# Drew Murray

# EDA: Python Data Visualization with Matplotlib, Pandas, and Numpy

## Import Libraries and Load Dataset

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

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
```

These are the libraries that are necessary to execute EDA for this dataset.

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



filepath = "C:/Users/dgmur/OneDrive/Desktop/ADTA 5340 Discovery and Learning with B
df = pd.read_csv (filepath)
df.head(5)
```

Out[ ]:

| | 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 |

The file path to retrieve the iris csv file is assigned to the variable, filepath. Then, pd.read_csv() with filepath inside the parentheses will convert filepath into a dataframe, this is then assigned to df. Lastly, the df.head() with a 5 inside the parentheses will ouput the first 5 rows of df.

In [ ]:
```
#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
```

Inside the print function is the function df,info(), this will output the information about df, like the data types for each column, and the number of column and rows.

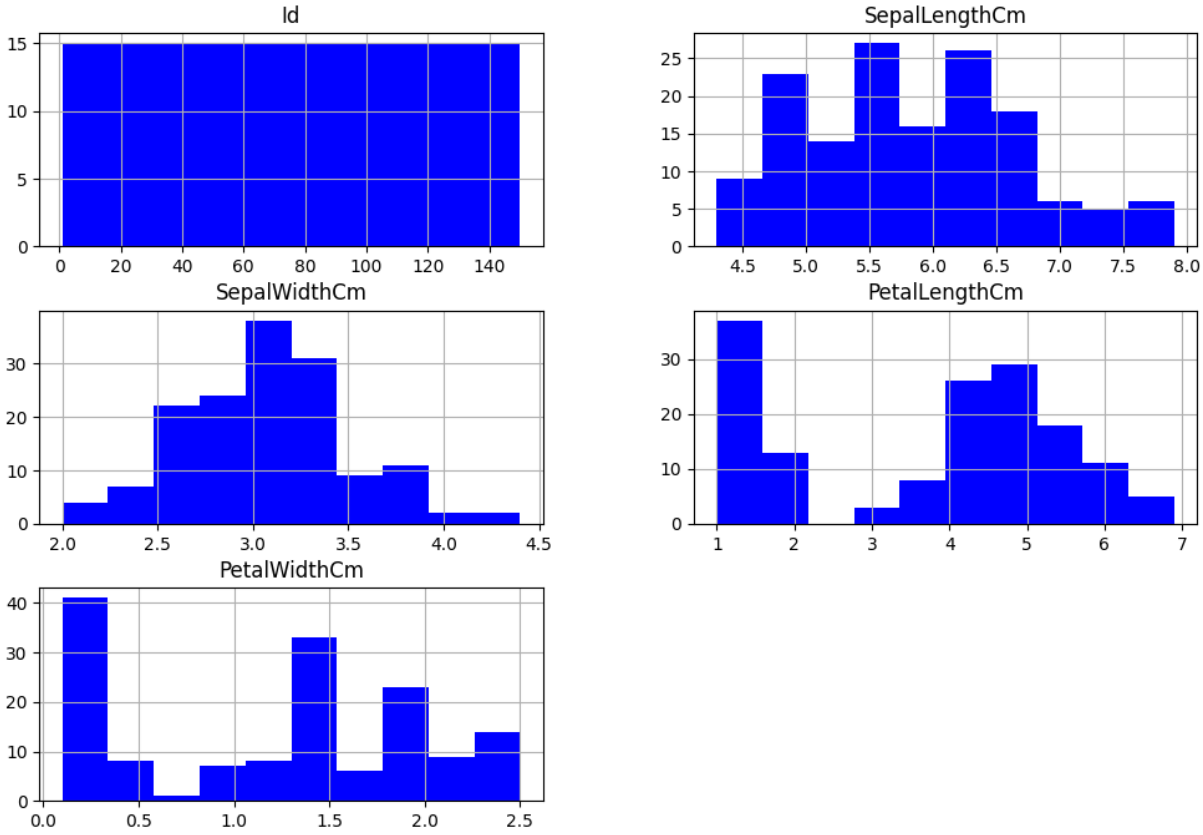
# Univariate Data Visualization

# Histograms

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

In [ ]:
```
# create a histogram
# SepalWidth  - normal dist, PetalLength - bimodal
# the normal distribution is so important easier for mathematical statisticians to
# many kinds of statistical tests can be derived from normal distributions.

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

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




The df.hist() with figsize equalling (12,8) and color equalling blue will create a 12x8 histogram where the bars are blue for each of the columns in df.

plt.show will output the histograms.

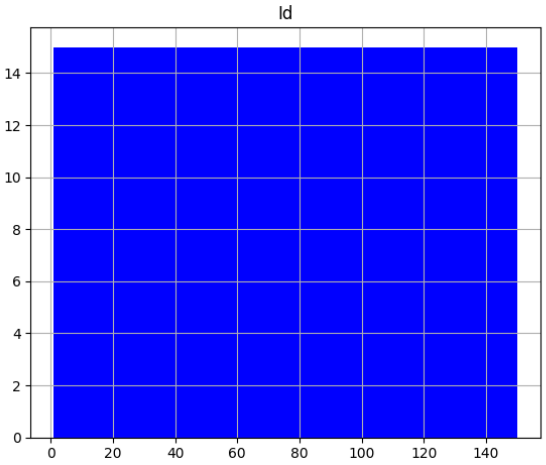
In [ ]:
```
# here we want to see the different Species

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

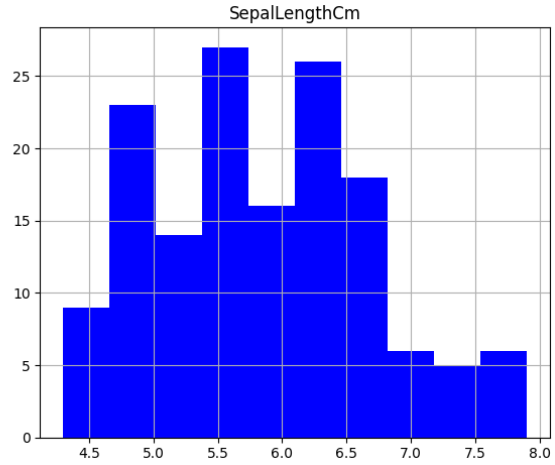
```
Species
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
dtype: int64
```

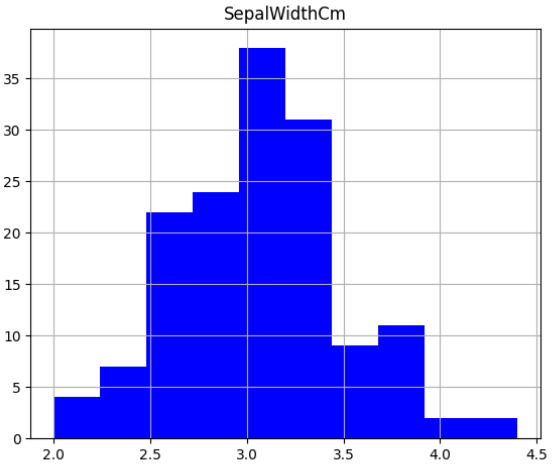
Inside the print function, is df.groupby() with 'species' inside the parentheses, followed by the size(). This will output the number of observations based on each factor for the column, species.

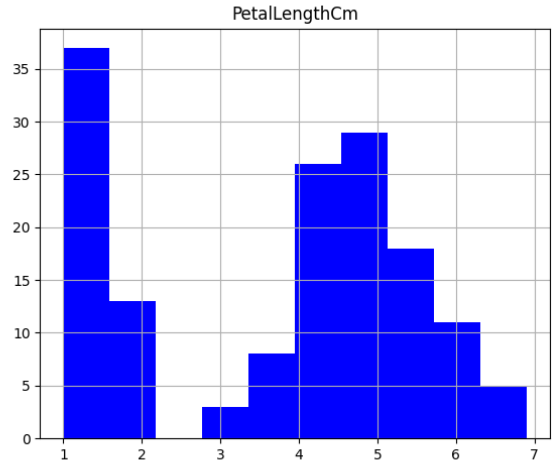
In [ ]:
```
# Histograms are great when we would like to show the distribution of the data we a

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

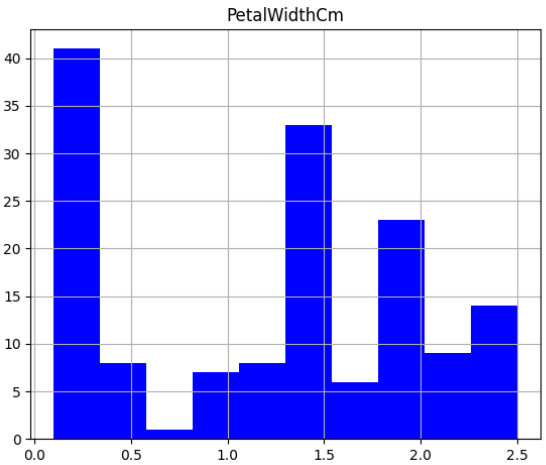
Out[ ]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```




The df.hist() with figsize equalling (15,9) and color equalling blue will create a 15x9 histogram where the bars are blue for each of the columns in df.

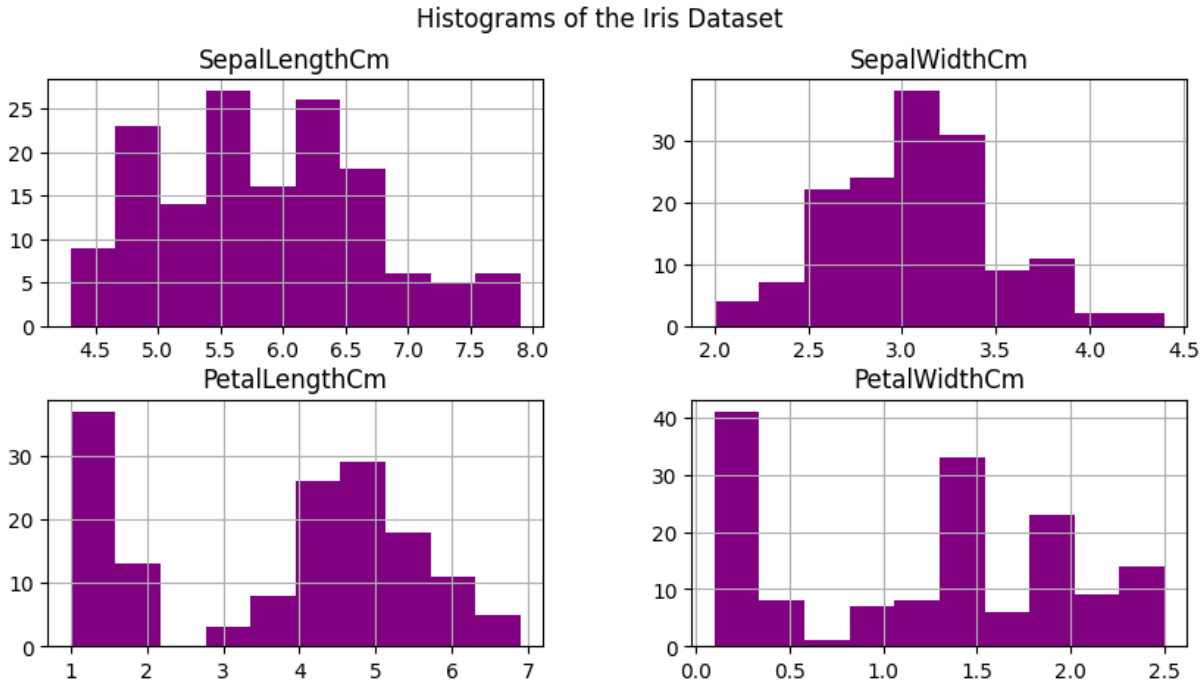
plt.show will output the histograms.

In [ ]:
```
# in the above code, we see the variable "Id" is included in the analysis.  In orde

df.__delitem__('Id')
```

The df.**delitem**() with id inside the parentheses will exclude the id column from the analysis.

In [ ]:
```
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 changed the color t
```

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




The df.hist() with figsize equalling (10,5) and color equalling purple will create a 10x5 histogram where the purple are blue for each of the columns in df. The ply.suptitle() with "histograms of the Iris Dataset" inside the parentheses will add a title for the histograms

plt.show will output the histograms, along with the title.

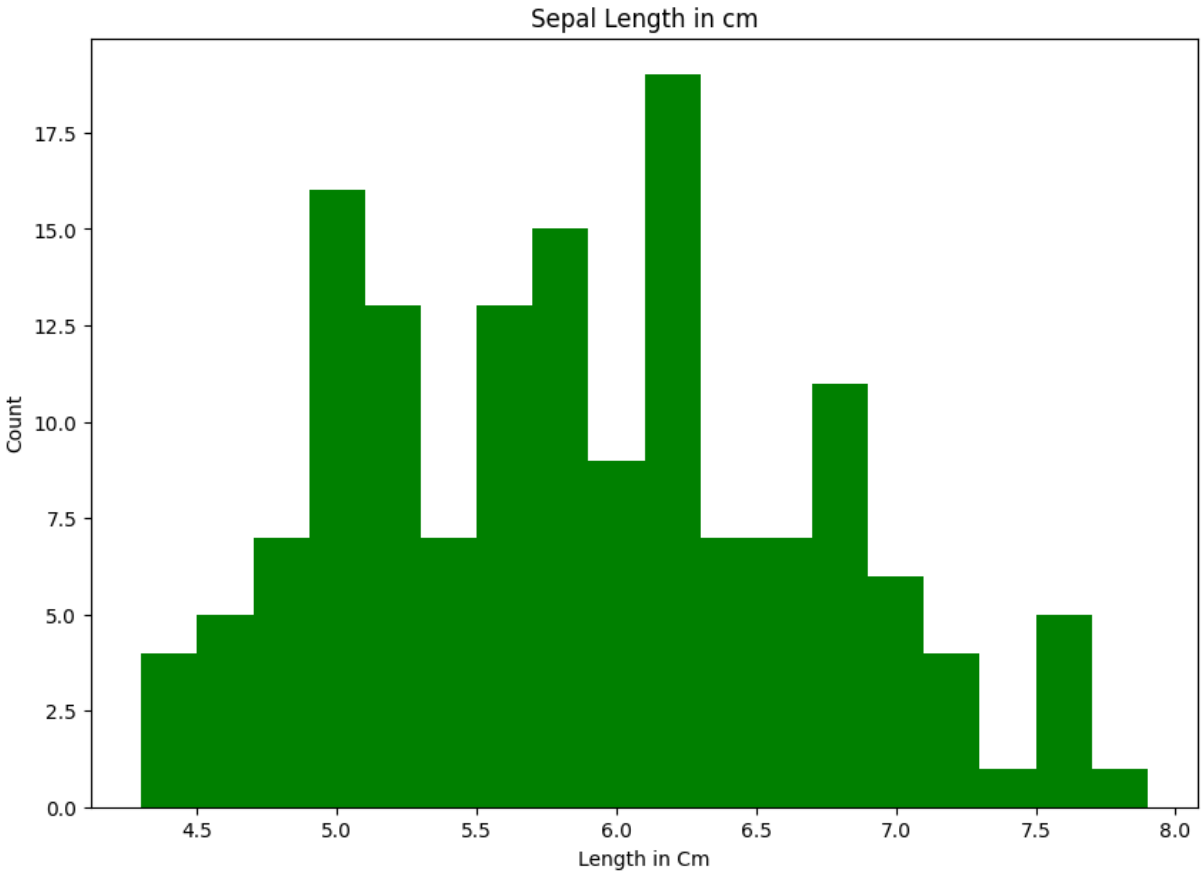
# Let's say we want to make changes to the histograms.

# We will look at only the PetalLengthCm Variable variable.

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

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

