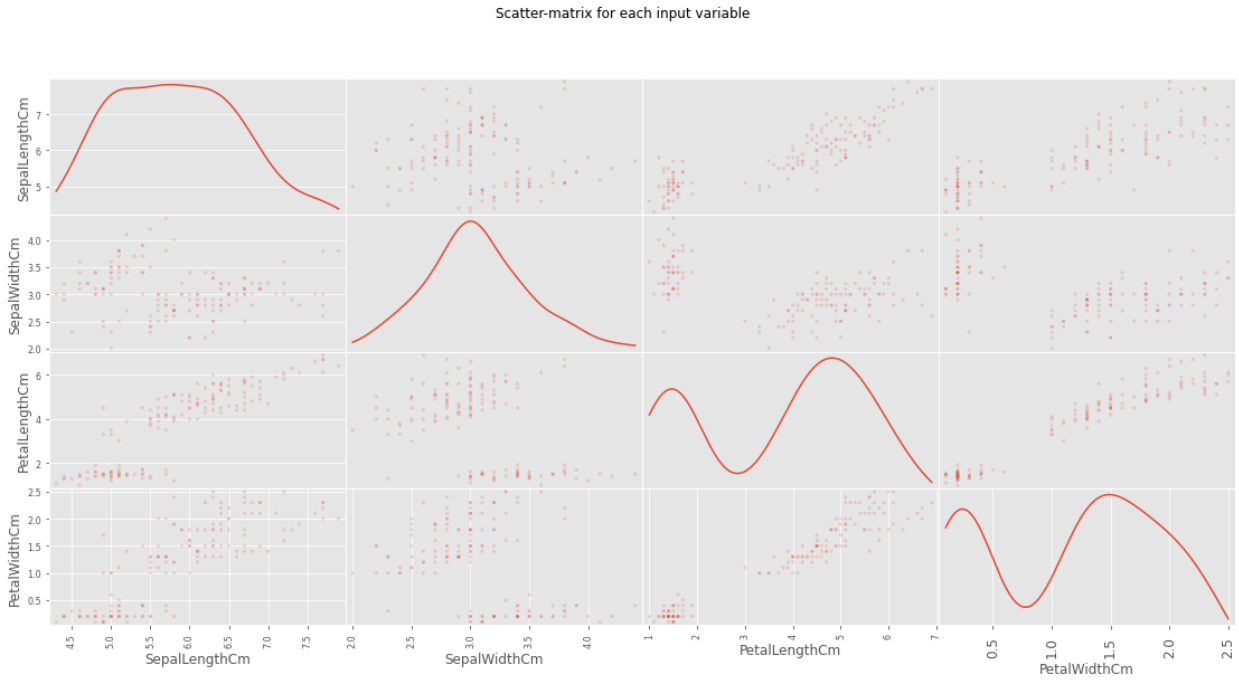
- A scatter plot matrix is a grid (or matrix) of scatter plots. This type of graph is used to visualize bivariate relationships between different combinations of variables. Each scatter plot in the matrix visualizes the relationship between a pair of variables, allowing many relationships to be explored in one chart. For example, the first row shows the relationship between SepalLength and the other 3 variables.

In [21]:
```
# create a scatter matrix plot

scatter_matrix (df, alpha=0.8, figsize=(9,9))
plt.show()
```



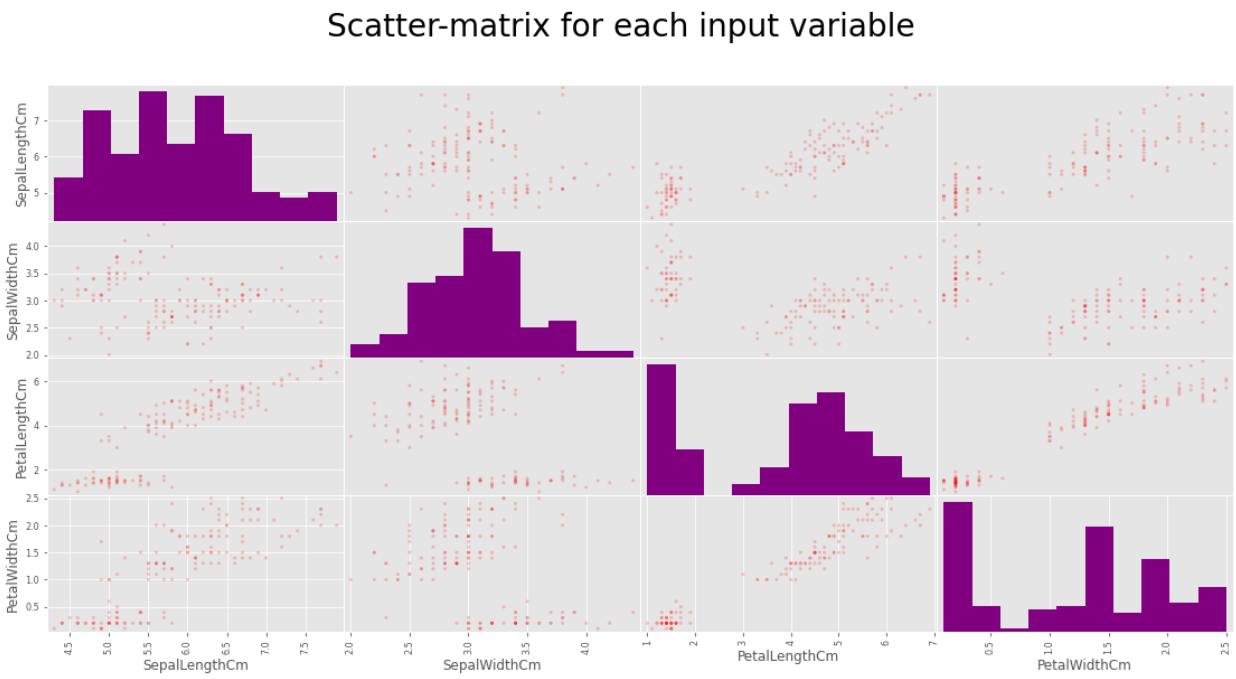
In [22]:
```
# change the color to purple.  Notice only one part of the plot change

scatter_matrix (df, alpha=0.8, figsize=(19,9), color = 'purple')
plt.show()
```

In [23]:
```
# Here you are adding a suptitle.

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)
```

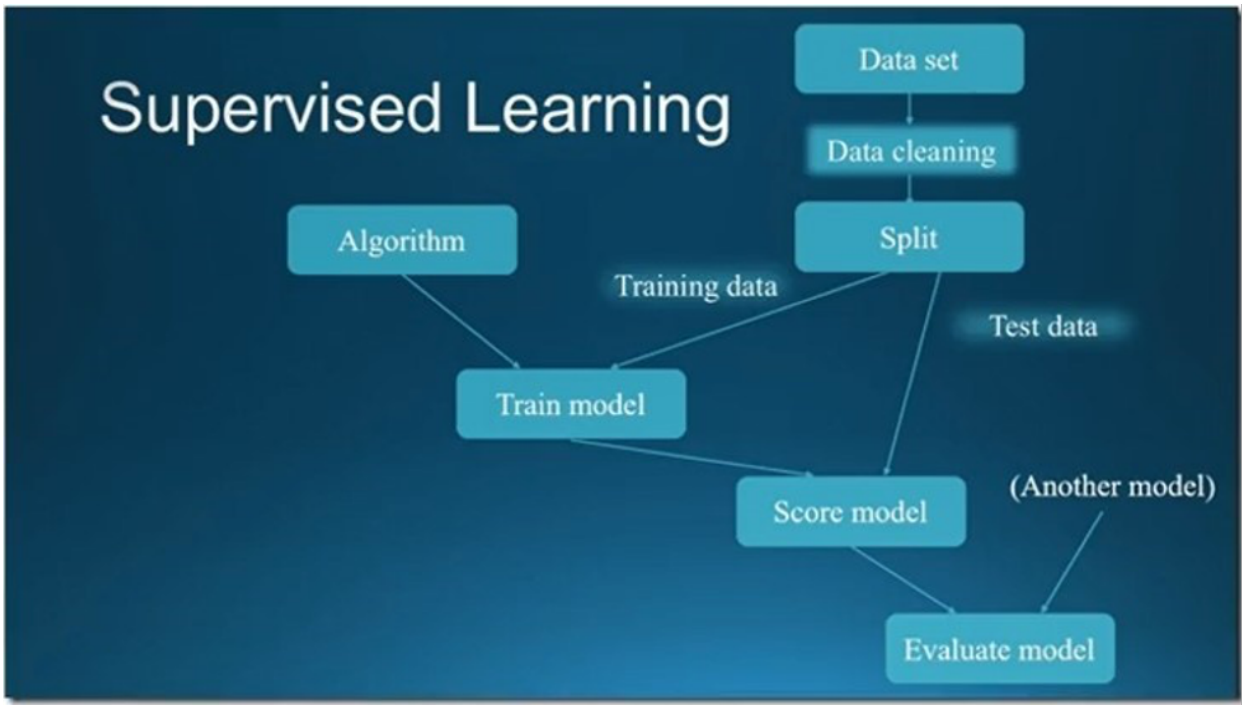


In [24]:
```
# Lastly, you are changing the color to both parts of the plot.

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



Machine Learning Supervised Logistic Regression



- You can see in this picture that supervised learning starts with the data set. Remember since it is supervised, the data is labeled. Then there is some data preprocessing (cleaning) to be done. Next, you will declare your input (X/Indepentdent variables) and output (Target Variable/Dependent or Y) NumPy Arrays. Then the data is split into a testing and training set. Then you will build and train the model, use the model for predictions, and lastly, evaluate/validate the model. So let's begin.

Description Iris Dataset

Data Set: Iris.csv Title: Iris Plants Database Updated Sept 21 by C. Blake -Added discrepancy information Sources:

- Creator: RA_ Fisher
- Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
- Date: 1988

Relevant Information: This is perhaps the best-known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example)

The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant.

Predicted attribute: class of Iris plant

Number of Instances: 150 (50 in each of three classes)

Number of predictors: 4 numeric

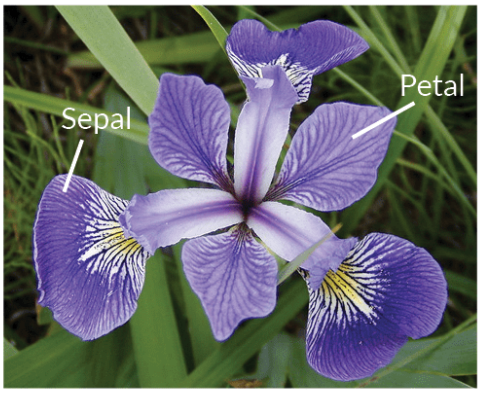
Predictive attributes and the class attribute information:

sepal length in cm

sepal width in cm

petal length in cm

petal width in cm



Iris Versicolor

Iris Setosa

Iris Virginica

In [32]:
```
location = (r'/Users/laptopcheckout/Downloads/Iris.csv')
df = pd.read_csv (location)
df.head(5)
```

Out[32]:

	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

Preprocess the Dataset

Clean the data: Find and Mark Missing Values

In [33]:
```
# mark zero values as missing or NaN

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

# count the number of NaN values in each column

print (df.isnull().sum())
```

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

Performing the Exploratory Data Analysis (EDA)

In [34]:
```
# get the dimensions or shape of the dataset
# i.e. number of records / rows X number of variables / columns

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

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

In [35]:
```
#get the data types of all the variables / attributes in the data set

print(df.dtypes)
```

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

In [36]:
```
#return the summary statistics of the numeric variables/attributes in

print(df.describe())
```

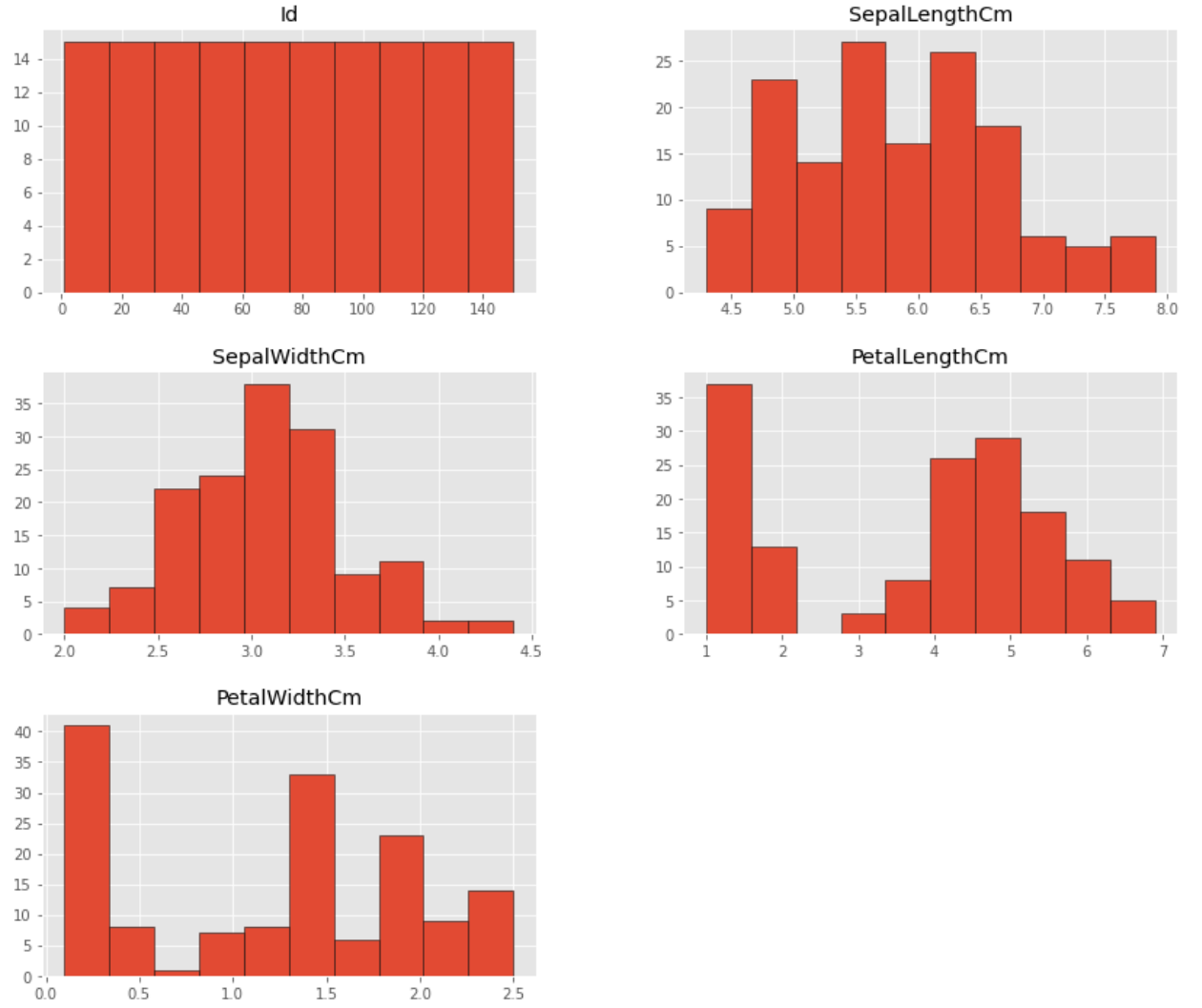
```
                Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalW
idthCm
count  150.000000     150.000000    150.000000     150.000000    150.
000000
mean    75.500000       5.843333      3.054000       3.758667      1.
198667
std     43.445368       0.828066      0.433594       1.764420      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 [37]:
```
#class distribution i.e. how many records are in each class

print(df.groupby('Species').size())
```

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

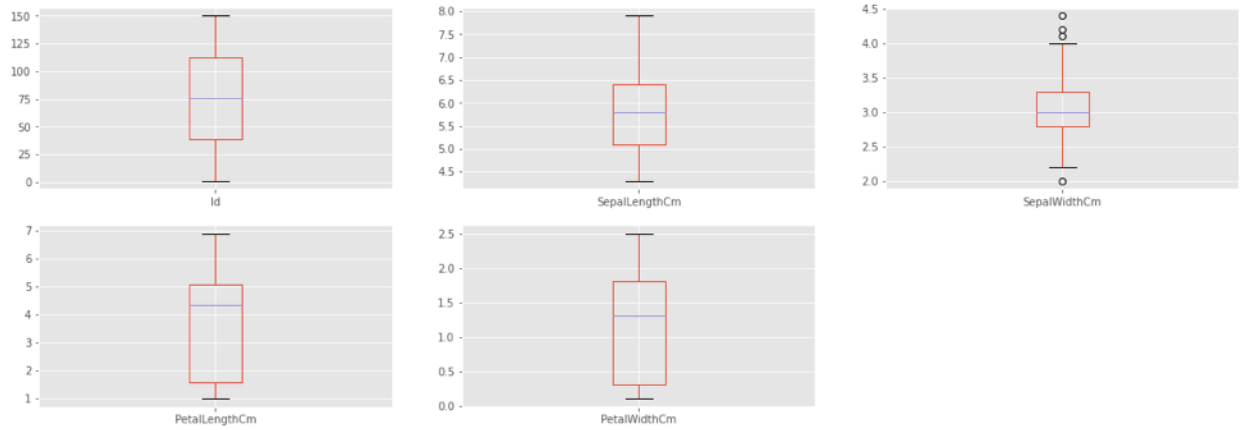
Creating a Histogram

In [38]:
```
# Plot histogram for each variable. I encourage you to work with the h