```
plt.xlabel("Length in Cm")
plt.ylabel("Count")
```

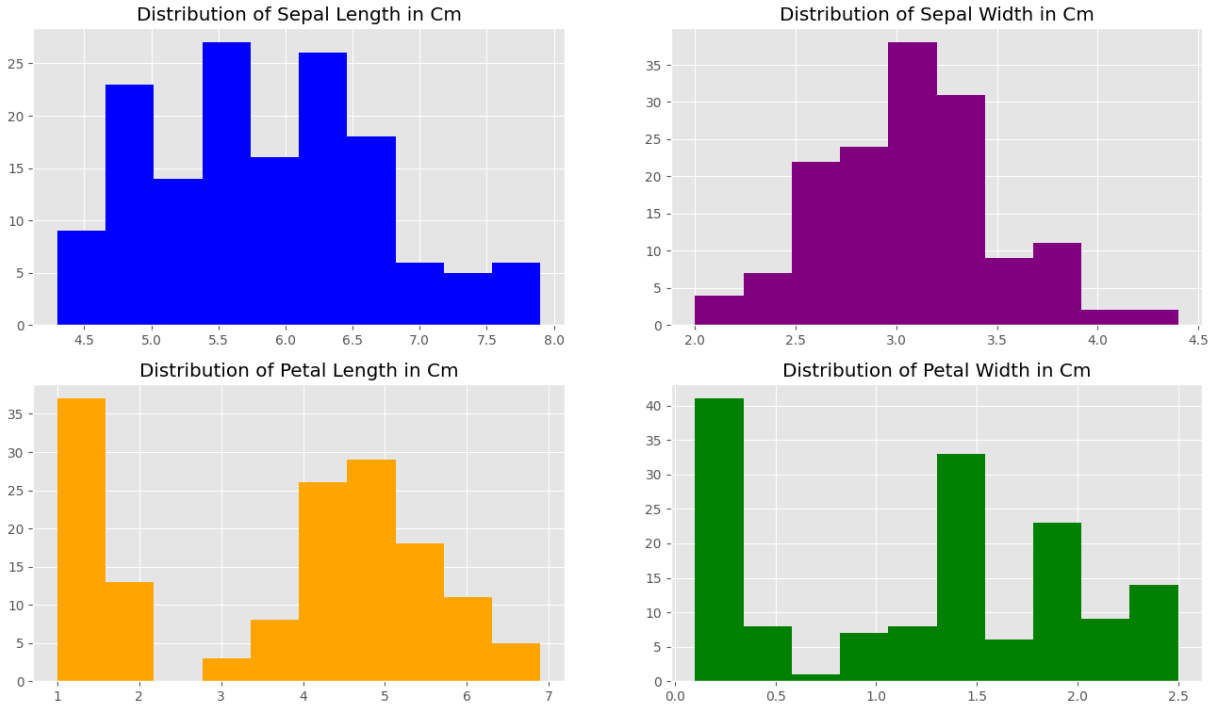
Out[ ]: Text(0, 0.5, 'Count')




- The plt.figure() with figsize equal to (10,7) inside the parentheses will create a 10x7 for the upcoming plot.
- df followed by a pair brackets with the string, "SepalLengthCm" will select the SepalLengthCm and its values, this will be then assigned to the variable x. Next, the plt.hist () function containing x followed by bins equal to 18 and then color equalling green will create a histogram for SepalLengthCm that has 18 bins and there bars are green. The title of the plot is "Sepal Length in cm" by using plt.title(), the x axis is labeled, "Length in Cm" using the plt.xlabel(), and then the y axis is labeled "Count" by using the plt.ylabel().

In [ ]:
```
# here we are isolating the variable and giving each one a color and a specific num
# grammar of graphics plot

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');
```




- The plt.style.use that contains the string, ggplot, will use the ggplot style for the upcoming plots. Then inside the plt.subplots() is a 2 followed by a 2 which is then followed by figsize equal to (16,9). This will create a 2x2 plot format where each plot is 16x9 for the upcoming plots.
- axes followed by a pair of brackets that contains 0 and 0, which is then followed by the set_title(), where it contains the string, Distribution of Sepal Length in Cm. This will assign the title, "Distribution of Sepal Length in Cm" to the [0,0] position. Then axes followed a bracket that contains 0,0, which is then followed hist (). Inside the hist() is df['SepalLengthCm], followed by the bins equal to 20, and the color equal to blue. This will create a histogram in the [0,0] position where the it has 20 bins and the bars are blue.
- axes followed by a pair of brackets that contains 0 and 1, which is then followed by the set_title(), where it contains the string, Distribution of Sepal Width in Cm. This will assign the title, "Distribution of Sepal Width in Cm" to the [0,1] position. Then axes followed a bracket that contains 0 and 1, which is then followed hist (). Inside the hist() is df['SepalWidthCm], followed by the bins equal to 20, and the color equal to purple. This will create a histogram in the [0,1] position where the it has 20 bins and the bars are purple.
- axes followed by a pair of brackets that contains 1 and 0, which is then followed by the set_title(), where it contains the string, Distribution of Petal Length in Cm. This will assign the title, "Distribution of Petal Length in Cm" to the [1,0] position. Then axes followed a bracket that contains 1 and 0, which is then followed hist (). Inside the hist() is

df['PetalLengthCm], followed by the bins equal to 20, and the color equal to orange. This will create a histogram in the [1,0] position where the it has 20 bins and the bars are orange.

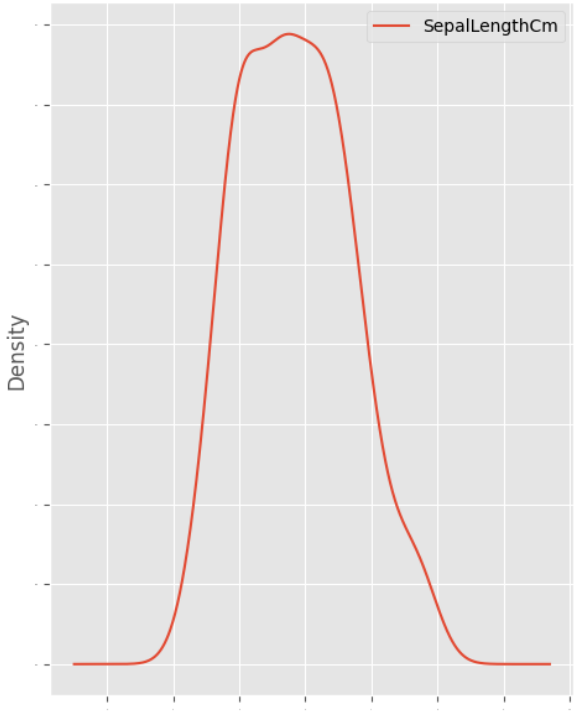
- axes followed by a pair of brackets that contains 1 and 1, which is then followed by the set_title(), where it contains the string, Distribution of Petal Width in Cm. This will assign the title, "Distribution of Petal Width in Cm" to the [1,1] position. Then axes followed a bracket that contains 1 and 1, which is then followed hist (). Inside the hist() is df['PetalWidthCm], followed by the bins equal to 20, and the color equal to green. This will create a histogram in the [1,1] position where the it has 20 bins and the bars are green.

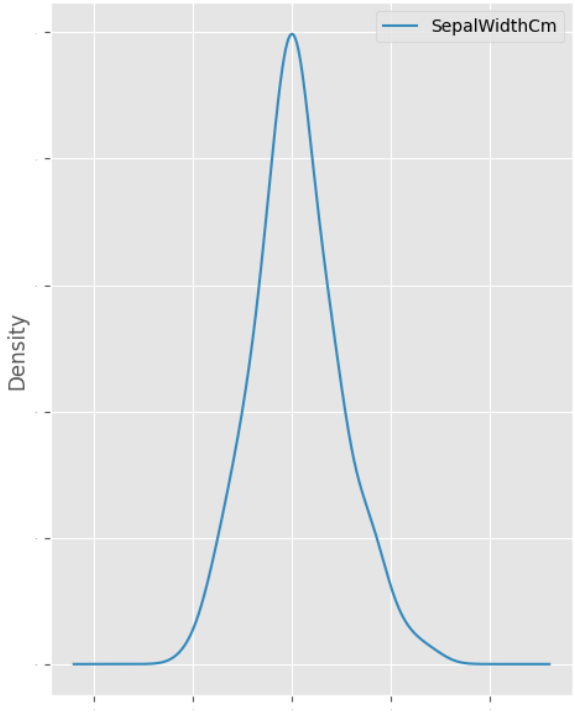
## Density Plots -

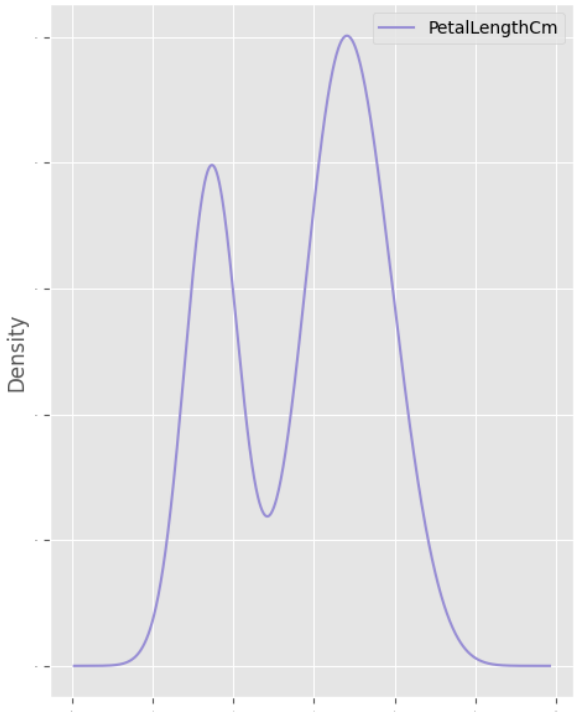
- A Density Plot visualizes the distribution of data over a time period or a continuous interval. This chart is a variation of a Histogram but it smooths out the noise made by binning.

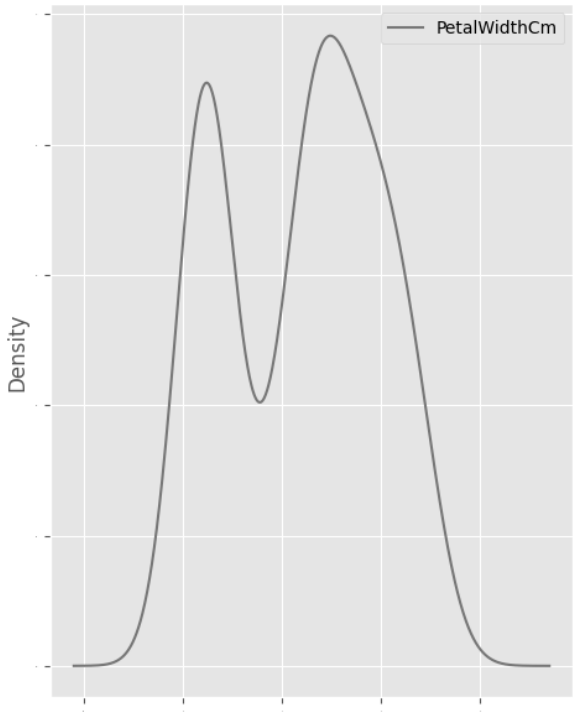
- Density Plots have a slight advantage over Histograms since they're better at determining the distribution shape and, as mentioned above, they are not affected by the number of bins used (each bar used in a typical histogram). As we saw above a Histogram with only 10 bins wouldn't produce a distinguishable enough shape of distribution as a 20-bin Histogram would. With Density Plots, this isn't an issue.

In [ ]:
```
# create the density plot
# If subplots=True is specified, plots for each column are drawn as subplots

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


SepalLengthCm
Density

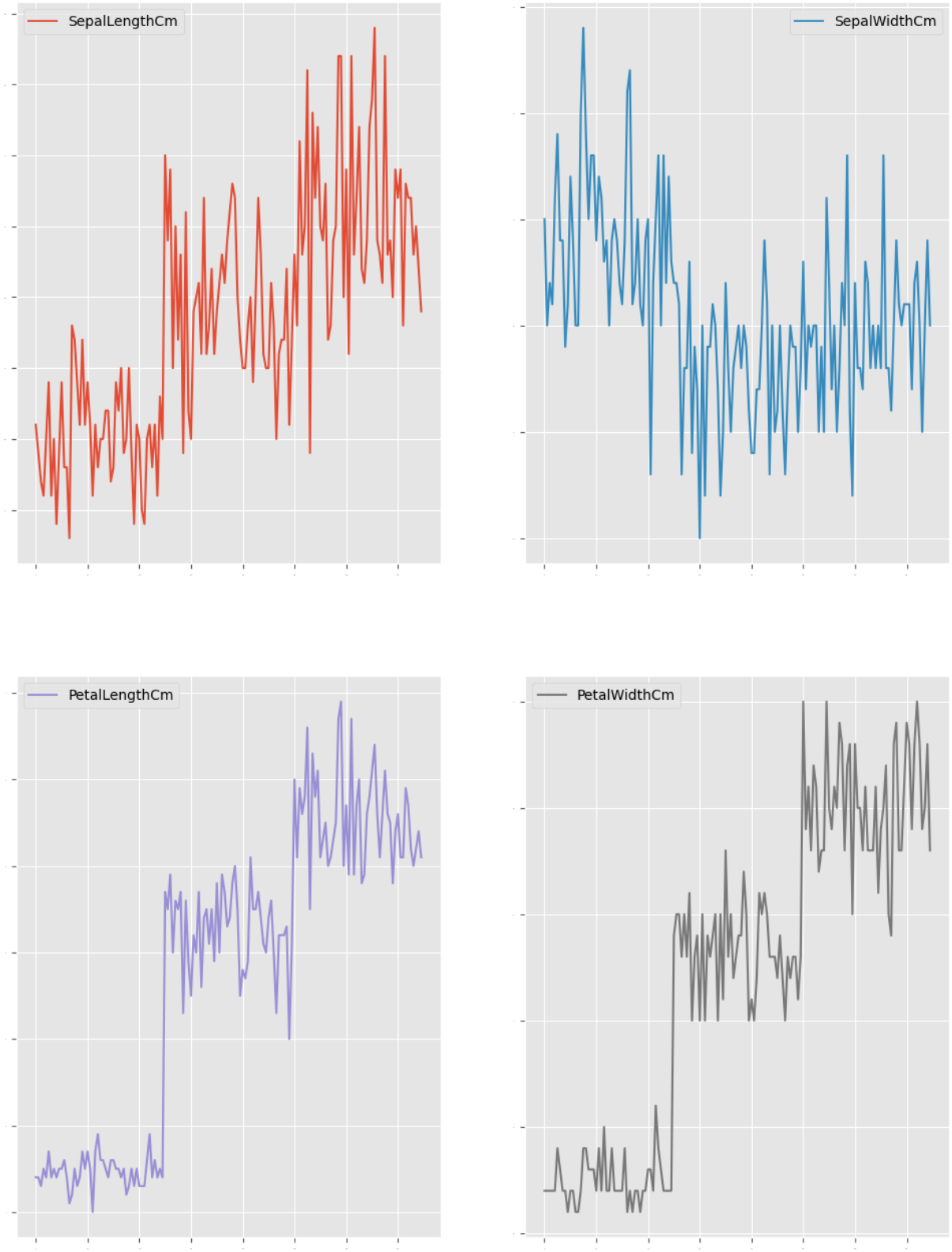

SepalWidthCm
Density


PetalLengthCm
Density


PetalWidthCm
Density

The df.plot() function will create a plot based on what's inside its parentheses. Inside the df.plot() is the parameter kind which is equal to the string, density, which is then followed by subplots equal to True, followed by layout equal to (2,2). sharex = False, Legend equal to True, then font size equal to 1,and then lastly figsize is equal to (12,16). This will create a 4 density plots that are in 2x2 format and have a 12x16 dimension that consist of the 4 columns in the df variable, the font size of the labels for each of the plots are a 1. The plt.show() function will output the plot.

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

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




The df.plot() function will create a plot based on what's inside its parentheses. Inside the df.plot() is the parameter kind which is equal to the string, line, which is then followed by subplots equal to True, followed by layout equal to (2,2). sharex = False, Legend equal to True, then font size equal to 1,and then lastly figsize is equal to (12,16). This will create a 4 line graph that are in a 2x2 and have a 12x16 dimension that consist of the 4 columns in the

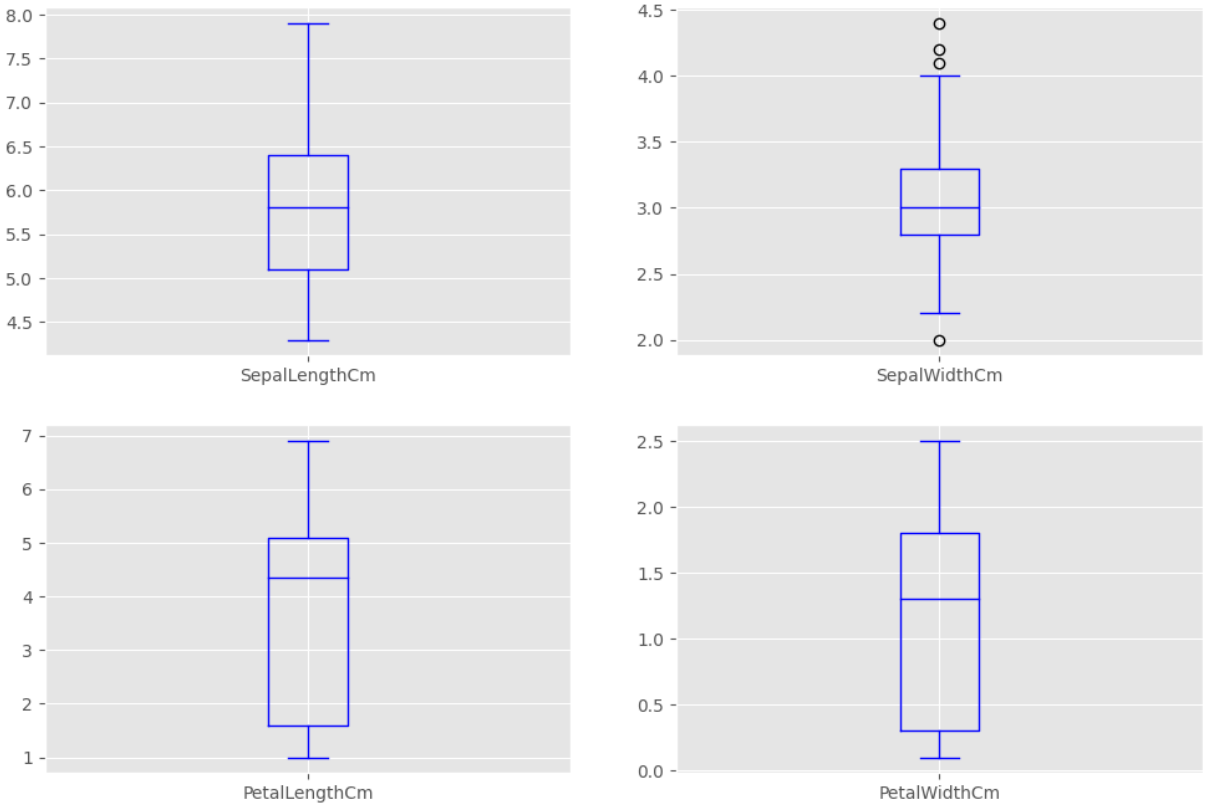
df variable, the font size of the labels for each of the plots are a 1. The plt.show() function will output the plot.

# Boxplots

- A box plot is a very good plot to understand the spread, median, and outliers of data Below is an illustration of the boxplot
- Boxplot.jpeg

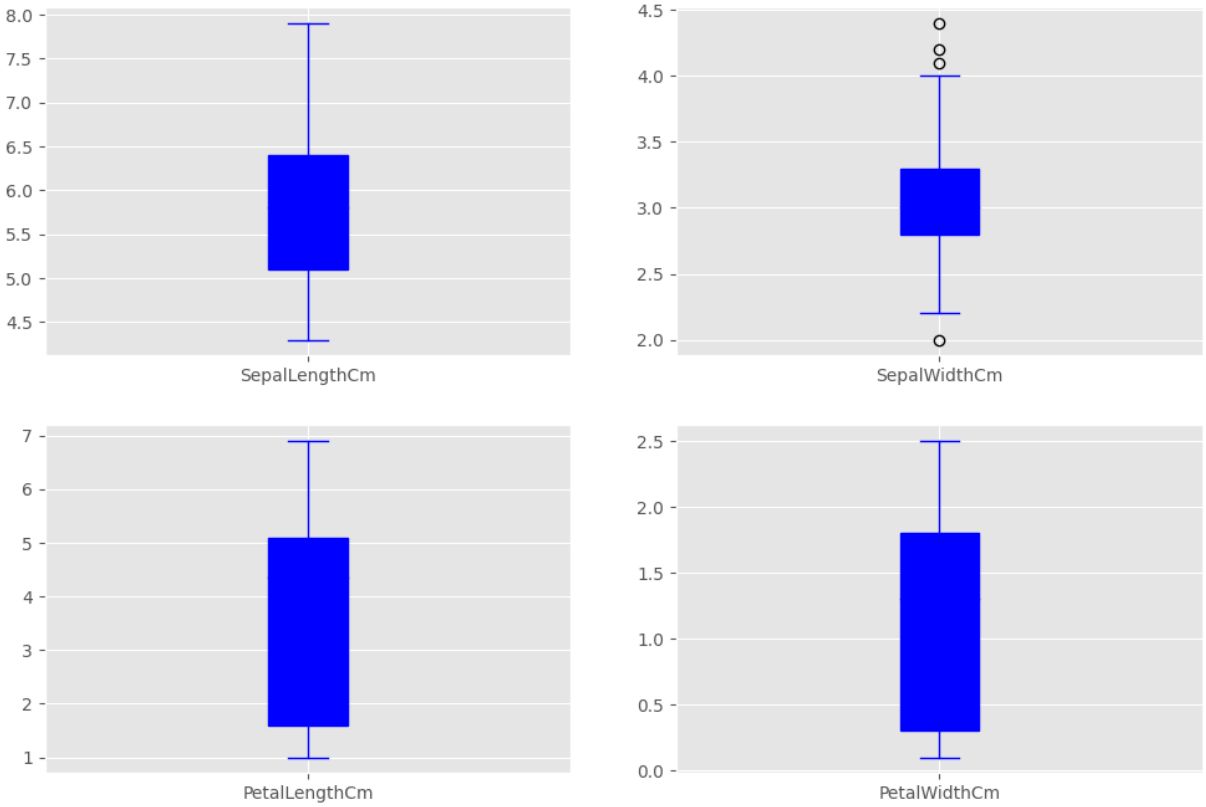
1. Q3: This is the 75th percentile value of the data. It's also called the upper hinge.
2. Q1: This is the 25th percentile value of the data. It's also called the lower hinge.
3. Box: This is also called a step. It's the difference between the upper hinge and the lower hinge.
4. Median: This is the midpoint 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 [ ]:
```
# create a box plot

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




The df.plot() function will create a plot based on what's inside its parentheses. Inside the df.plot() is the parameter kind which is equal to the string, box, which is then followed by subplots equal to True, followed by layout equal to (2,2). sharex = False, Legend equal to True, sharey equal to false , color equal to blue and then lastly figsize is equal to (12,8). This will create 4 boxplots that are in a 2x2 format and have a 12x8 dimension that consist of the 4 columns in the df variable, the font size of the labels for each of the plots are a 1, and the outline of each boxplot will be blue. The plt.show() function will output the plot.

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

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




The df.plot() function will create a plot based on what's inside its parentheses. Inside the df.plot() is the parameter kind which is equal to the string, box, which is then followed by subplots equal to True, followed by layout equal to (2,2). sharex = False, Legend equal to True, sharey equal to false , color equal to blue, then figsize is equal to (12,8), lastly the patch_artist parameter will equal true. This will create 4 boxplots that are in a 2x2 format and have a 12x8 dimension that consist of the 4 columns in the df variable, the font size of the labels for each of the plots are a 1, the outline of each boxplot will be blue, and due to the patch_artist parameter being true the boxplots will be filled in with the color blue as well. The plt.show() function will output the plot.

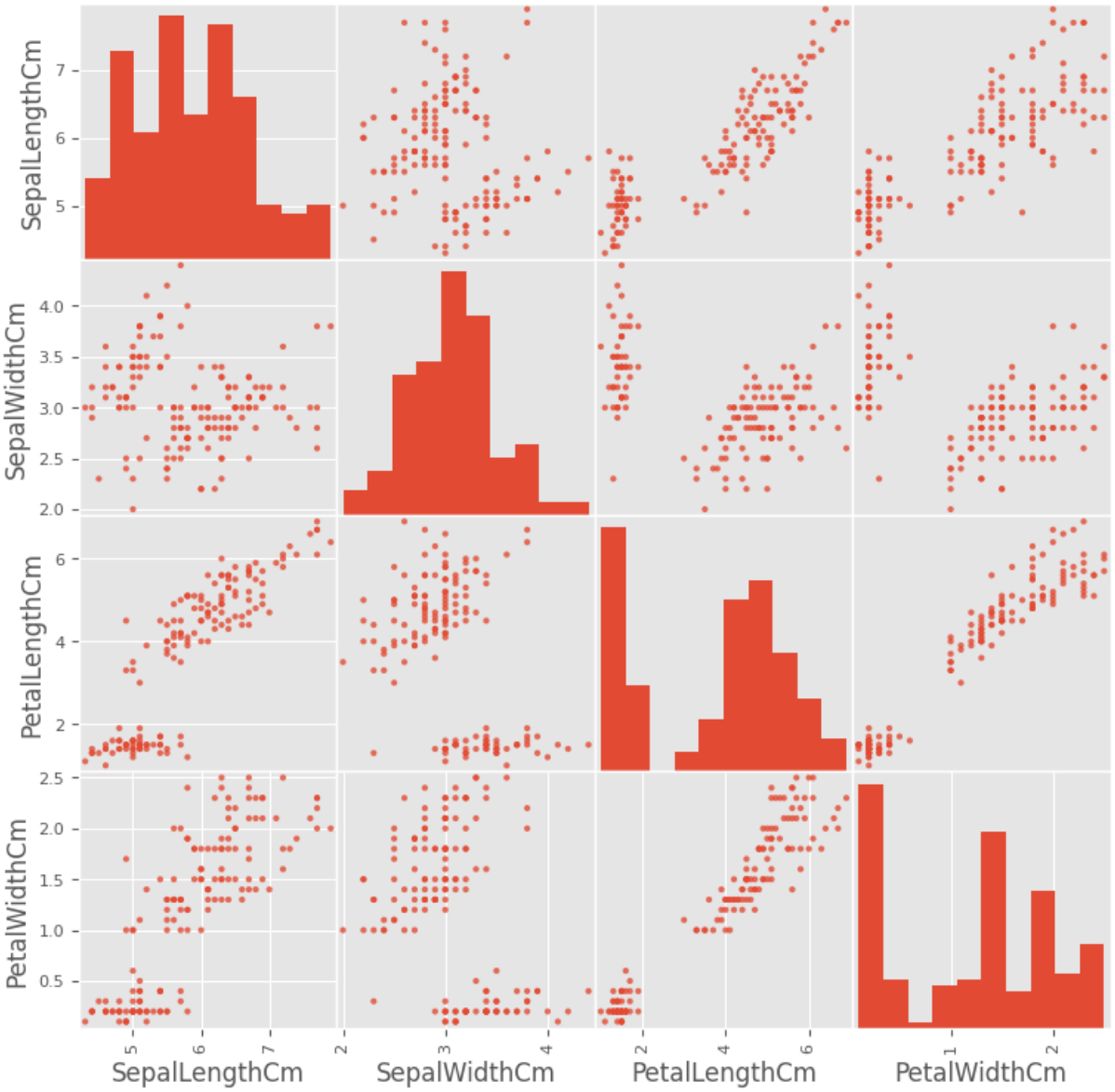
# Multivariate Data Visualization

## 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 [ ]:
```
# create a scatter matrix plot
# alpha = Amount of transparency applied
```

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



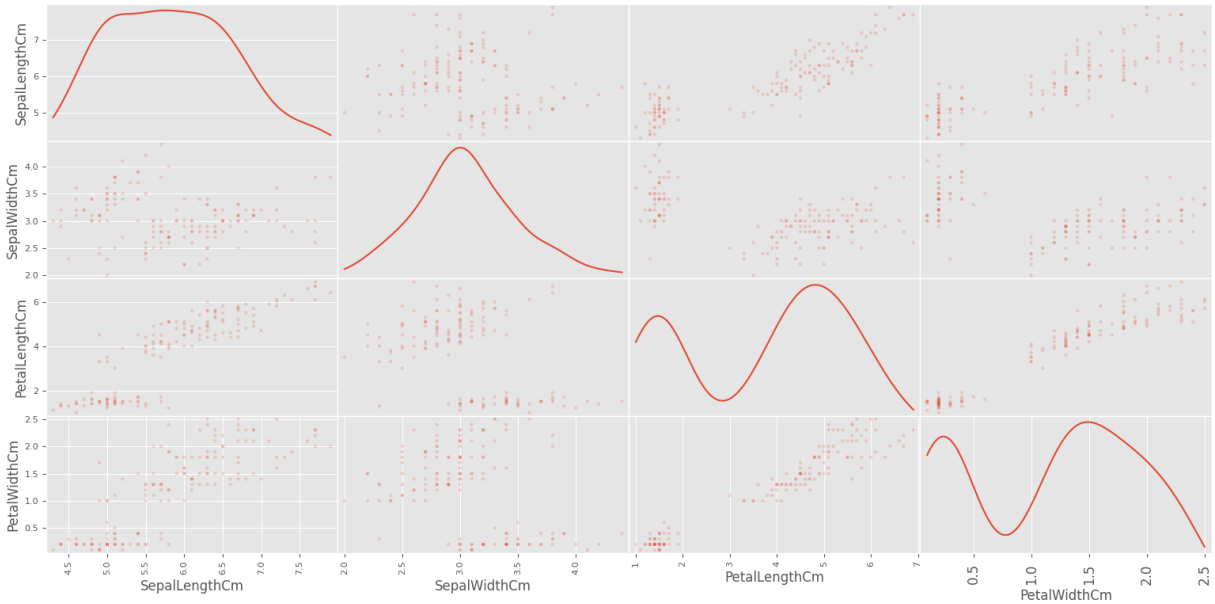

The scatter_matrix () is used to create a scatter matrix where the diagonal is a histogram (default). Inside the function is df, followed by alpha equalling 0.8 and then the figsize equalling to (9,9). This will make scatter matrix out of the columns in df that are in 9x9 dimensions and the scatterplots have a transparency level of .08. Then the plt.show() function will output that plot.

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

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


The scatter_matrix () is used to create a scatter matrix where the diagonal is a histogram (default). Inside the function is df, followed by alpha equalling 0.8, followed by the figsize equalling to (9,9) and the color parameter equal to purple. This will make scatter matrix out of the columns in df that are in 9x9 dimensions and scatterplots have a transparency level of .08. Due to the color parameter being equal to purple, the scatterplots in the scatter_matrix will be the color purple. Then the plt.show() function will output that plot.

In [ ]:
```
# Here you are adding a suptitle.
# Pick between 'kde' and 'hist' for either Kernel Density Estimation or Histogram p
# KDE = a density estimator is an algorithm which seeks to model the probability di