df.hist(edgecolor= 'black',figsize=(14,12))
plt.show()
```



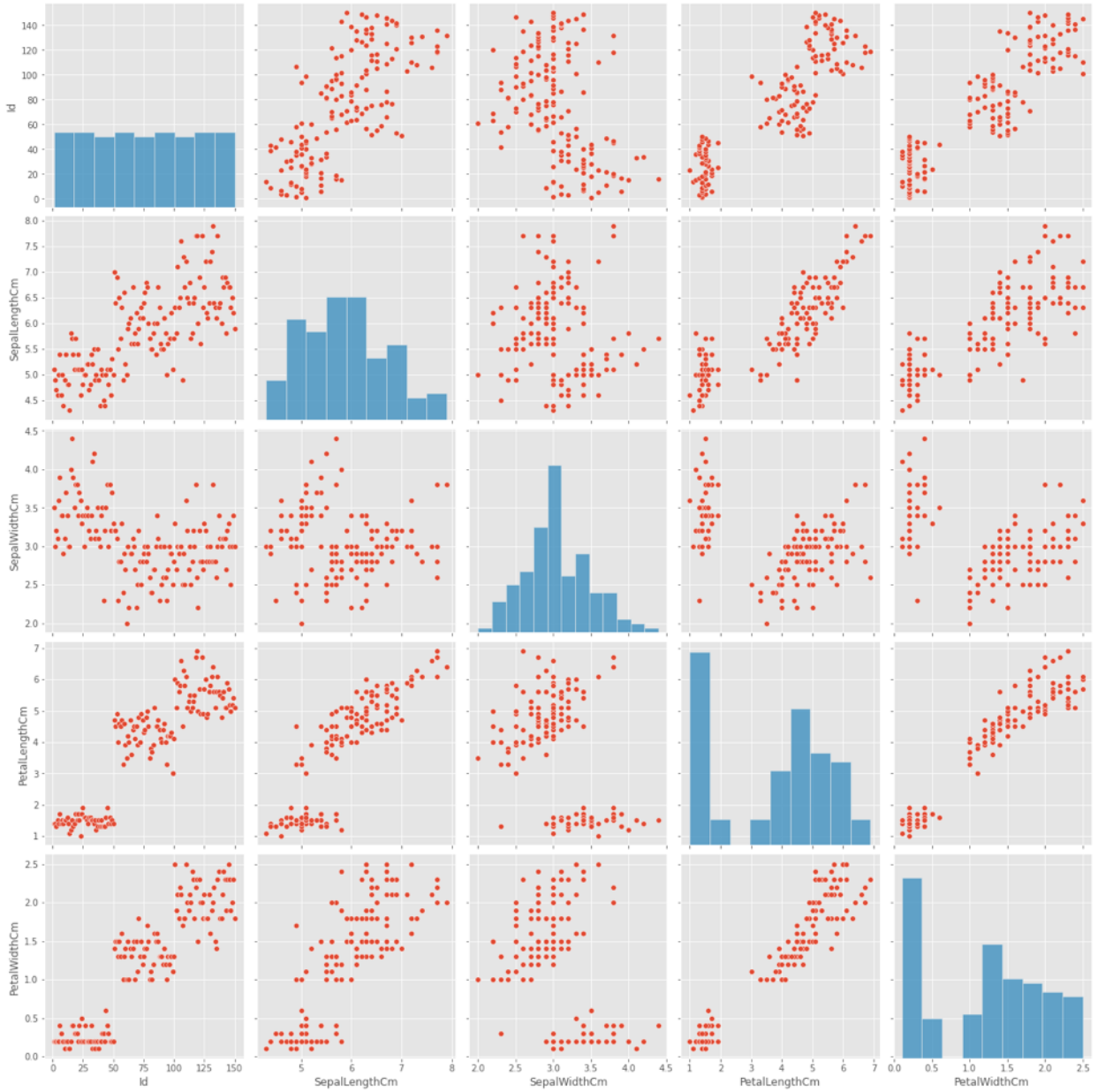
Creating a Box Plot

In [39]:
```
# Boxplots

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



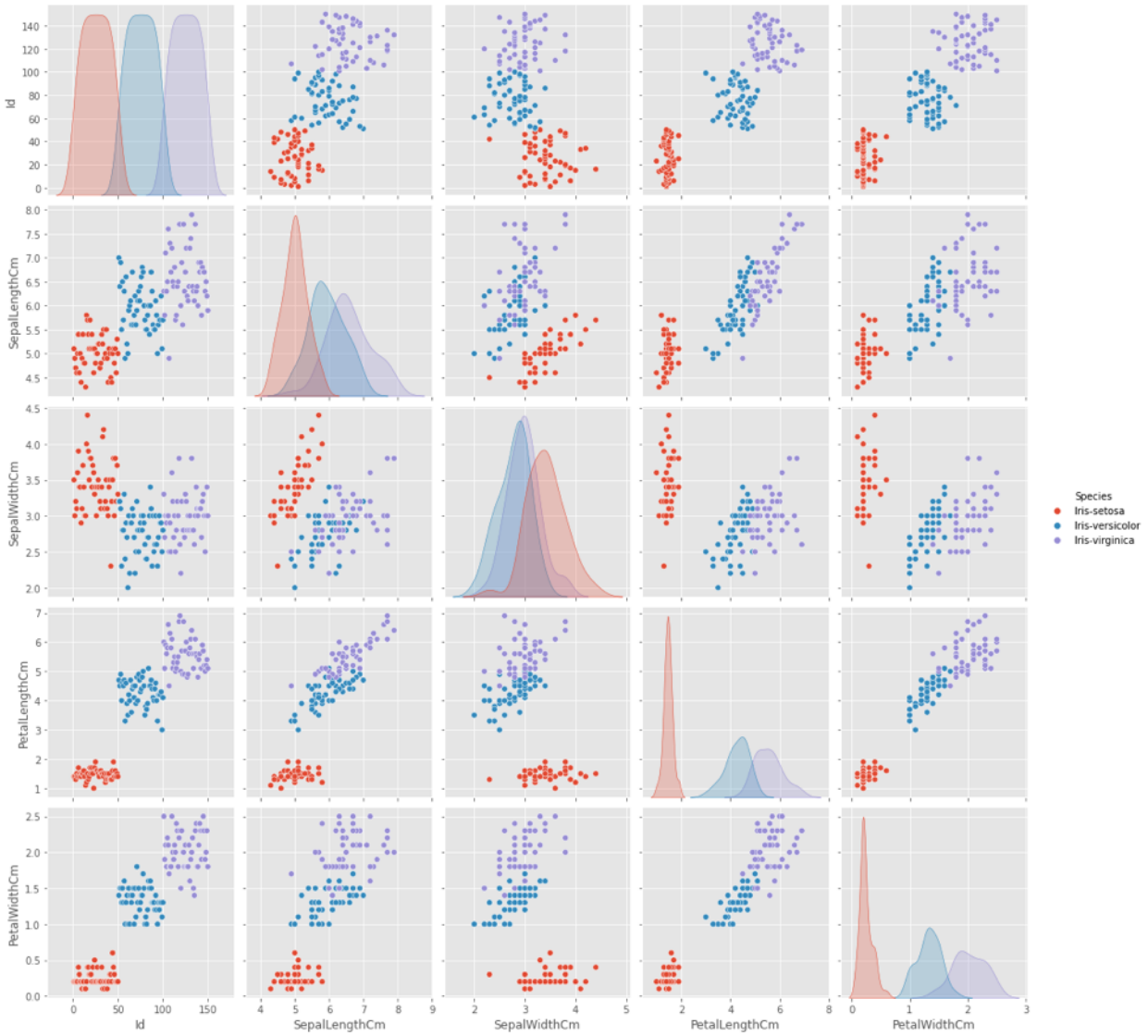
Create a Pair Plot

In [40]:
```
# 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

sns.pairplot(df, height=3.5);
plt.show()
```