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)
```

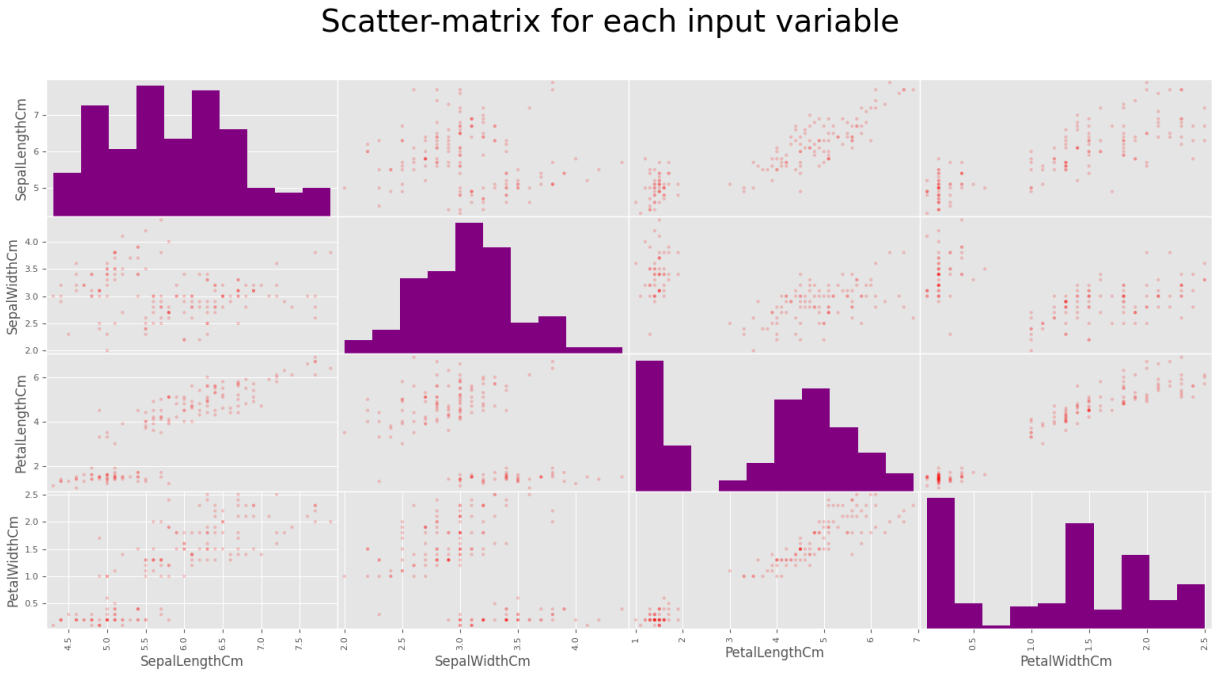
Scatter-matrix for each input variable



The scatter_matrix () is used to create a scatter matrix where the diagonal is a histogram (default). Inside the function is df, followed by alpha equalling 0.2, followed by the diagonal equalling kde, then the figsize equalling to (19,9). This will make scatter matrix out of the columns in df that are in 19x9 dimensions and scatterplots have a transparency level of .02. Due to the diagonal parameter being equal to kde, the diagonal in the scatter_matrix will be a kde, which presents the estimate probability distribution of df. Next, the sup.title() function with the string, "Scatter-matrix for each input variable, will add a title to the plot. Then, inside the plt.tick_params is the lablesize parameter equal to 12, and the pad equalling 6. This will change the font size of the labels in the plot to 12 along with changing the amount of distance between the values on the axis and the labels on the axis by a value of 6. Then the plt.show() function will output that plot.

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




The scatter_matrix () is used to create a scatter matrix where the diagonal is a histogram (default). Inside the function is df, followed by alpha equalling 0.2, followed by the figsize equalling to (19,9), followed by the c parameter equal to red, and then the hist_kwds parameter equalling a dictonary where color is the key and a list containing the string, purple is the value. This will make scatter matrix out of the columns in df that are in 19x9 dimensions, scatterplots are red and have a transparency level of .02. Also the diagonal of histograms are purple due to the hist_kwds parameter. Inside the plt.suptitle() function is the string, "Scatter-matrix for each input variable", followed by the fontsize parameter equal to

28. This will add a title called "Scatter-matrix for each inoput variable" to the scatter matrix that has a font size value of 28.

# Drew Murray

# Machine Learning Supervised Linear Regression

In [ ]:
```
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
```

These are the necessary libraries to conduct exploratory data analysis and 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 [ ]:
```
housingfile = "C:/Users/dgmur/OneDrive/Desktop/ADTA 5340 Discovery and Learning wit
```

The file path for the housing dataset is assigned to the variable, housingfile.

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

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

The pd.read_csv is used to convert csv files into a data frame. Inside the pd.read_csv() is housingfile, followed by the header parameter equalling none. This will convert housing file into a dataframe and removing the headers for the columns in the csv, this is then assigned to df.

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

Out[ ]:

| | 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 |

The df.head() function will output the first 5 rows in the dataframe, df.

# Label the Columns since there are no headers

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

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

A list of strings is assigned to a variable called, col_names.

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

df.columns = col_names
```

The list of strings col_names is assigned to the variable df.columns. The list of strings in col_names are now the headers for the columns in df.

## Look at the dataframe

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

df.head()
```

Out[ ]:

| | 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 | |

The df.head() function will output the first 5 rows of df.

## Preprocess the Dataset

Clean the data: Find and Mark Missing Values

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

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

Out[ ]:
```
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
```

The df.isnull() followed the sum() function will ouput the amount of missing values for each column.

# Perform the Exploratory Data Analysis (EDA)

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

print(df.shape)
```

```
(452, 14)
```

Inside the print function is the function, df.shape, this will output the number of rows followed by the number of columns of df in complex number format.

In [ ]:
```
# 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
```

Inside the print function is the function df.dtypes(). This will output the data type for each column of df.

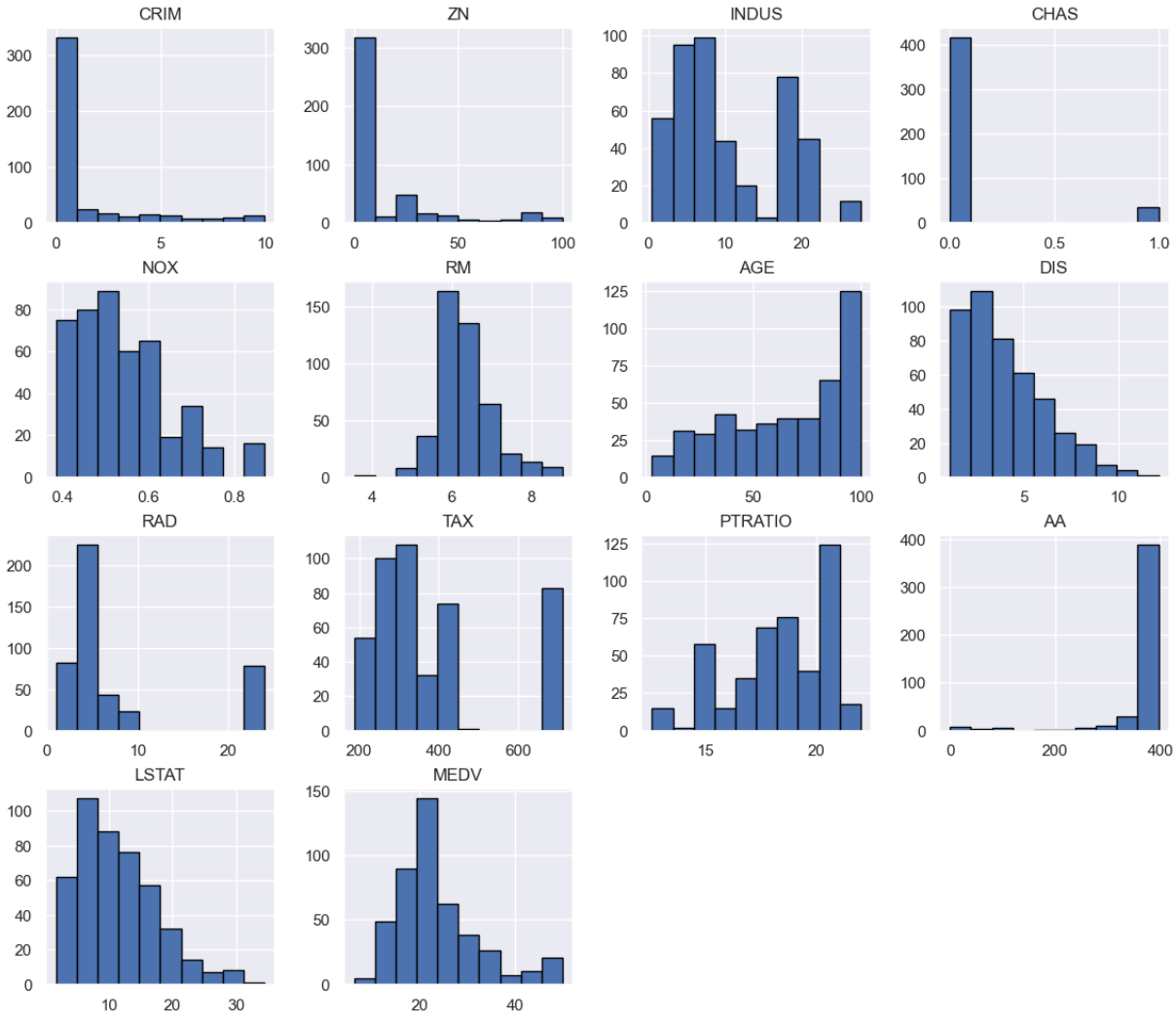
In [ ]:
```
# Obtain the summary statistics of the data

print(df.describe())
```

```
              CRIM          ZN       INDUS        CHAS         NOX          RM  \
count  452.000000  452.000000  452.000000  452.000000  452.000000  452.000000
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  452.000000
mean    65.557965    4.043570    7.823009  377.442478   18.247124  369.826504
std     28.127025    2.090492    7.543494  151.327573    2.200064   68.554439
min      2.900000    1.129600    1.000000  187.000000   12.600000    0.320000
25%     40.950000    2.354750    4.000000  276.750000   16.800000  377.717500
50%     71.800000    3.550400    5.000000  307.000000   18.600000  392.080000
75%     91.625000    5.401100    7.000000  411.000000   20.200000  396.157500
max    100.000000   12.126500   24.000000  711.000000   22.000000  396.900000

            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
```

Inside the print function is the function df.describe(), this will output summary statistics of df. This includes the count, mean, standard deviation, minimum value, 25% quartile, median (50% quartile), 75% quartile and the maximum value of df.

# Creating a Histogram

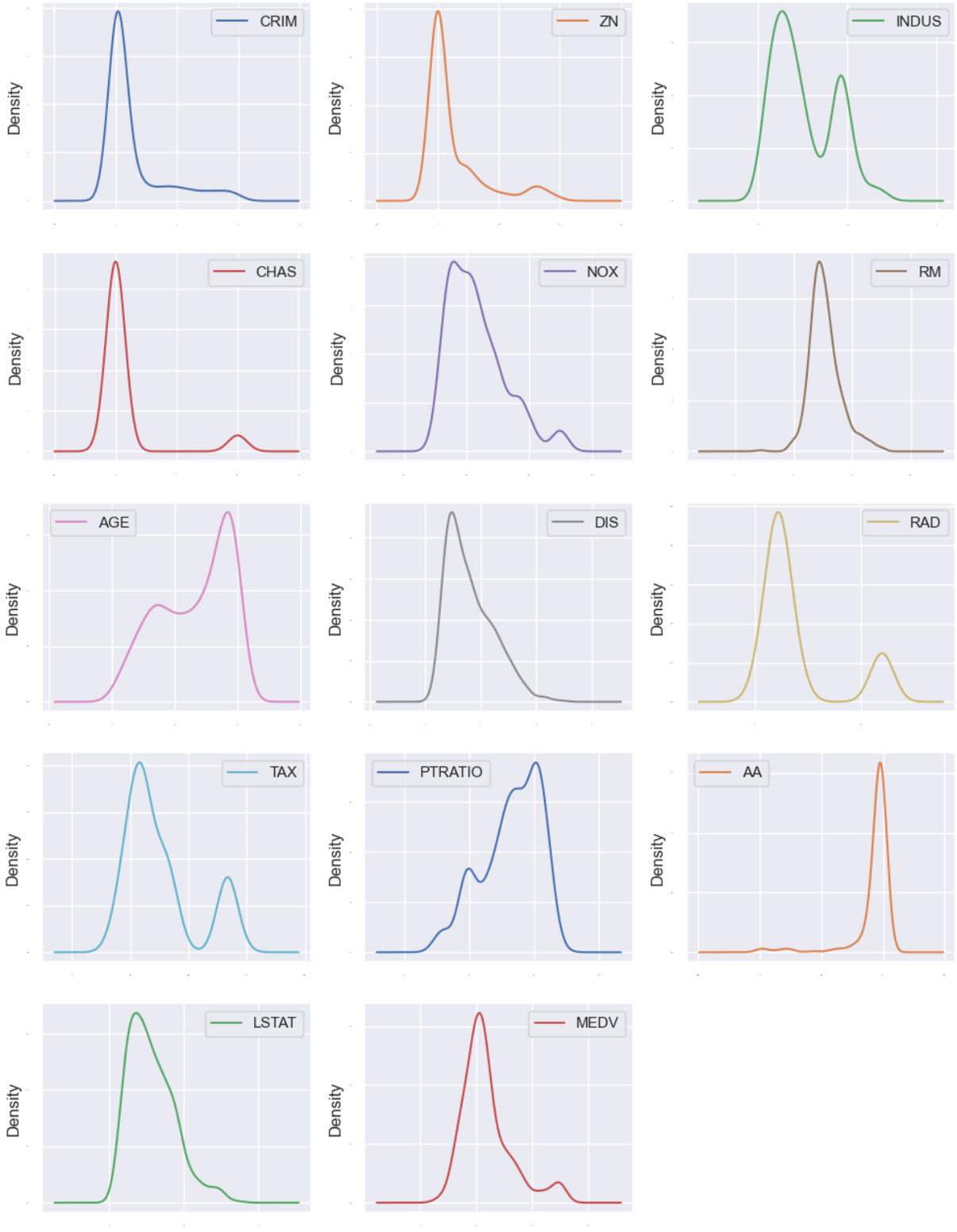
In [ ]:
```
# Plot histogram for each variable. I encourage you to work with the histogram. Rem

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




The df.hist( ) function is used to create a histogram for each column of df. Inside the function is the parameter edgecolor that is equal to black followed by figsize equalling (14,12). This will make the outline of the bars in the histograms black and change to the figure size to a 14x12 value. The plt.show() function will output the histograms.

# Create a Density Plot

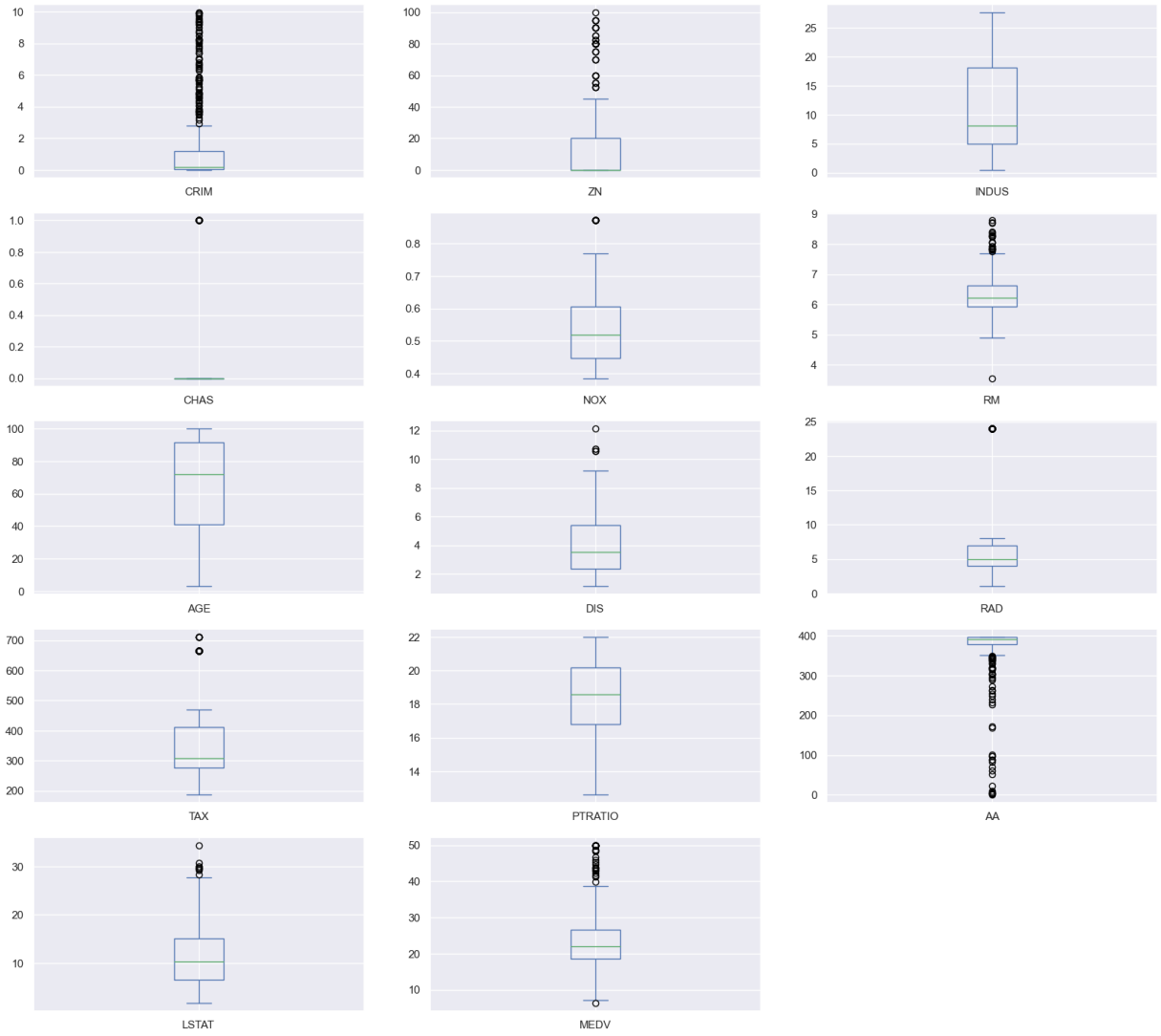
In [ ]:
```
# Density plots
# Notes:  14 numeric variable, at least 14 plots, layout (5,3): 5 rows, each row wi
# When subplots have a shared x-axis along a column, only the x tick labels of the

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




The df.plot() function is used to create a plot using df. Inside df.plot() is the parameter kind which is equal to density, followed by the subplot parameter equalling true, then layout equal to (5,3), sharex equalling false, legend equal to true, the fontsize equal to 1 and then the figsize equal to (12,16). This will create 14 density plots for all the numerical values in a 5x3 format that includes a legend for each plot that have a figure size value of 12x16 and a fontsize value of 1. The plt.show() function will output the density plots.

# Creating a Box Plot

In [ ]:
```
# Boxplots

df.plot(kind="box", subplots=True, layout=(5,3), sharex=False, figsize=(20,18))
plt.show()
```

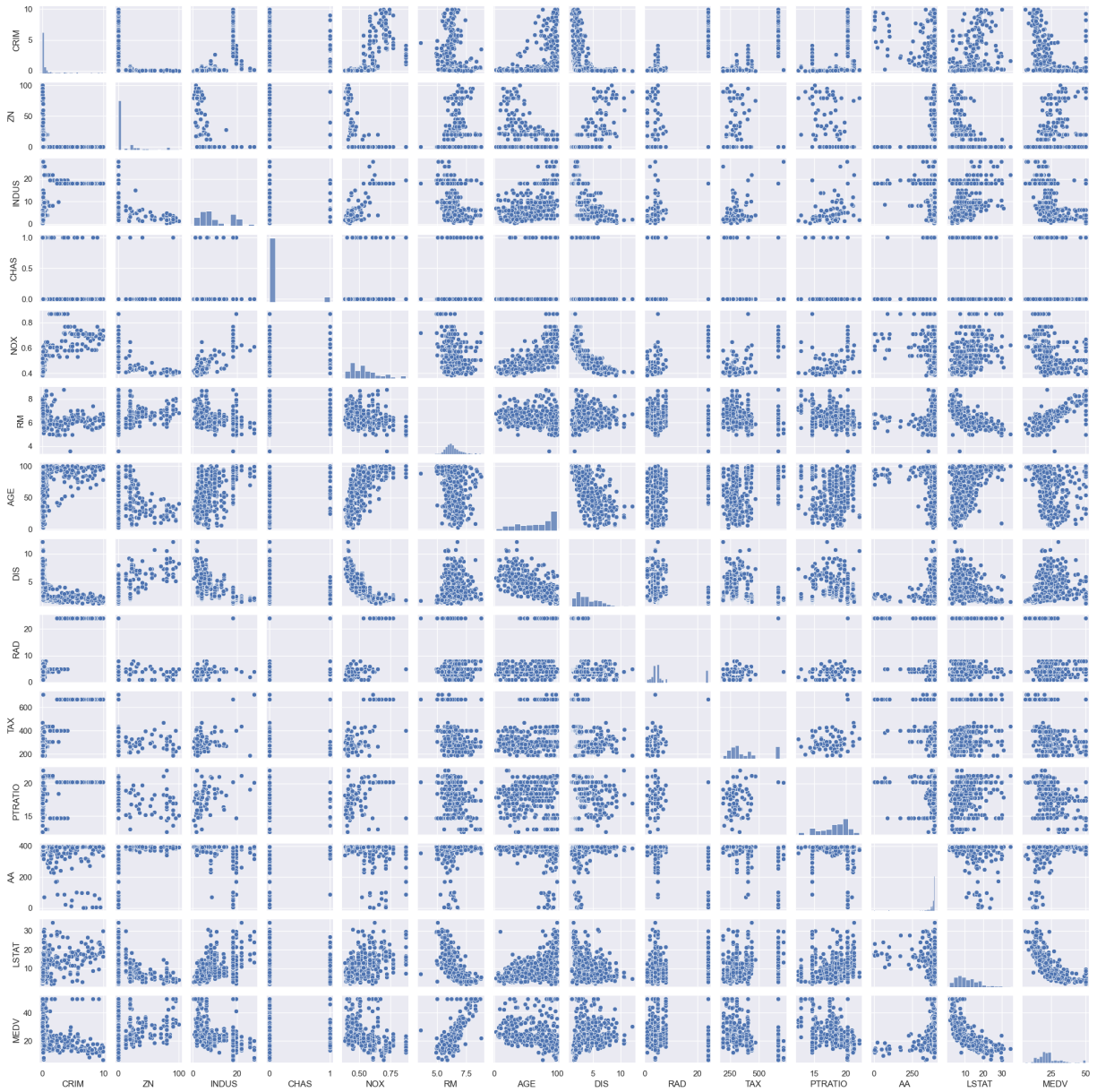



The df.plot() function is used to create a plot using df. Inside df.plot() is the parameter kind which is equal to box, followed by the subplot parameter equalling true, then layout equal to (5,3), sharex equalling false, and then figsize equal to (12,18). This will create a boxplot for each of the numerical variables in df in a 5x3 format where each boxplot has a figure size value of 20x18. The plt.show() will output the boxplots.

# Correlation Analysis and Feature Selection

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

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




The sns.pairplot() function will create a pairplot, which is used to evaluate the correlation between the numerical variables. Inside the sns.pairplot() is df followed by height equal to 1.5. This will create a pairplot based on the numerical variables of df, and the height will be adjusted to the value 1.5. The plt.show() function will output the pairplots.

# Correlations

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

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

inside the pair of single quotations is a pair of curly brackets containing a colon and the .3f which is then followed by the format function and assigned to the variable,

pd.options.display.float_format. This will convert the outputs that are a float data type and derive from the pandas library to a 3 decimal place float.

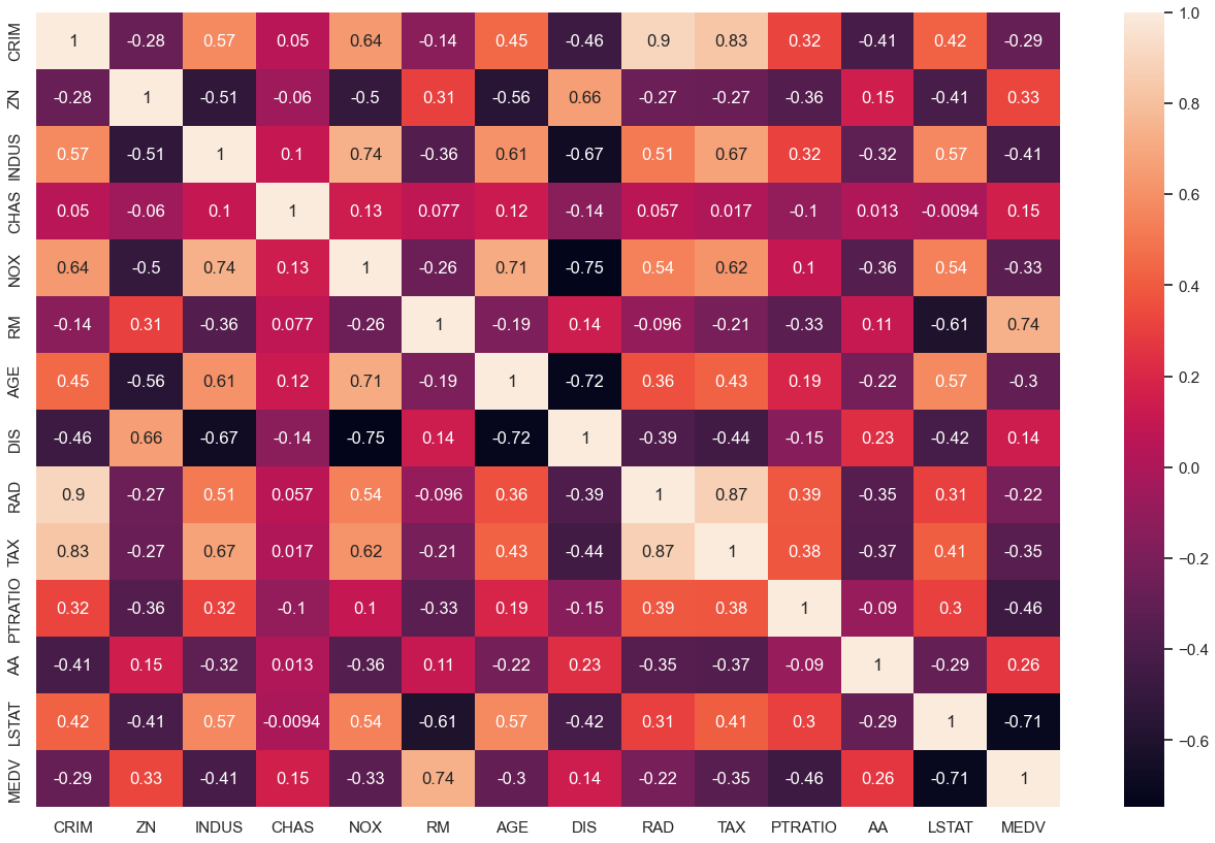
In [ ]:
```
# Here we will get the correlations, with only 3 decimals.

df.corr()
```

Out[ ]:

| | CRIM | ZN | INDUS | CHAS | NOX | RM | AGE | DIS | RAD | TAX | PTRA |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **CRIM** | 1.000 | -0.281 | 0.574 | 0.050 | 0.637 | -0.142 | 0.448 | -0.462 | 0.898 | 0.826 | 0 |
| **ZN** | -0.281 | 1.000 | -0.514 | -0.060 | -0.501 | 0.307 | -0.556 | 0.656 | -0.267 | -0.269 | -0 |
| **INDUS** | 0.574 | -0.514 | 1.000 | 0.103 | 0.739 | -0.365 | 0.606 | -0.669 | 0.513 | 0.673 | 0 |
| **CHAS** | 0.050 | -0.060 | 0.103 | 1.000 | 0.134 | 0.077 | 0.123 | -0.141 | 0.057 | 0.017 | -0 |
| **NOX** | 0.637 | -0.501 | 0.739 | 0.134 | 1.000 | -0.265 | 0.707 | -0.746 | 0.542 | 0.615 | 0 |
| **RM** | -0.142 | 0.307 | -0.365 | 0.077 | -0.265 | 1.000 | -0.188 | 0.139 | -0.096 | -0.215 | -0 |
| **AGE** | 0.448 | -0.556 | 0.606 | 0.123 | 0.707 | -0.188 | 1.000 | -0.720 | 0.359 | 0.427 | 0 |
| **DIS** | -0.462 | 0.656 | -0.669 | -0.141 | -0.746 | 0.139 | -0.720 | 1.000 | -0.388 | -0.444 | -0 |
| **RAD** | 0.898 | -0.267 | 0.513 | 0.057 | 0.542 | -0.096 | 0.359 | -0.388 | 1.000 | 0.873 | 0 |
| **TAX** | 0.826 | -0.269 | 0.673 | 0.017 | 0.615 | -0.215 | 0.427 | -0.444 | 0.873 | 1.000 | 0 |
| **PTRATIO** | 0.319 | -0.364 | 0.317 | -0.100 | 0.103 | -0.334 | 0.193 | -0.152 | 0.387 | 0.385 | 1 |
| **AA** | -0.413 | 0.150 | -0.317 | 0.013 | -0.358 | 0.108 | -0.224 | 0.234 | -0.353 | -0.367 | -0 |
| **LSTAT** | 0.425 | -0.411 | 0.565 | -0.009 | 0.537 | -0.607 | 0.573 | -0.424 | 0.310 | 0.411 | 0 |
| **MEDV** | -0.286 | 0.332 | -0.412 | 0.154 | -0.333 | 0.740 | -0.300 | 0.139 | -0.218 | -0.346 | -0 |

The df.corr() function will output the correlations of the numerical variables in df in a dataframe format.

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




The plt.figure () function containing figsize equalling (16,10), will set the size of the plot to a have a 12x16 value. Next the sns.heatmap () function will create a heatmap, inside the parentheses is df.corr() followed by the annot parameter equalling true. This will create a heatmap for the correlations of df where the values are labeled. Lastly, plt.show() will output the heatmap based on the parameters set in the plt.figure() and sns.heatmap().