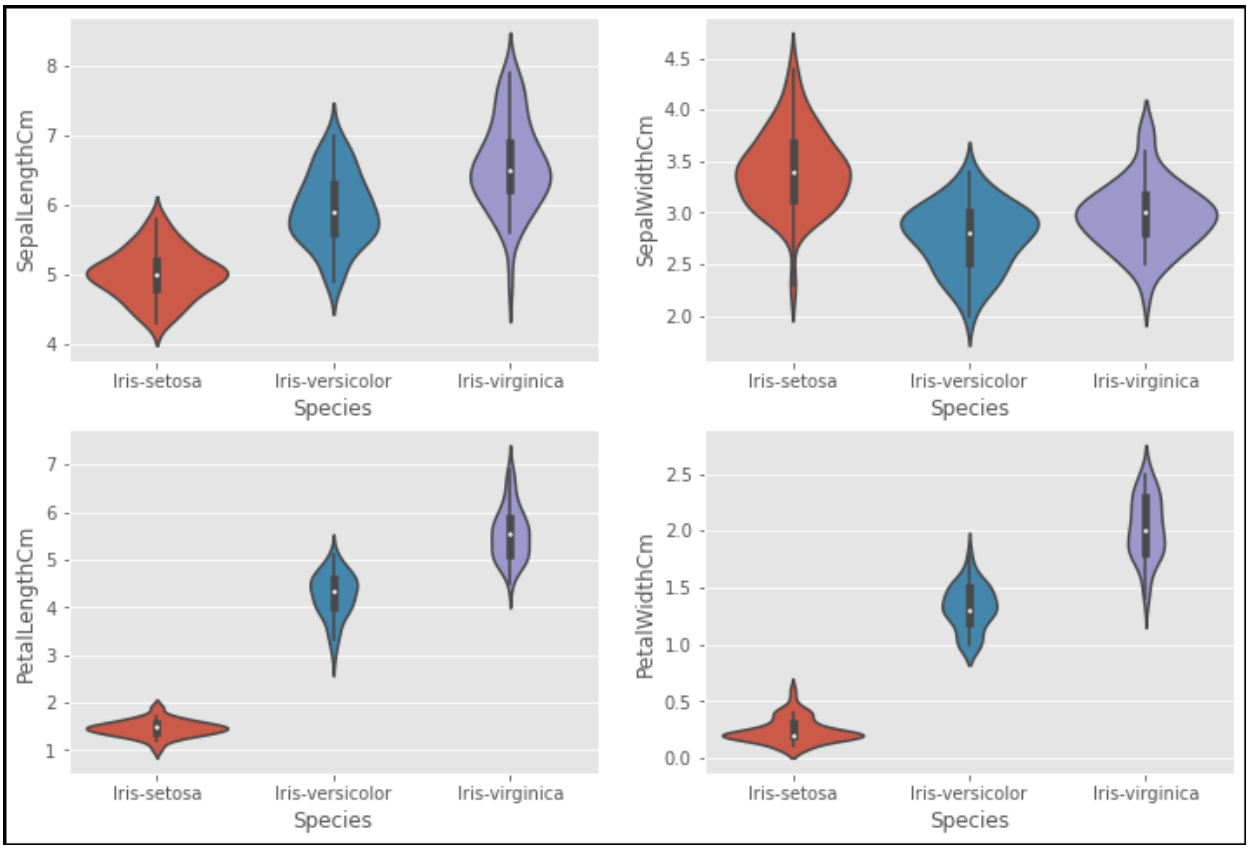
Creating a Pair Plot with Color

In [41]:
```
# Let's try that again using color.  Notice: assigning a hue variable

sns.pairplot(df, hue='Species', height=3, aspect= 1);
```



Creating a Violin Plot

In [42]:
```
#  Please click on the URL above to learn more about Violin Plots

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);
```



Separate the Dataset into Input & Output NumPy Arrays

In [43]:
```
# 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]
```

Spilt into Input/Output Array into Training/Testing Datasets

In [44]:
```
# split the dataset --> training sub-dataset: 67%; test sub-dataset: 3

test_size = 0.33

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

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=te
random_state=seed)
```

Build and Train the Model

In [45]:
```
#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
```

Score the Accuracy of the Model

In [46]:
```
#score the accuracy level

result = model.score(X_test, Y_test)

#print out the results

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

```
Accuracy: 92.000%
```

Classify/Prediction

In [47]:
```
model.predict([[5.3, 3.0, 4.5, 1.5]])
```

Out[47]:
```
array(['Iris-versicolor'], dtype=object)
```

In [48]:
```
model.predict([[5, 3.6, 1.4, 1.5]])
```

Out[48]:
```
array(['Iris-setosa'], dtype=object)
```

Evaluate the Model using the 10-fold Cross-Validation Technique.

In [49]:
```
# Evaluate the algorithm and specify the number of times of repeated s

n_splits=10

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

seed=7

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

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

scoring="accuracy"

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

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

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

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

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

Machine Learning Supervised Linear Regression

Description of Boston Housing Dataset

- CRIM: This is the per capita crime rate by town
- ZN: This is the proportion of residential land zoned for lots larger than 25,000 sq. ft.
- INDUS: This is the proportion of non-retail business acres per town.
- CHAS: This is the Charles River dummy variable (this is equal to 1 if tract bounds river; 0 otherwise)
- NOX: This is the concentration of the nitric oxide (parts per 10 million)
- RM: This is the average number of rooms per dwelling
- AGE: This is the proportion of owner-occupied units built prior to 1940
- DIS: This is the weighted distances to five Boston employment centers
- RAD: This is the index of accessibility to radial highways
- TAX: This is the full-value property-tax rate per 10,000 dollars
- PTRATIO: This is the pupil-teacher ratio by town
- AA: This is calculated as $1000(AA - 0.63)^2$, where AA is the proportion of people of African American descent by town
- LSTAT: This is the percentage lower status of the population
- MEDV: This is the median value of owner-occupied homes in $1000s

Load Data

In [53]:
```
# Specify location of the dataset.  Be sure to  NOT  use the housing_b

location = '/Users/laptopcheckout/Downloads/housing boston.csv'
```

In [55]:
```
# Load the data into a Pandas DataFrame

df= pd.read_csv (location, header=None)
```

In [56]:
```
df.head()
```

Out[56]:

	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 since there are no headers

In [57]:
```
#give names to the columns

col_names = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX','RM', 'AGE', 'DIS',
```

In [58]:
```
# Let's check to see if the column names were added

df.columns = col_names
```

Look at the dataframe

In [59]:
```
#  Look at the first 5 rows of data

df.head()
```

Out[59]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	AA	LST
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.

Preprocess the Dataset

Clean the data: Find and Mark Missing Values

In [60]:
```
df.isnull().sum()

# We see there are no missing data points
```

Out[60]:
```
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
```

Performing the Exploratory Data Analysis (EDA)

In [61]:
```
# Get the number of records/rows, and the number of variables/columns

print(df.shape)
```

```
(452, 14)
```

In [62]:
```
# Get the data types of all variables

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
```