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

sns.heatmap?
```

```
Signature:
sns.heatmap(
    data,
    *,
    vmin=None,
    vmax=None,
    cmap=None,
    center=None,
    robust=False,
    annot=None,
    fmt='.2g',
    annot_kws=None,
    linewidths=0,
    linecolor='white',
    cbar=True,
    cbar_kws=None,
    cbar_ax=None,
    square=False,
    xticklabels='auto',
    yticklabels='auto',
    mask=None,
    ax=None,
    **kwargs,
)
Docstring:
Plot rectangular data as a color-encoded matrix.

This is an Axes-level function and will draw the heatmap into the
currently-active Axes if none is provided to the ``ax`` argument.  Part of
this Axes space will be taken and used to plot a colormap, unless ``cbar``
is False or a separate Axes is provided to ``cbar_ax``.

Parameters
----------
data : rectangular dataset
    2D dataset that can be coerced into an ndarray. If a Pandas DataFrame
    is provided, the index/column information will be used to label the
    columns and rows.
vmin, vmax : floats, optional
    Values to anchor the colormap, otherwise they are inferred from the
    data and other keyword arguments.
cmap : matplotlib colormap name or object, or list of colors, optional
    The mapping from data values to color space. If not provided, the
    default will depend on whether ``center`` is set.
center : float, optional
    The value at which to center the colormap when plotting divergent data.
    Using this parameter will change the default ``cmap`` if none is
    specified.
robust : bool, optional
    If True and ``vmin`` or ``vmax`` are absent, the colormap range is
    computed with robust quantiles instead of the extreme values.
annot : bool or rectangular dataset, optional
    If True, write the data value in each cell. If an array-like with the
    same shape as ``data``, then use this to annotate the heatmap instead
    of the data. Note that DataFrames will match on position, not index.
fmt : str, optional
```

```
    String formatting code to use when adding annotations.
annot_kws : dict of key, value mappings, optional
    Keyword arguments for :meth:`matplotlib.axes.Axes.text` when ``annot``
    is True.
linewidths : float, optional
    Width of the lines that will divide each cell.
linecolor : color, optional
    Color of the lines that will divide each cell.
cbar : bool, optional
    Whether to draw a colorbar.
cbar_kws : dict of key, value mappings, optional
    Keyword arguments for :meth:`matplotlib.figure.Figure.colorbar`.
cbar_ax : matplotlib Axes, optional
    Axes in which to draw the colorbar, otherwise take space from the
    main Axes.
square : bool, optional
    If True, set the Axes aspect to "equal" so each cell will be
    square-shaped.
xticklabels, yticklabels : "auto", bool, list-like, or int, optional
    If True, plot the column names of the dataframe. If False, don't plot
    the column names. If list-like, plot these alternate labels as the
    xticklabels. If an integer, use the column names but plot only every
    n label. If "auto", try to densely plot non-overlapping labels.
mask : bool array or DataFrame, optional
    If passed, data will not be shown in cells where ``mask`` is True.
    Cells with missing values are automatically masked.
ax : matplotlib Axes, optional
    Axes in which to draw the plot, otherwise use the currently-active
    Axes.
kwargs : other keyword arguments
    All other keyword arguments are passed to
    :meth:`matplotlib.axes.Axes.pcolormesh`.

Returns
-------
ax : matplotlib Axes
    Axes object with the heatmap.

See Also
--------
clustermap : Plot a matrix using hierarchical clustering to arrange the
             rows and columns.

Examples
--------

.. include:: ../docstrings/heatmap.rst
File:      c:\users\dgmur\appdata\local\packages\pythonsoftwarefoundation.python.3.9
_qbz5n2kfra8p0\localcache\local-packages\python39\site-packages\seaborn\matrix.py
Type:      function
```

The sns.heatmap function followed by a question mark will output the syntax of the sns.heatmap to help the user know what parameters are in the sns.heatmap function.

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

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

df followed by a pair of square brackets that contain a list of strings, where the strings are some of the names of the variables of df, this is then assigned to a variable called df2. This will subset the dataset to only include the variables of CRIM, INDUS, TAX, and MEDV.

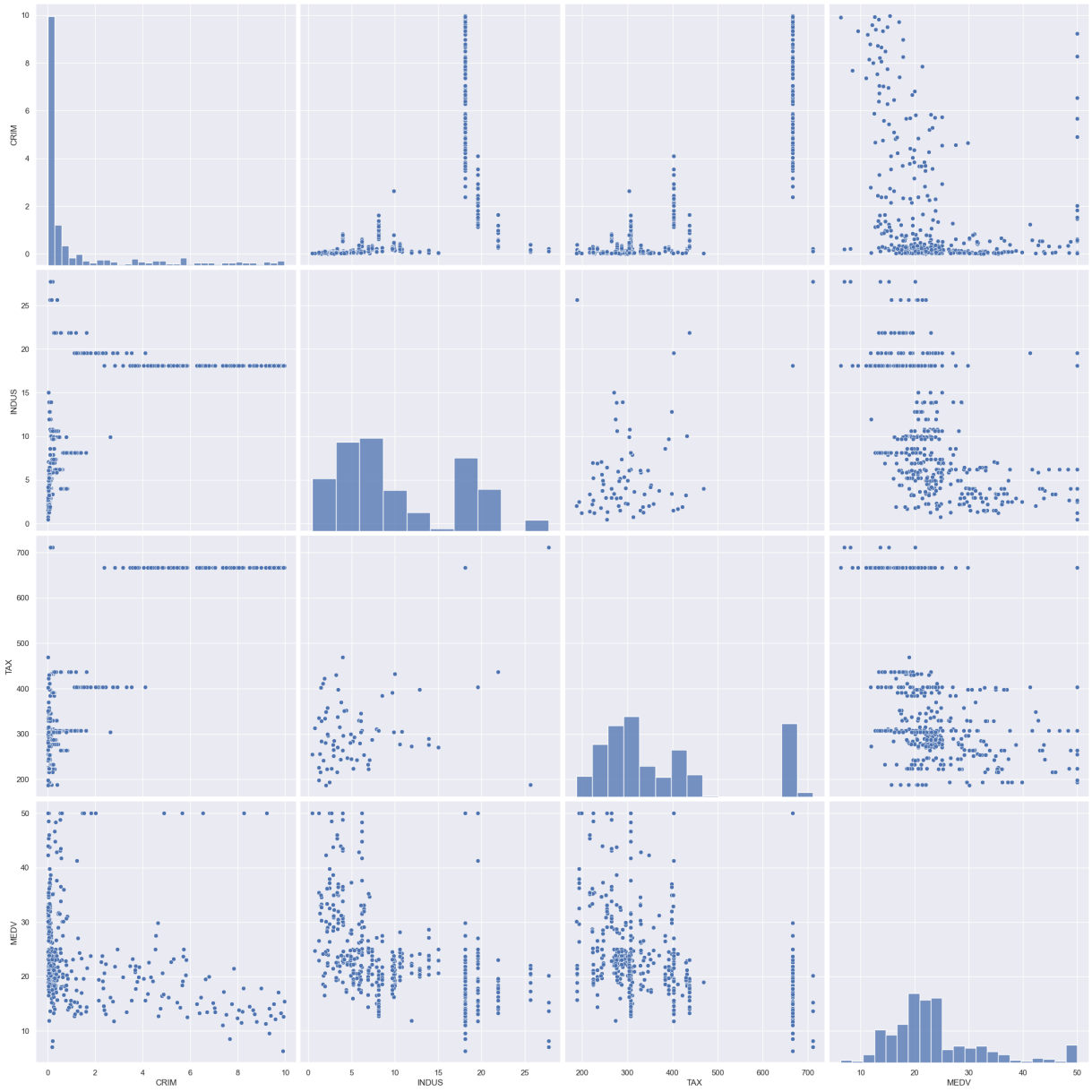
In [ ]:
```
# Here we will look at the correlations for only the variables in df2.

df2.corr()
```

Out[ ]:

| | 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 |

The df2.corr() function will output the correlation of the numerical variables in df2 in a dataframe format.

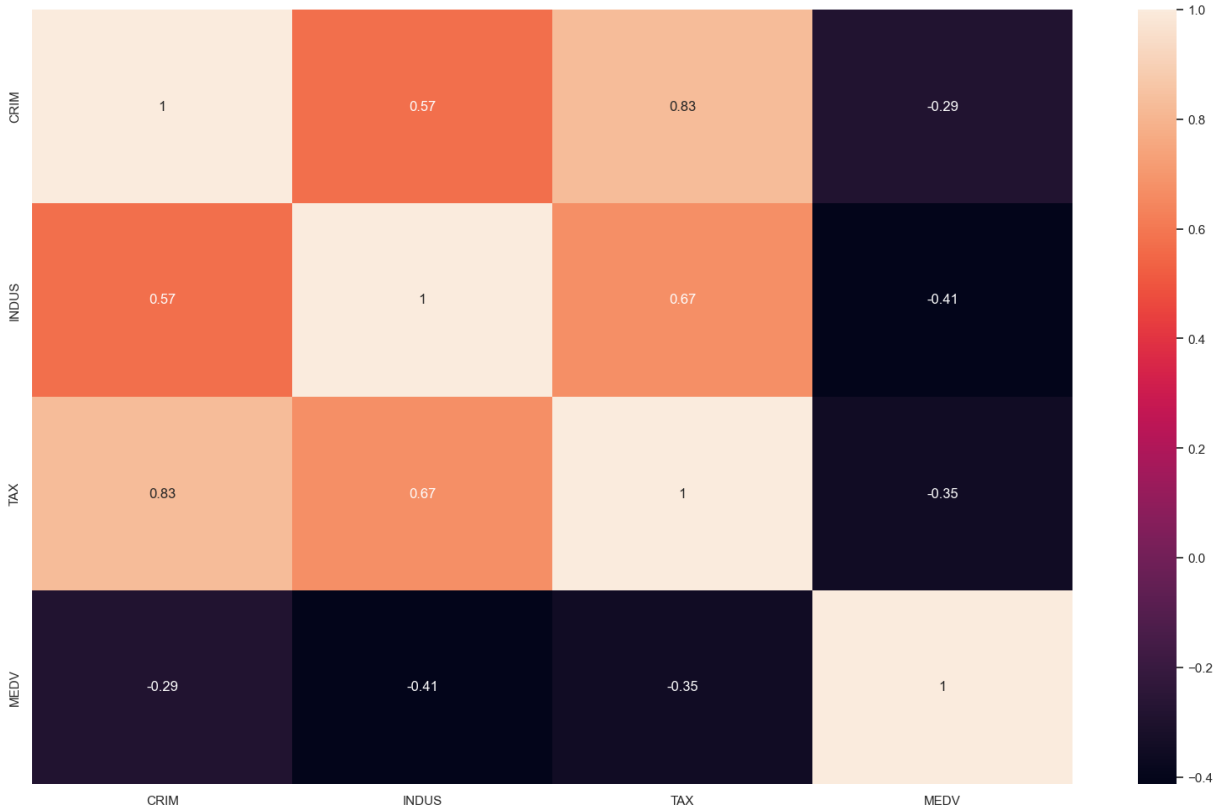
In [ ]:
```
# Let's try the pairplot with only the variables in df2

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




The sns.pairplot() function will create a pairplot based on the numerical variables in the given dataset. Inside the parentheses is df2 followed by height equalling 5.5. Next, teh plt.show() function will output the pairplot with a height value of 5.5.

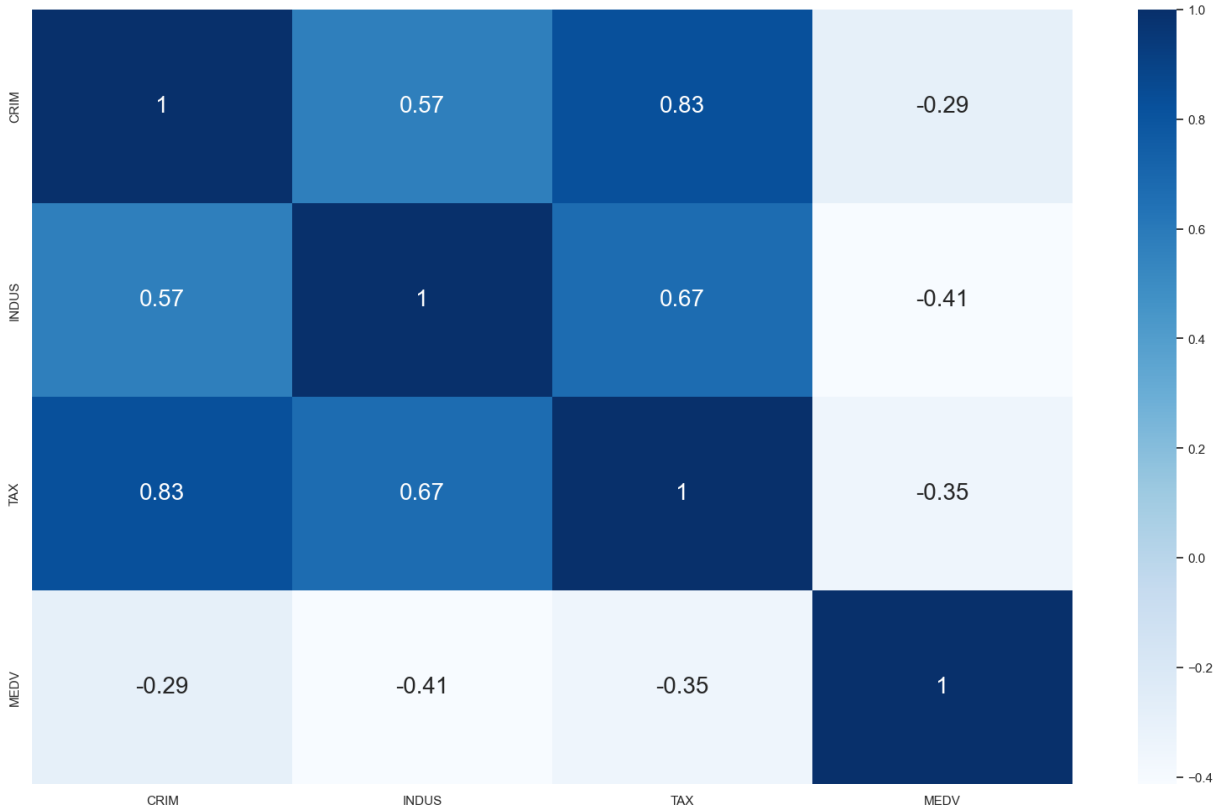
In [ ]:
```
# Now we will make a heatmap with only the variables in df2 subset.  Again, it is v

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

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




The plt.figure () function containing figsize equalling (20,12), will set the size of the plot to a have a 20x12 value. Next, The plt.figure () function containing figsize equalling (20,12), will set the size of the plot to a have a 12x20 value. Then the sns.heatmap () function will create a heatmap, inside the parentheses is df2.corr() followed by the annot parameter equalling true. This will create a heatmap for the correlations of df where2 the values are labeled. Lastly, plt.show() will output the heatmap based on the parameters set in the plt.figure() and sns.heatmap().

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

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




The plt.figure () function containing figsize equalling (20,12), will set the size of the plot to a have a 20x12 value. Next the sns.heatmap () function will create a heatmap, inside the parentheses is df.corr(), followed by the cmap parameter equal to blue, followed by the annot parameter equalling true, then annot_kws equalling a dictionary where the key is fontsize and the value is 20. This will create a heatmap for the correlations of df2 where the values are labeled, the color map is blue and the fontsize for the labels have a value of 20. Lastly, plt.show() will output the heatmap based on the parameters set in the plt.figure() and sns.heatmap().

# Separate the Dataset into Input & Output NumPy Arrays

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

This will import the train_test_split function from sci-kit learn library.

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

array= df2.values

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

The df2.values is assigned to a variable called array. This will store the dataframe values of df2 into an array called array. Array followed by a square bracket containing a colon, then a 0, then a colon, and the a 3. This will slice array to include all the values from column index position 0 to 3 but not including position 3. This is then assigned to X. Then, array followed by a square bracket containing a colon and a 3. This will slice array to only include the values of column 3. This is assigned to Y.

# Spilt into Input/Output Array into Training/Testing Datasets

In [ ]:
```
# Split the dataset --> training sub-dataset:  67%, and test sub-dataset:  33%

test_size = 0.33

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

seed = 7

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

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

0.33 is assigned to test_size, 7 is assigned to seed. Next, the train_test_split() function will split the data into two randomly sampled subsets. Inside the function is X followed by Y, followed by the test_size parameter equal to 0.2 and then the random_state parameter equalling seed. This is then assigned to the variables x_train, x_test, Y_train and Y_test. This will create a train test split based on X and Y where the test contains 20% of the observations and train contains 80% of the observations.

# Build and Train the Model

In [ ]:
```
# 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
# intercept = mean (average) value of Y
```

```
# if the value is less than 0.05: there is a strong relationship between the variab

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

```
Intercept: 31.393427670412926
Coefficients: [ 0.09859287 -0.42388844 -0.00931847]
```

The LinearRegression() function is assigned to model. The model.fit function will calculate and train the arrays inside the parentheses. In this case it is X_train and Y_train. Next there are 2 print function. Inside the first print function is "intercept:" which is in between the quotations, this followed by the model.intercept function. This will output the text between the quotations and then the intercept of the linear regression. Inside the second print function is "Coefficients:" which is in between the quotations, this followed by the model.coef_ function. This will output the text between the quotations and then the coefficients for the predictor variables.

In [ ]:
```
# If we want to print out the list of the coefficients with their correspondent var
# 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', -0.4238884417716138)

('TAX', -0.009318474474503402)

('CRIM', 0.09859287239144143)
```

The predictor variables are presented in a list in string format and then assigned to names_2. Then using the zip() function names_2 and model.coef_ will aggregate together which is assigned to coeffs_zip. Then coeffs_zip is converted into a set using set() function and then assigned to the variable, coeffs. Lastly, a for loop that iterates through coeffs using coef after each element in coeff is printed, the python cursor skips to the next line.

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

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

## Calculate R-Squared

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

```
R-squared:  0.1547331337359038
```

model.score() where X_test and Y_test is inside the parentheses will calculate the r-squared value for the given arrays. This is then assigned to R_squared. Then using the print function, is the text, "R-squared: " inside the quotations followed by R-squared will be outputted.

## Notes: The higher the R-squared, the better (0 – 100%). Depending on the model, the best models score above 83%. The R-squared value tells us how well the independent variables predict the dependent variable. This is very low. Think about how you could increase the R-squared. What variables would you use? This will be important for the final.

## Prediction

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

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

The model.predict() function where a nested list containing the values, 12,10, 450 inside the parentheses. This will output the predicted value based on the values given in the nested list.

## 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 [ ]:
```
# 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 model

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 validation

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

```
Average of all results from the K-fold Cross Validation, using negative mean squared
error: -64.35862748210984
```

10 assigned to num_folds, 7 assigned to seed. Inside the Kfolds() function is the parameter n_splits equalling num_folds, random_state equalling seed and shuffle equal to true. This will split the data into 10 folds, this is assigned to the variable kfold. The string, 'neg_mean_squared_error' is assigned to the variable scoring. Next, inside the cross_val_score() function is the variable model, followed by X, followed by Y, cv equalling to kfold and the scoring parameter assigned to the variable scoring. This assigned to results. Lastly, inside the print function, the text between the quotations, "Average of all results from the K-fold Cross Validation, using negative mean square error:" followed by results.mean(), which will find the average of results, wil be outputted.

# 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 [ ]:
```
# 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
# 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 the model/

scoring = 'explained_variance'

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

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 validation

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

```
Average of all results from the K-fold Cross Validation, using exlpained variance:
0.19023822025958675
```

10 assigned to num_folds, 7 assigned to seed. Inside the Kfolds() function is the parameter n_splits equalling num_folds, random_state equalling seed and shuffle equal to true. This will split the data into 10 folds, this is assigned to the variable kfold. The string, 'explained_variance' is assigned to the variable scoring. Next, inside the cross_val_score() function is the variable model, followed by X, followed by Y, cv equalling to kfold and the scoring parameter assigned to the variable scoring. This assigned to results. Lastly, inside the print function, the text between the quotations, "Average of all results from the K-fold Cross Validation, using explained variance:" followed by results.mean(), which will find the average of results, wil be outputted.

# Discovery and Learning with Big Data/Machine Learning

# Drew Murray

# Machine Learning Supervised Logistic Regression

## 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

## Import Libraries

In [ ]:
```
# Import Python Libraries: NumPy and Pandas
```

```
import pandas as pd
import numpy as np
```

Pandas library is imported to convert csv file into a dataframe. NumPy is imported to perform mathematical array operations.

In [ ]:
```
# Import Libraries & modules for data visualization

from pandas.plotting import scatter_matrix
import matplotlib.pyplot as plt
import seaborn as sns
```

scatter_matrix from the pandas.plotting is used to create a scatter matrix, matplotlib,pyplot is imported to create and display plots. seaborn is imported as a tool for data visualizations.

In [ ]:
```
# Import scikit-Learn module for the algorithm/modeL: Logistic Regression
from sklearn.linear_model import LogisticRegression
```

The LogisticRegression funciton is imported from scikit learn library to execute logistic regression.

In [ ]:
```
# Import scikit-Learn module to split the dataset into train/ test sub-datasets
from sklearn.model_selection import train_test_split
```

The train_test_split is imported from the scikit learn library to subset the data into a train and test subset.

In [ ]:
```
# Import scikit-Learn module for K-fold cross-validation - algorithm/modeL evaluati

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

Both Kfold and cross_val_score functions were imported from the scikit learn library to execute cross validation.

In [ ]:
```
# Import scikit-Learn module classification report to later use for information abo
#try to classify/lable each record

from sklearn.metrics import classification_report
```

The classification_report function is imported from the scikit learn library to classify the given observations.

# Load Data

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

filename = "C:/Users/dgmur/OneDrive/Desktop/ADTA 5340 Discovery and Learning with B
```

```
# Load the data into a Pandas DataFrame

df = pd.read_csv(filename)
```

The file path to retrieve the iris dataset csv is assigned to filename. The pd.read_csv is used to convert filename into a dataframe, this is assigned to the variable df.

# Look at the data frame

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

Out[ ]:

| | 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 df.head() function will output the first 5 rows of df.

# Preprocess the Dataset

Clean the data: Find and Mark Missing Values

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

df[[ 'SepalLengthCm' , 'SepalWidthCm' , 'PetalLengthCm' ,'PetalWidthCm' ]] \
= df[['SepalLengthCm' , 'SepalWidthCm' ,'PetalLengthCm' , 'PetalWidthCm' ]].replace

# 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
```

df followed by a nested list containing some of the variables of df in string form, followed by the replace function where 0, and np.NaN is inside. This is assigned to df which followed by a nested list containing some of the variables of df in string form which is then followed by a . This will replace NaN values with a zero. Lastly, using the print function. df.isnull().sum() is used to output the number of NaN values for each variable of df.

# Performing the Exploratory Data Analysis (EDA)

In [ ]:
```
# 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)
```

Inside the print function is text between quotations, "Shape of the dataset (rows, columns):", which followed by the df.shape function which displays the total rows and columns of df, will be outputted.

In [ ]:
```
#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
```

Inside the print function is the df.types function, this will output the data types of each of the variables of df.

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

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
```

Inside the print function is the df.describe() function. This will output descriptive statistics for each of the variables for df, this includes the total count, mean, std, minimum, 25% quartile, median, 75% quartile, and maximum.

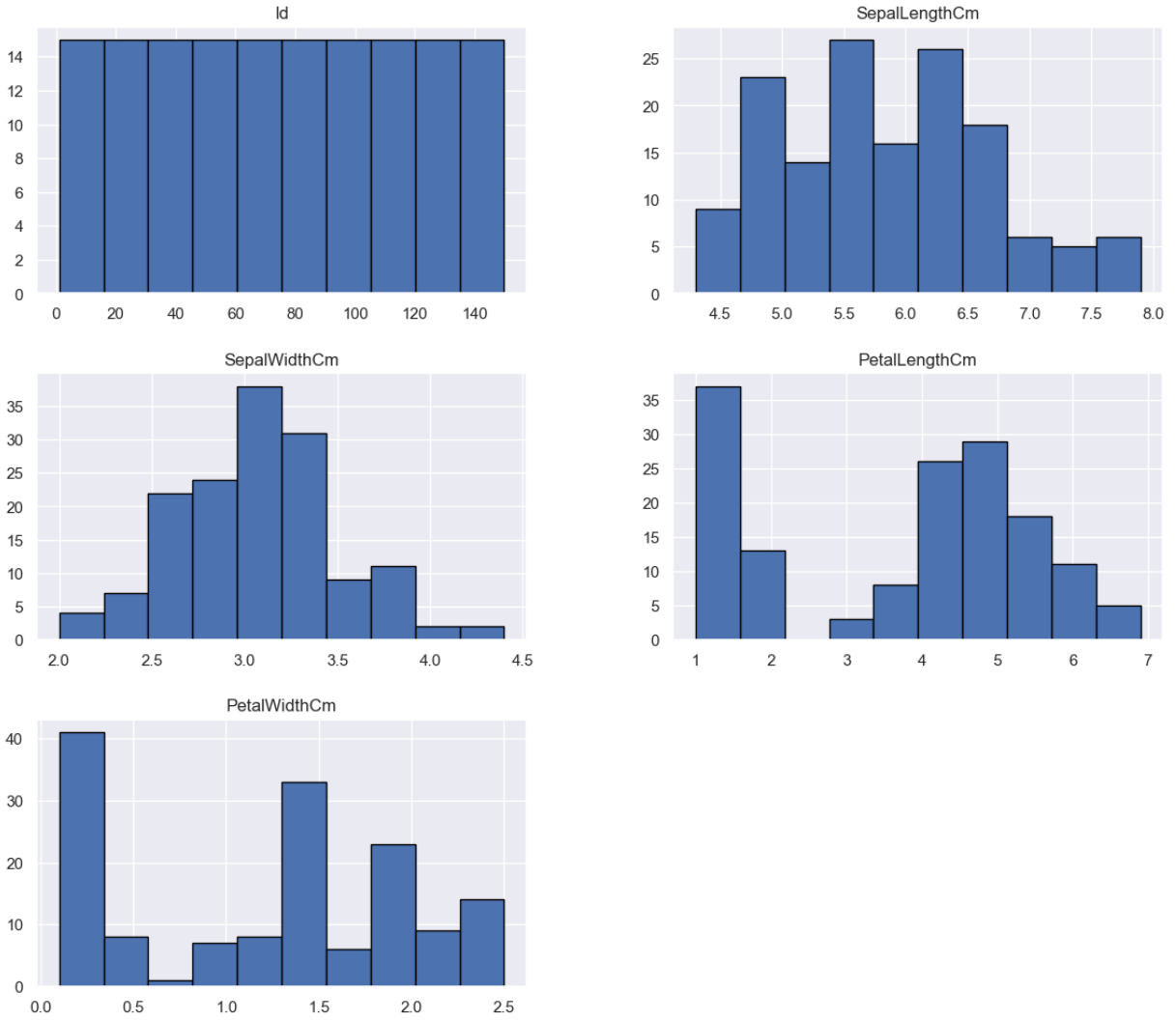
In [ ]:
```
#class distribution i.e. how many records are in each class

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

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

Inside the print function is df,groupby where the species variable is inside the parentheses as a string, followed by the size() function. This will output the number of observations for each of the class for species.

# Creating Histogram

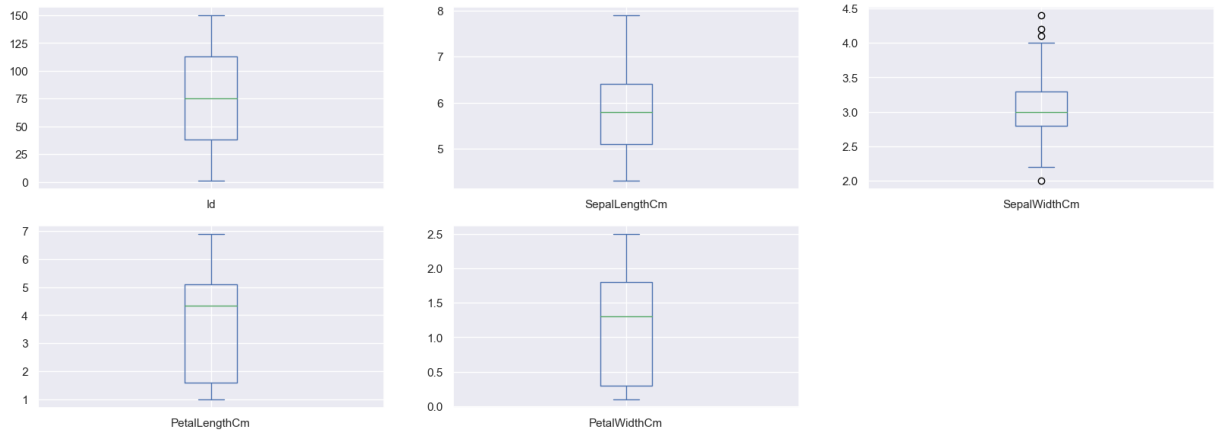
In [ ]:
```
# Plot histogram for each variable. I encourage you to work with the histogram. Rem

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




The df.hist() containing edgecolor equalling to black and figsize equalling to (14,12) inside the parentheses. This will create a histogram for each of the variables of df where the bars of the histograms are outlined in black and each of the histograms have a 14x12 value. Lastly, the plt.show() will output the histograms.

# Creating a Box plot

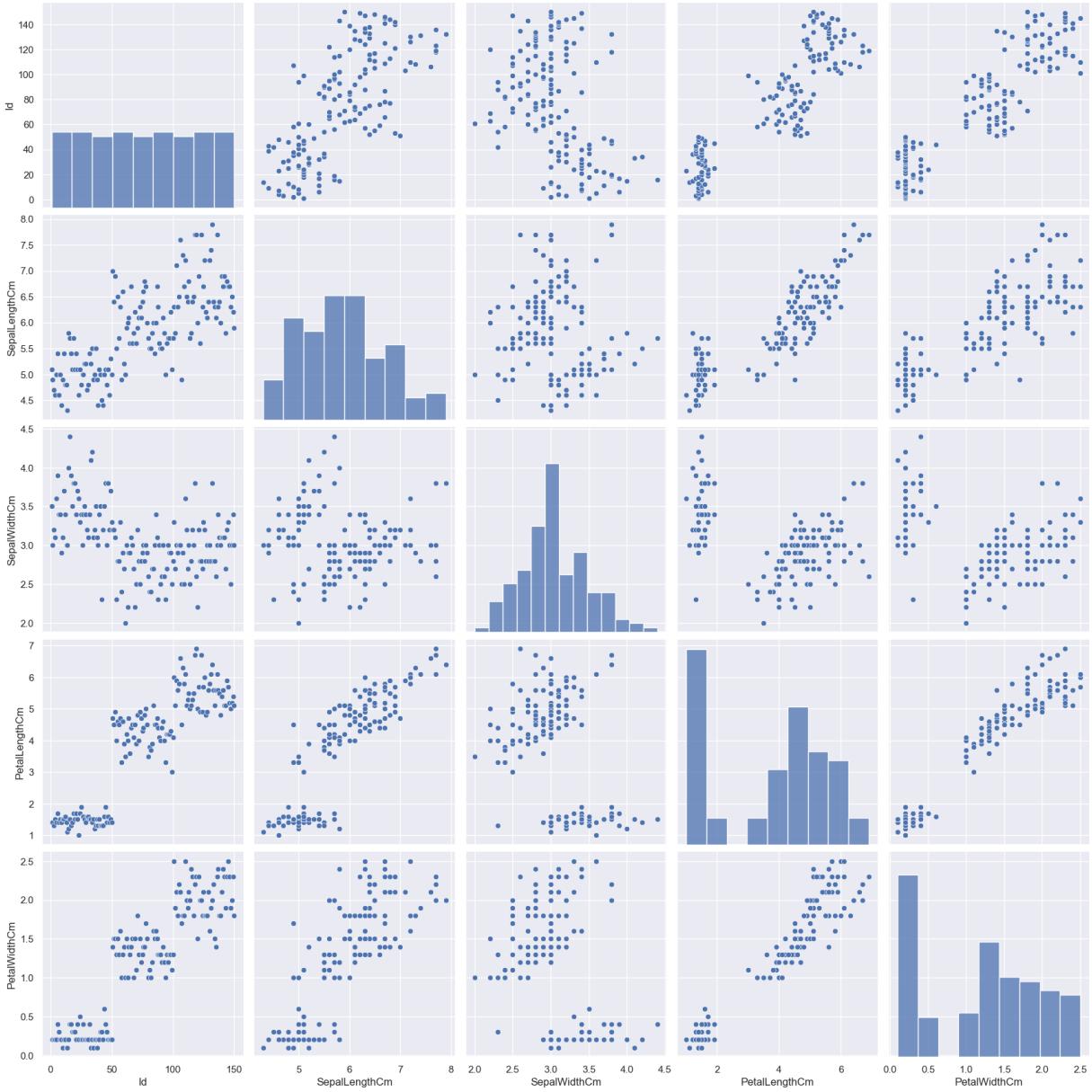
In [ ]:
```
# Boxplots