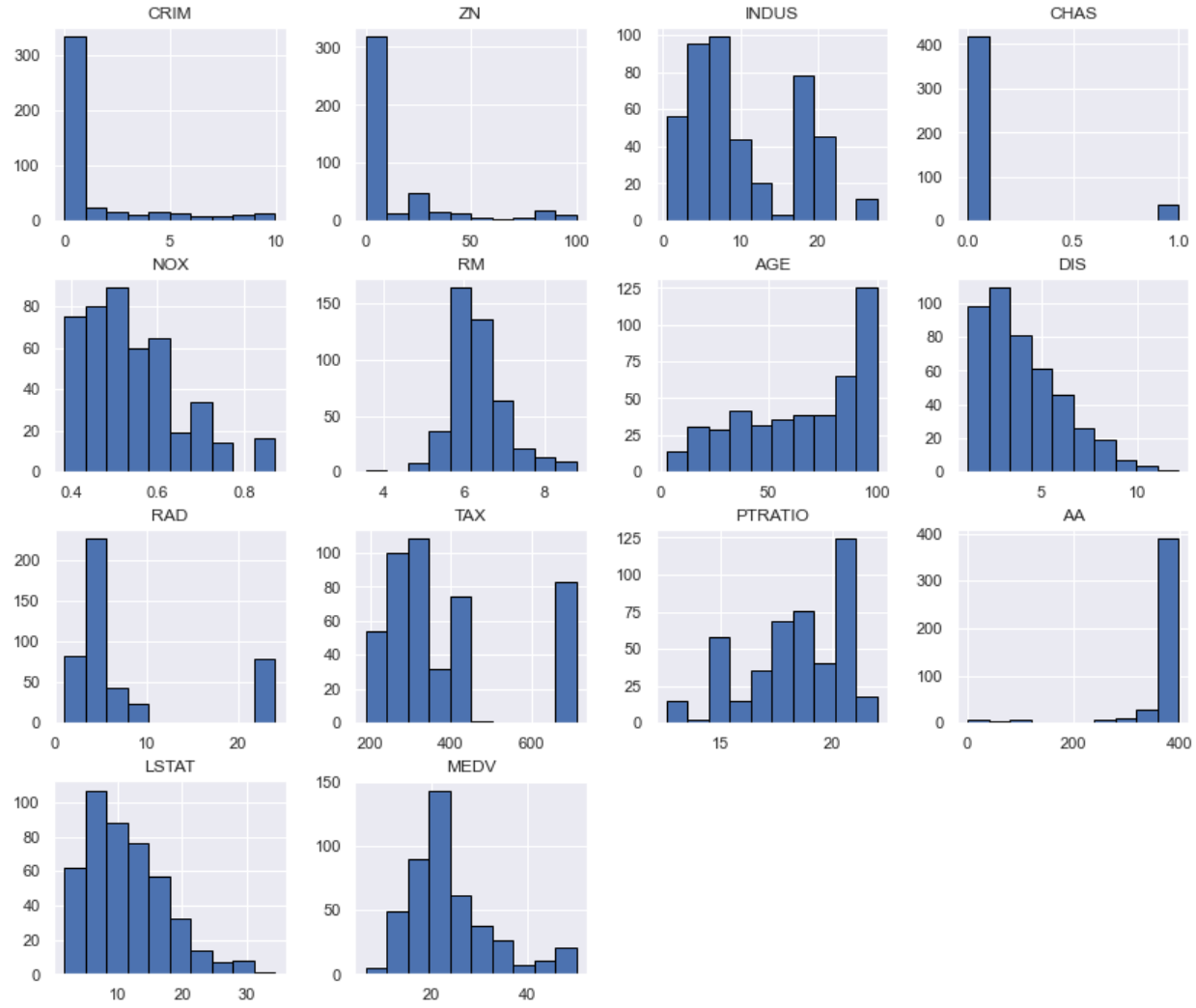
In [63]:
```
# Obtain the summary statistics of the data

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
AA  \
count  452.000000  452.000000  452.000000  452.000000  452.000000  45
2.000000
mean    65.557965    4.043570    7.823009  377.442478   18.247124  36
9.826504
```

```
std     28.127025    2.090492    7.543494  151.327573    2.200064   6
8.554439
min      2.900000    1.129600    1.000000  187.000000   12.600000
0.320000
25%     40.950000    2.354750    4.000000  276.750000   16.800000  37
7.717500
50%     71.800000    3.550400    5.000000  307.000000   18.600000  39
2.080000
75%     91.625000    5.401100    7.000000  411.000000   20.200000  39
6.157500
max    100.000000   12.126500   24.000000  711.000000   22.000000  39
6.900000

            LSTAT        MEDV
count  452.000000  452.000000
mean    11.441881   23.750442
std      6.156437    8.808602
min      1.730000    6.300000
25%      6.587500   18.500000
50%     10.250000   21.950000
75%     15.105000   26.600000
max     34.410000   50.000000
```

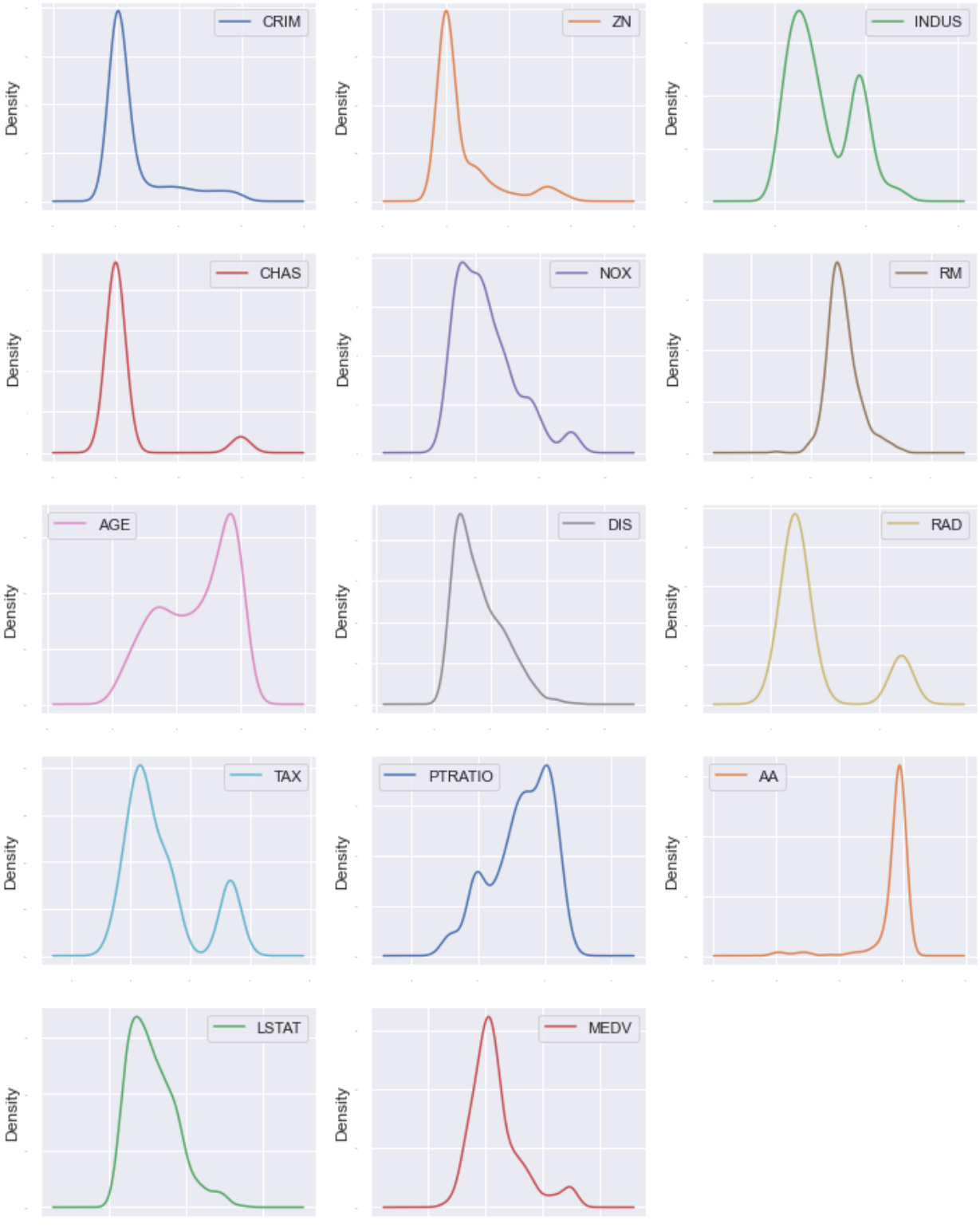
Creating a Histogram

In [64]:
```
# Plot histogram for each variable. I encourage you to work with the h

df.hist(edgecolor= 'black',figsize=(14,12))
plt.show()
```