df.plot(kind="box", subplots=True, layout=(5,3), sharex=False, figsize=(20,18))
plt.show()
```




Inside the df.plot() is the kind parameter equalling box, followed by subplots equal to true, layout equal to (5,3), then sharex is equal to false and the figsize is equal to (20,18). This will create a boxplot for each of the variables in df in a 5 by 3 layout where each boxplot has a figure size value of 20x18. Then plt.show() will output the histograms.

# Create a Pair Plot

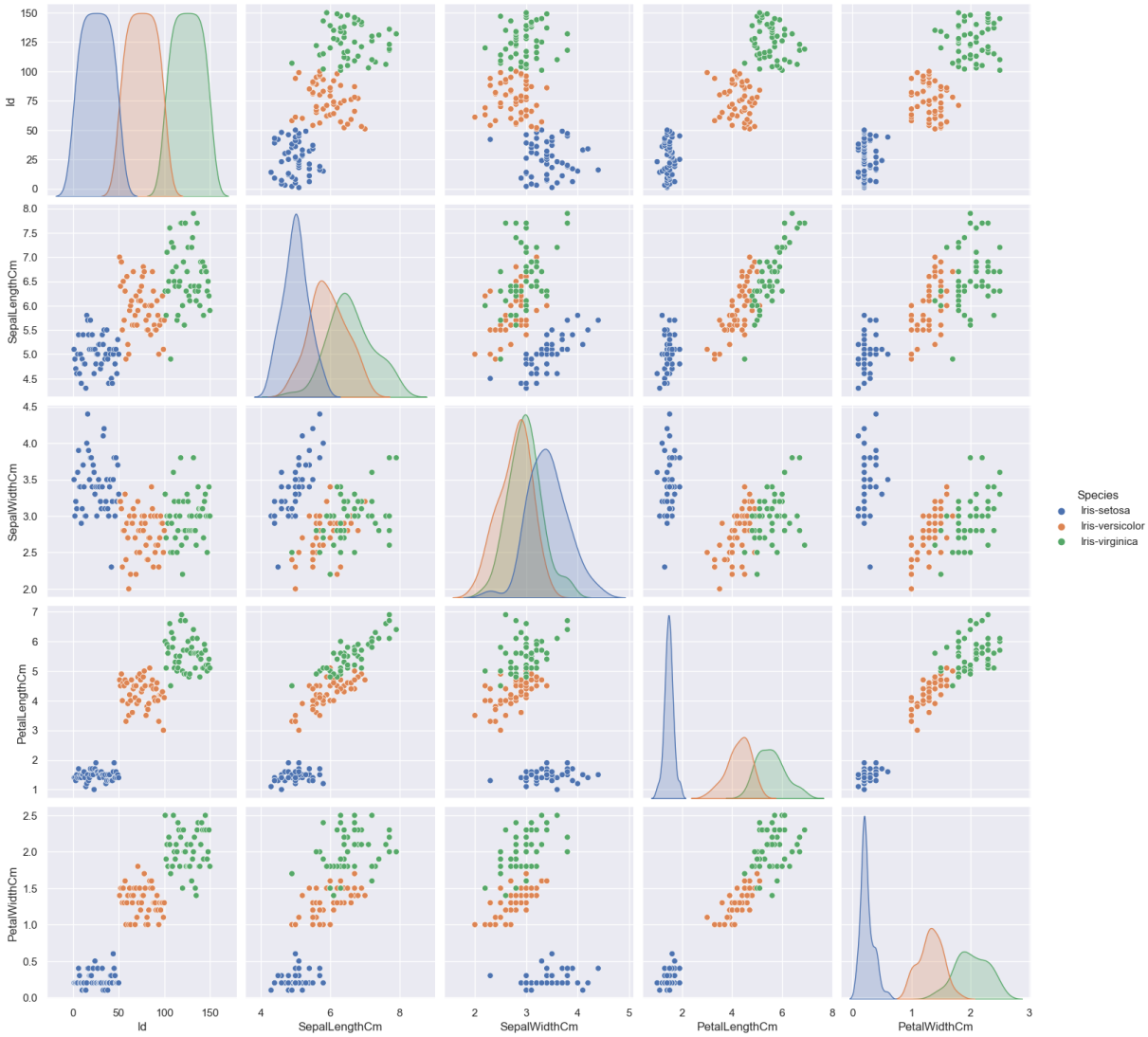
In [ ]:
```
# Please click on the above URL to learn more about Pair Plots
# I know this is a lot of information but I wanted you to see what is possible with

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




The sns.pairplot() function will create a pairplot based on the numerical variables in the given dataset. Inside the parentheses is df followed by height equalling 3.5. Next, the plt.show() function will output the pairplot with a height value of 3.5.

# Creating a Pair Plot with Color

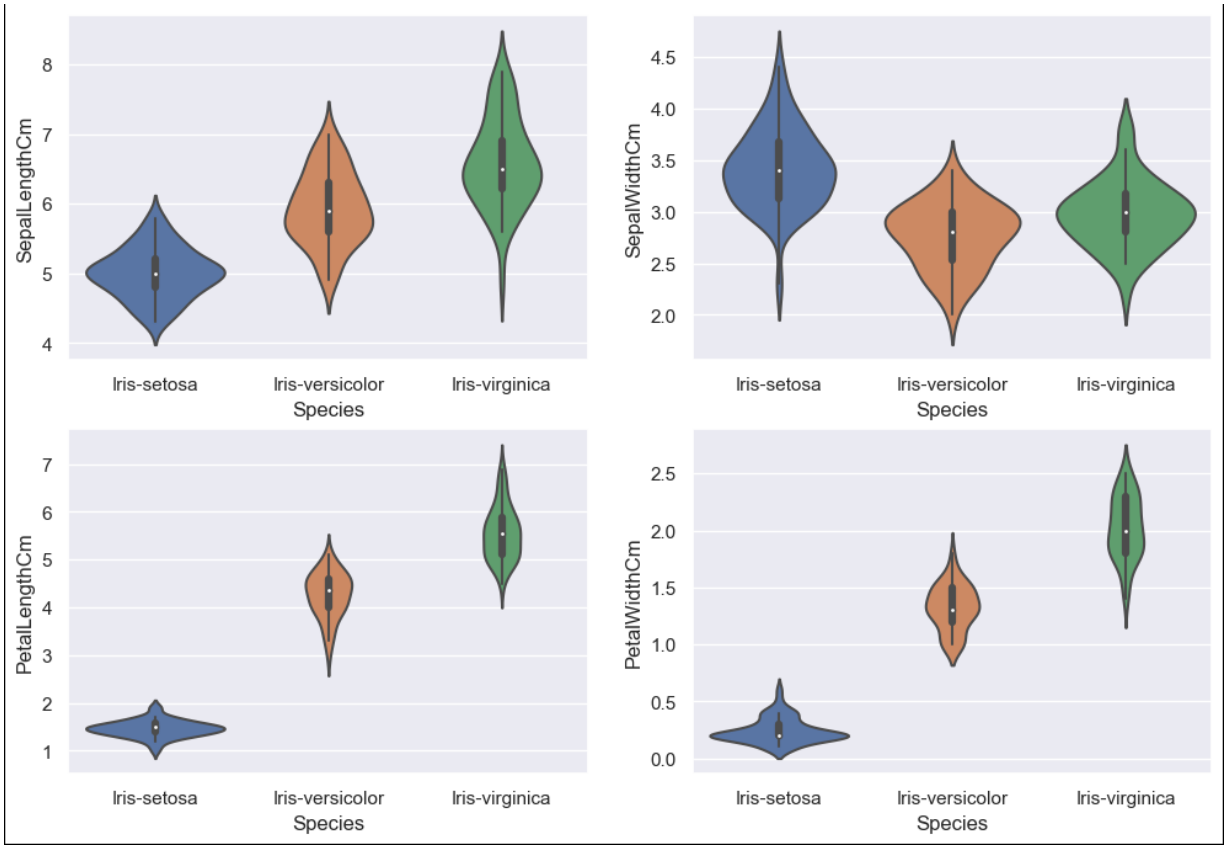
In [ ]:
```
# Let's try that again using color.  Notice: assigning a hue variable adds a semant

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




The sns.pairplot() function will create a pairplot based on the numerical variables in the given dataset. Inside the parentheses is df followed by hue equal to species, height equal to 3 and aspect equal to 1. This will create a pairplot where the color is based on the species variable, and the diagonal changes from a histogram to a kde. Next, the plt.show() function will output the pairplot.

# Creating a Violin Plot

In [ ]:
```
#  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);
```




- Inside plt.figure is the parameter edgecolor equalling to black, linewidth equallin 1,2 and figsize equal to (12,8). This will make the given plots have a black outline with a linewidth value of 1.2 and the figure size having a 12x8 format. Next inside the plt.subplot(2,2,1) will assign the next plot to the 2,2,1 position. Then the sns.violinplot() containing x equalling species, y equalling SepalLengthCm, which is then followed by data equal to df. This will create a violinplot where the y axis is sepalLengthcm variable and the x axis is the species variable. This plot is assigned to the 2,2,1 position. Next inside the plt.subplot(2,2,2) will assign the next plot to the 2,2,2 position. Then the sns.violinplot() containing x equalling species, y equalling SepalWidthCm, which is then followed by data equal to df. This will create a violinplot where the y axis is sepalWidthcm variable and the x axis is the species variable. This plot is assigned to the 2,2,2 position. Next inside the plt.subplot(2,2,3) will assign the next plot to the 2,2,3 position. Then the sns.violinplot() containing x equalling species, y equalling PetalLengthCm, which is then followed by data equal to df. This will create a violinplot where the y axis is PetalLengthcm variable and the x axis is the species variable. This plot is assigned to the 2,2,3 position. Next inside the plt.subplot(2,2,4) will assign the next plot to the 2,2,4 position. Then the sns.violinplot() containing x equalling species, y equalling PetalWidthCm, which is then followed by data equal to df. This will create a violinplot where the y axis is PetalWidthcm variable and the x axis is the species variable. This plot is assigned to the 2,2,4 position.

# Separate the Dataset into Input & Output NumPy Arrays

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

df.values is stored as an array and assigned to the variable array. The variable, array followed by a pair of square brackets containing a colon, followed by 1, colon and 5. This will slice array to include all the observations for the position index of the columns 1 to 5 but not including 5. This is assigned to X. Then, the variable array followed by a pair of square brackets containing a colon followed by a 5. This will slice array to only all the observations for the index position of column 5. This is assigned to Y.

# Spilt into Input/Output Array into Training/Testing Datasets

In [ ]:
```
# 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 randomly

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_size,
random_state=seed)
```

0.33 is assigned to test_size, 7 is assigned to seed. Next, the train_test_split() function will split the data into two randomly sampled subsets. Inside the function is X followed by Y, followed by the test_size parameter equal to the test_size variable, and then the random_state parameter equalling seed. This is then assigned to the variables x_train, x_test, Y_train and Y_test. This will create a train test split based on X and Y where the test contains 33% of the observations and train contains 67% of the observations.

# Build and Train the Model

In [ ]:
```
#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 LogisticRegression() function containing the parameter random_state equal to seed and max_iter equal to 1000. This will call the LogisticRegression function where the model is randomized and iterates 1000 times, This is then assigned to the variable model. Next the model.fit function containing X-train followed by Y_train in the parentheses. This will train the model using the X and Y from the train subset. Then, the model.predict function containing X_train in the parentheses. This will use the elements of X in the test subset to classify its values, this is then assigned ot predicted. The classification_report() containing Y_test followed by predicted, which will test the accuracy between the predicted corresponding values of X_test and Y_test., this is assigned to report. Lastly inside the print function the text between the quotations, "Classification Report: " followed by a "\n" followed by another "\n" then followed by report, will be outputted where there two lines are skipped between the outputs of "Classification Report: " and report.

# Score the Accuracy of the Model

In [ ]:
```
# 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 model.score() function containing X_test and Y_test, will score the average accuracy for test subset of X and Y. Next, inside the print function is another set parentheses containing "Accuracy: %.3%%f" followed by the %, followed by another set of parentheses containing results times 100. This will output "Accuracy:" and results*100 where the output of results is converted to a float that is 3 decimal places long.

# Classify/Prediction

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

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

Inside the model.predict function is a nested list of values. The list of values is aligned with predictor variables and is used to predict species based on those values.

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

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

Inside the model.predict function is a nested list of values. The list of values is aligned with predictor variables and is used to predict species based on those values.

# We have now trained the model and used that trained model to predict the type of flower we have with the listed values for each variable.

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

In [ ]:
```
# Evaluate the algorithm and specify the number of times of repeated splitting, in

n_splits=10

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

seed=7

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

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

scoring="accuracy"
```

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

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

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

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

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

10 assigned to num_folds, 7 assigned to seed. Inside the Kfolds() function is the parameter n_splits equalling num_folds, random_state equalling seed and shuffle equal to true. This will split the data into 10 folds, this is assigned to the variable kfold. The string, 'scoring' is assigned to the variable scoring. Next, inside the cross_val_score() function is the variable model, followed by X, followed by Y, cv equalling to kfold and the scoring parameter assigned to the variable scoring. This assigned to results. Lastly, inside the print function, the text between the quotations, "Accuracy %.3f (%.3f):" followed by results.mean() and results.std(). This will output the text, "Accuracy:" and the std and mean of results in a floating point with 3 decimal places format.