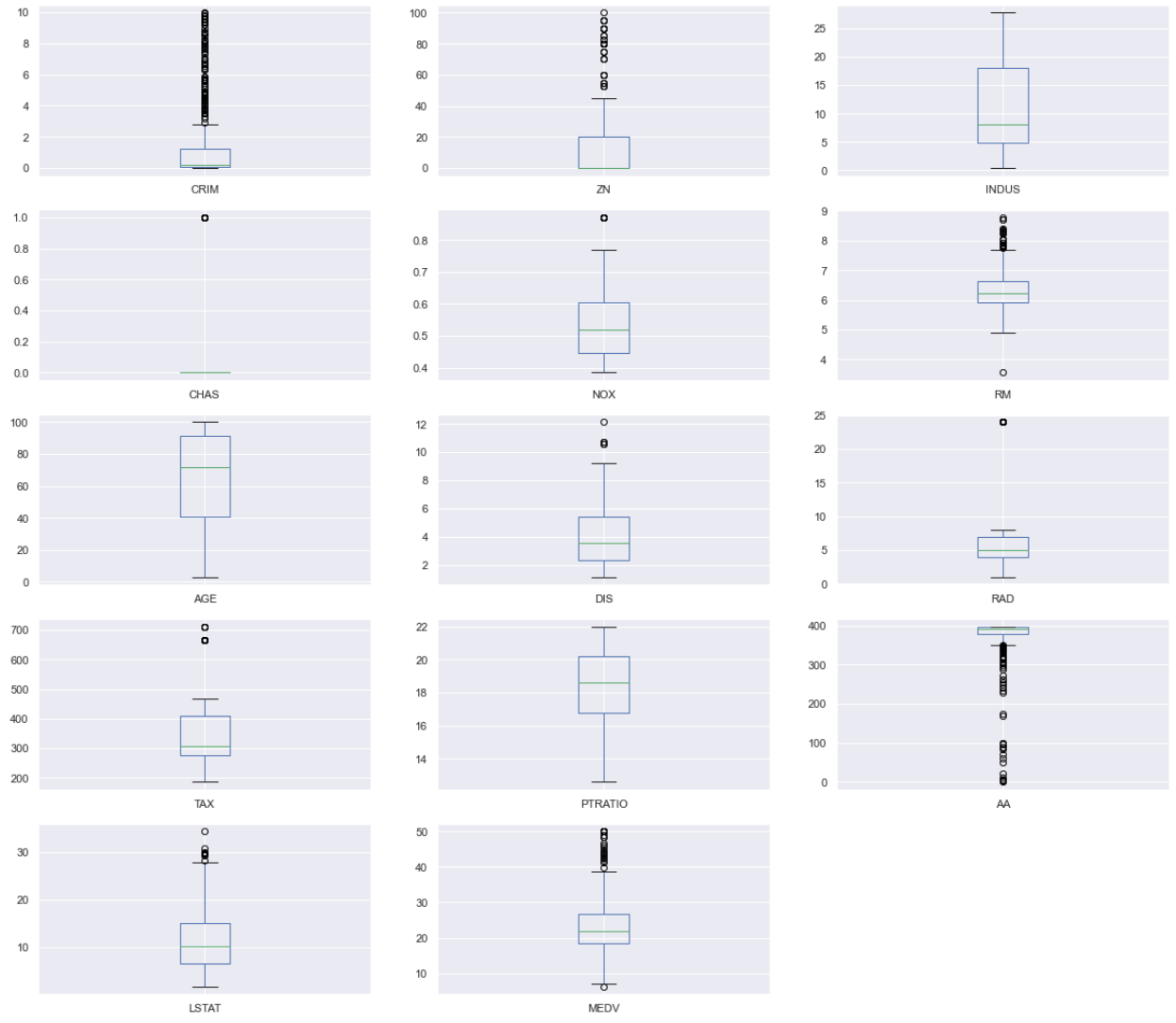
Creating a Density Plot

In [65]:
```
# Density plots
# Notes:  14 numeric variable, at least 14 plots, layout (5,3): 5 rows

df.plot(kind='density', subplots=True, layout= (5,3), sharex=False, le
plt.show()
```



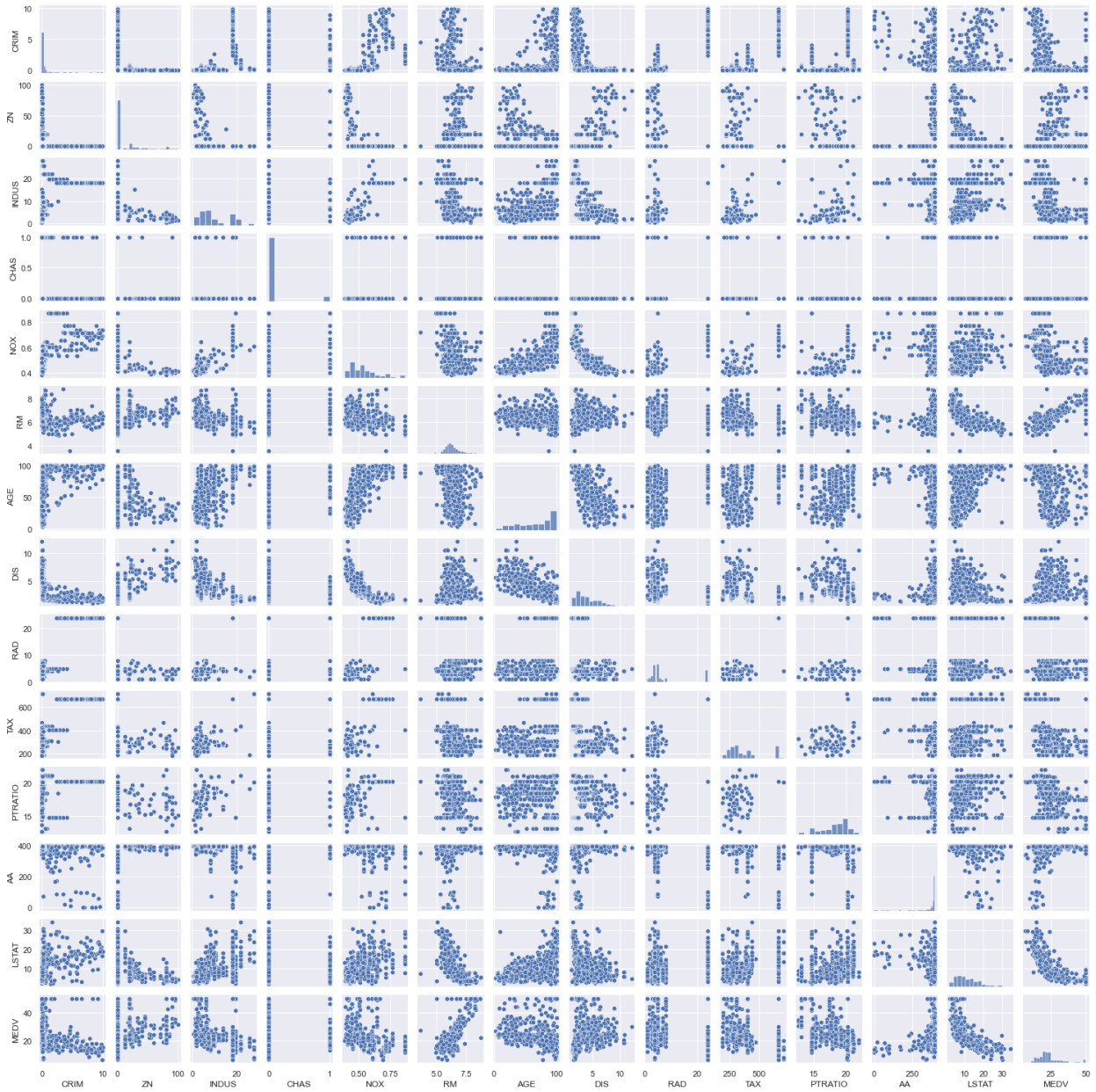
Creating a Box Plot

In [66]:
```
# Boxplots

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



Correlation Analysis and Feature Selection

In [67]:
```
#Obtain pair plots of the data.  I know this is a lot of information b

sns.pairplot(df, height=1.5);
plt.show()
```



Correlations

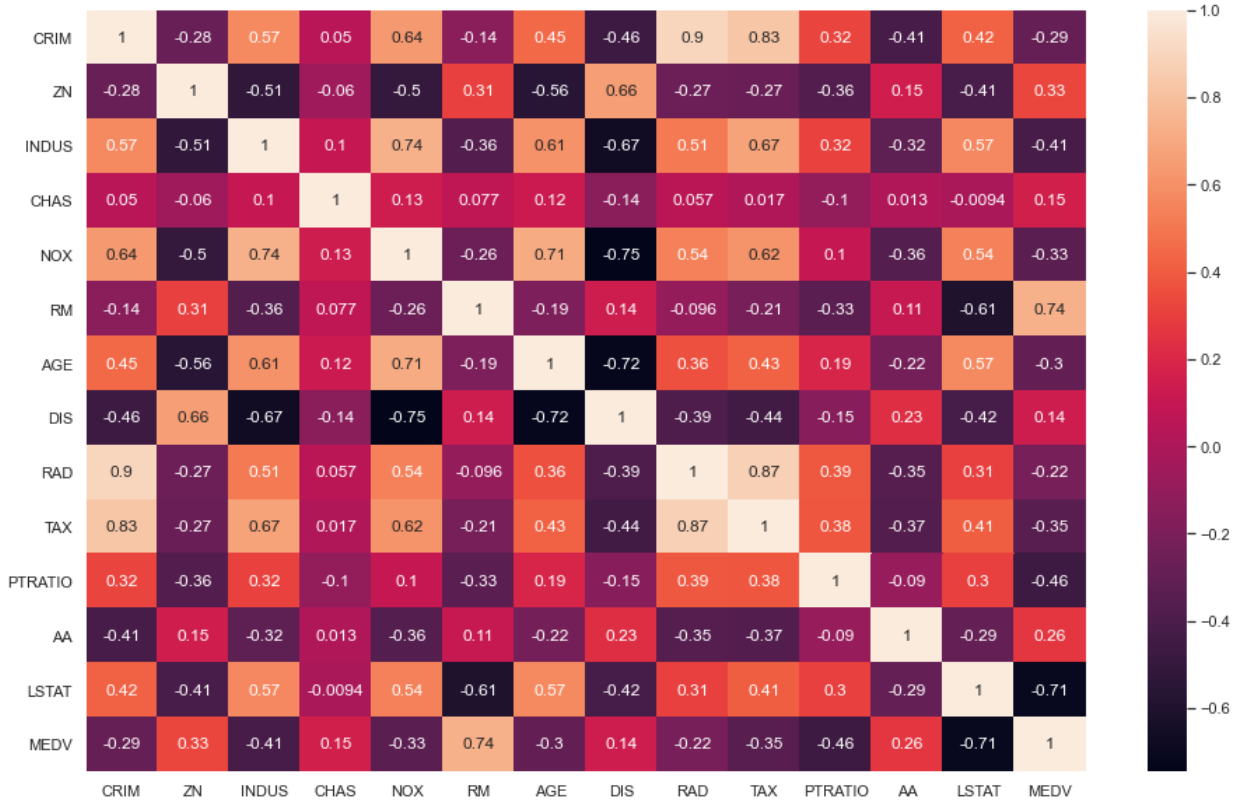
In [68]:
```
# We will decrease the number of decimal places with the format functi

pd.options.display.float_format = '{:,.3f}'.format
```

In [69]:
```
# Here we will get the correlations, with only 3 decimals.

df.corr()
```

Out[69]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO
CRIM	1.000	-0.281	0.574	0.050	0.637	-0.142	0.448	-0.462	0.898	0.826	0.319
ZN	-0.281	1.000	-0.514	-0.060	-0.501	0.307	-0.556	0.656	-0.267	-0.269	-0.364
INDUS	0.574	-0.514	1.000	0.103	0.739	-0.365	0.606	-0.669	0.513	0.673	0.317
CHAS	0.050	-0.060	0.103	1.000	0.134	0.077	0.123	-0.141	0.057	0.017	-0.100
NOX	0.637	-0.501	0.739	0.134	1.000	-0.265	0.707	-0.746	0.542	0.615	0.103
RM	-0.142	0.307	-0.365	0.077	-0.265	1.000	-0.188	0.139	-0.096	-0.215	-0.334
AGE	0.448	-0.556	0.606	0.123	0.707	-0.188	1.000	-0.720	0.359	0.427	0.193
DIS	-0.462	0.656	-0.669	-0.141	-0.746	0.139	-0.720	1.000	-0.388	-0.444	-0.152
RAD	0.898	-0.267	0.513	0.057	0.542	-0.096	0.359	-0.388	1.000	0.873	0.387
TAX	0.826	-0.269	0.673	0.017	0.615	-0.215	0.427	-0.444	0.873	1.000	0.385
PTRATIO	0.319	-0.364	0.317	-0.100	0.103	-0.334	0.193	-0.152	0.387	0.385	1.000
AA	-0.413	0.150	-0.317	0.013	-0.358	0.108	-0.224	0.234	-0.353	-0.367	-0.090
LSTAT	0.425	-0.411	0.565	-0.009	0.537	-0.607	0.573	-0.424	0.310	0.411	0.303
MEDV	-0.286	0.332	-0.412	0.154	-0.333	0.740	-0.300	0.139	-0.218	-0.346	-0.461

In [70]:
```
# We could simply look at the correlations but a heatmap is a great wa

plt.figure(figsize =(16,10))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



In [71]:
```
# If you get stuck on what can be done with the heatmap, you can use t

sns.heatmap
```

Out[71]:
```
<function seaborn.matrix.heatmap(data, *, vmin=None, vmax=None, cmap=
None, center=None, robust=False, annot=None, fmt='.2g', annot_kws=Non
e, linewidths=0, linecolor='white', cbar=True, cbar_kws=None, cbar_ax
=None, square=False, xticklabels='auto', yticklabels='auto', mask=Non
e, ax=None, **kwargs)>
```

In [72]:
```
# Now let's say we want to decrease the amount of variables in our hea
#  Remember how to make a subset.  Try using different variables.

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

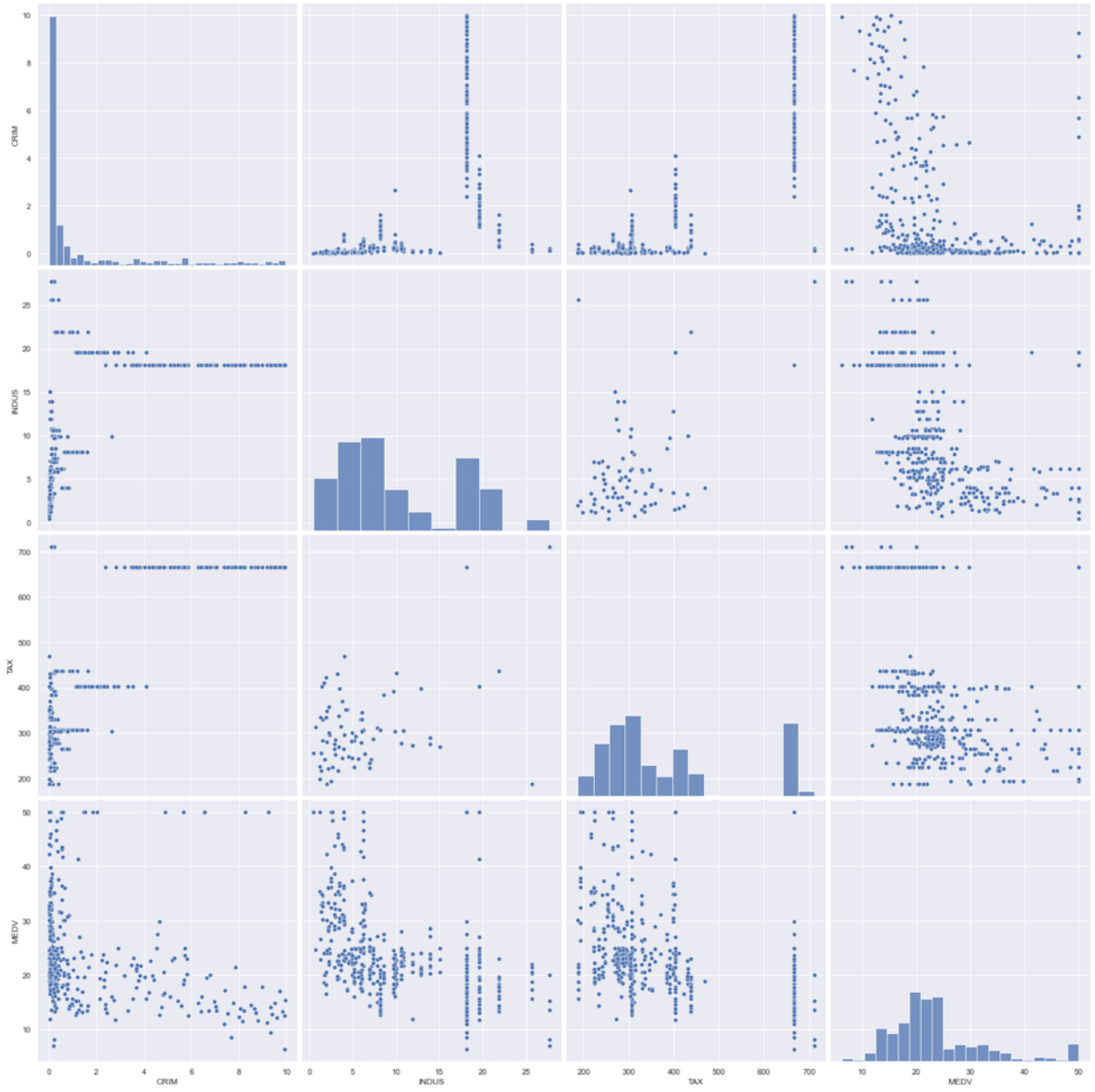
In [73]:
```
# Here we will look at the correlations for only the variables in df2.

df2.corr()
```

Out[73]:

	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

Creating a Pair Plot

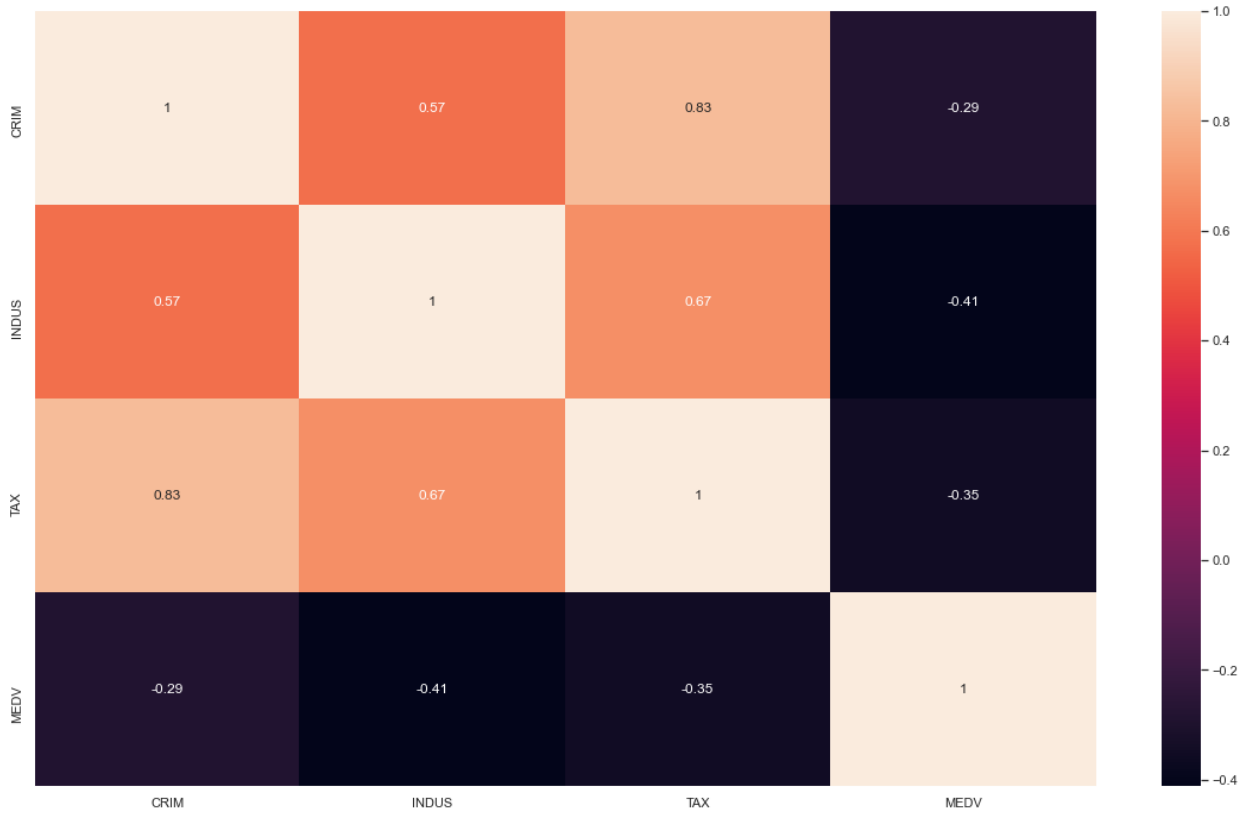
In [74]:
```
# Let's try the pairplot with only the variables in df2

sns.pairplot(df2, height=5.5);
plt.show()
```



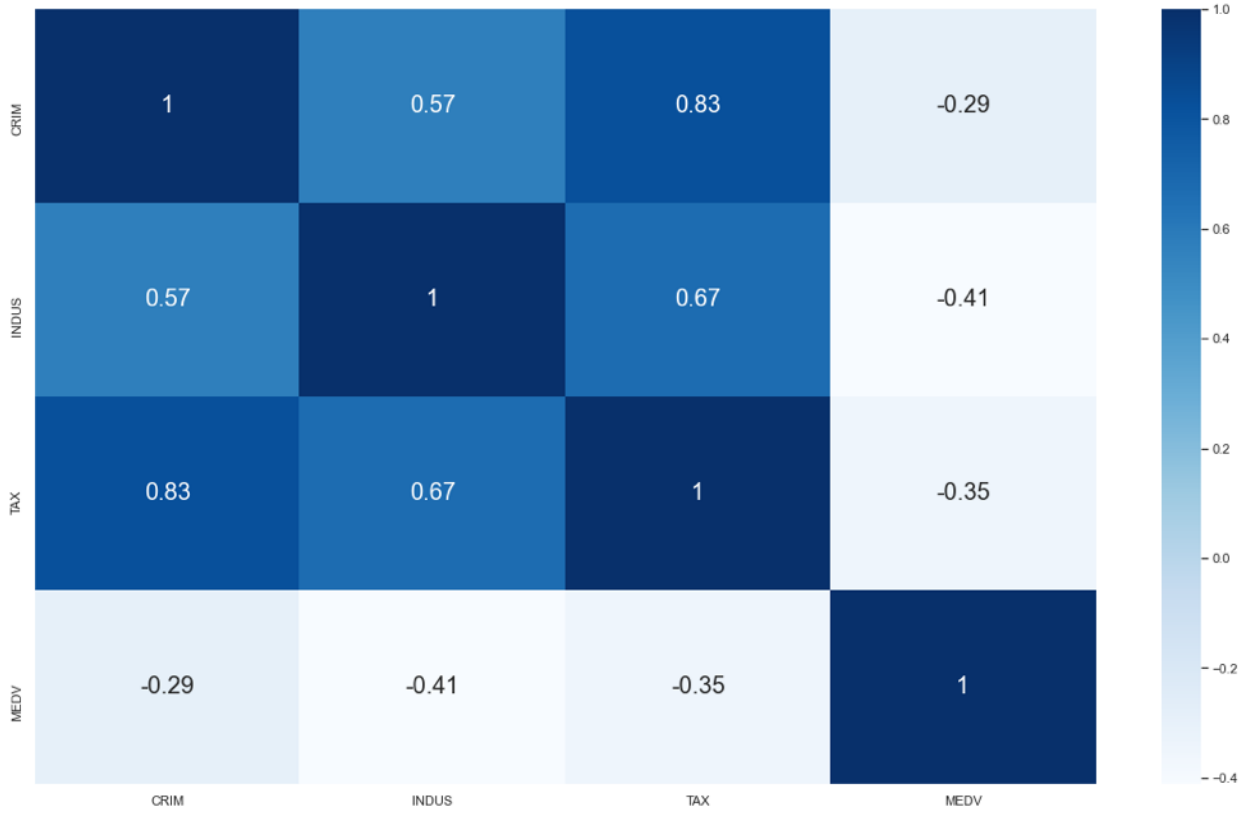
Creating a Heat Map

In [75]:
```
# Now we will make a heatmap with only the variables in df2 subset.  A

plt.figure(figsize =(20,12))
plt.figure(figsize = (20,12))
sns.heatmap(df2.corr(), annot=True)
plt.show()
```

```
<Figure size 1440x864 with 0 Axes>
```



In [76]:
```
#If you want to change the color and font, to make the labels easier t

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



Separate the Dataset into Input & Output NumPy Arrays

In [77]:
```
# Store the dataframe values into a numPy array

array= df2.values

# Separate the array into input and output components by slicing (you
# 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]
```

Spilt into Input/Output Array into Training/Testing Datasets

In [78]:
```
# Split the dataset --> training sub-dataset:  67%, and test sub-datas

test_size = 0.33

# Selection of records to inclue in which sub-dataset must be done ran

seed = 7

# Split the dataset (both input & output) into training/testing datase

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

Build and Train the Model

In [79]:
```
# 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

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

```
Intercept: 31.39342767041296
Coefficients: [ 0.09859287 -0.42388844 -0.00931847]
```

In [80]:
```
# If we want to print out the list of the coefficients with their corr
# 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', -0.4238884417716141)

('CRIM', 0.09859287239144468)

('TAX', -0.009318474474503494)
```

In [81]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=
```

Out[81]:
```
LinearRegression(n_jobs=1)
```

Calculate R-Squared

In [84]:
```
R_squared = model.score(X_test, Y_test)
print("R-squared: ", R_squared)
```

```
R-squared:  0.15473313373590358
```

Prediction

In [83]:
```
model.predict([[12,10,450]])
```

Out[83]:
```
array([24.14434421])
```

We have now trained the model. Let's use the trained model to predict the "Median value of owner-occupied homes in 1000 dollars" (MEDV).

We are using the following predictors:

- CRIM: per capita crime rate by town: 12
- INDUS: proportion of non-retail business acres per town: 10
- TAX: full-value property-tax rate per $10,000: 450

Notes: So, the model predicts that the median value of owner-occupied homes in 1000 dollars in the above suburb should be around $24,144.

Evaluate/Validate Algorithm/Model, Using K-Fold Cross-Validation

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 obtaine
# 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

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 cros

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

```
Average of all results from the K-fold Cross Validation, using negati
ve mean squared error: -64.35862748210982
```

Notes: After we train, we evaluate. We are using K-fold to determine if the model is acceptable. We pass the whole set since the system will divide it for us. We see there is a -64 avg of all errors (mean of square errors). This value would traditionally be a positive value but scikit reports this value as a negative value. If the square root would have been evaluated, the value would have been around 8.

Let's use a different scoring parameter. Here we use the Explained Variance. The best possible score is 1.0, lower values are worse.

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 su
# 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 evalua

scoring = 'explained_variance'

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

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 cros

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

```
Average of all results from the K-fold Cross Validation, using exlpai
ned variance: 0.19023822025958698
```

In [ ]: