Discovery and Learning with Big Data/Machine Learning

Sri Charan Bodduna

Machine Learning Supervised Linear Regression

In [40]:
```
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
```

In [41]:
```
#We are provding location of the dataset.

housingfile = '/Users/sricharanbodduna/Downloads/housing boston.csv'
```

In [42]:
```
# Loading the data into dataframe

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

Label the columns since there are no headers

In [43]:
```
#give names to the columns

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

In [44]:
```
# we are adding columns to the dataset

df.columns = col_names
```

Look at the dataframe

In [45]:
```
#  To print the first 5 rows of data

df.head()
```

Out[45]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	AA	LSTAT	MEDV
0	0.006	18.000	2.310	0	0.538	6.575	65.200	4.090	1	296	15.300	396.900	4.980	24.000
1	0.027	0.000	7.070	0	0.469	6.421	78.900	4.967	2	242	17.800	396.900	9.140	21.600
2	0.027	0.000	7.070	0	0.469	7.185	61.100	4.967	2	242	17.800	392.830	4.030	34.700
3	0.032	0.000	2.180	0	0.458	6.998	45.800	6.062	3	222	18.700	394.630	2.940	33.400
4	0.069	0.000	2.180	0	0.458	7.147	54.200	6.062	3	222	18.700	396.900	5.330	36.200

Preprocess the Dataset

Clean the data: Find and Mark Missing Values

In [46]:
```
df.isnull().sum()

# To fetch the sum of null data in the dataframe object
```

Out[46]:
```
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 [47]:
```
# To get the number of rows, and columns from dataframe

print(df.shape)
```

```
(452, 14)
```

In [48]:
```
# To print the data types of all variables

print(df.dtypes)
```

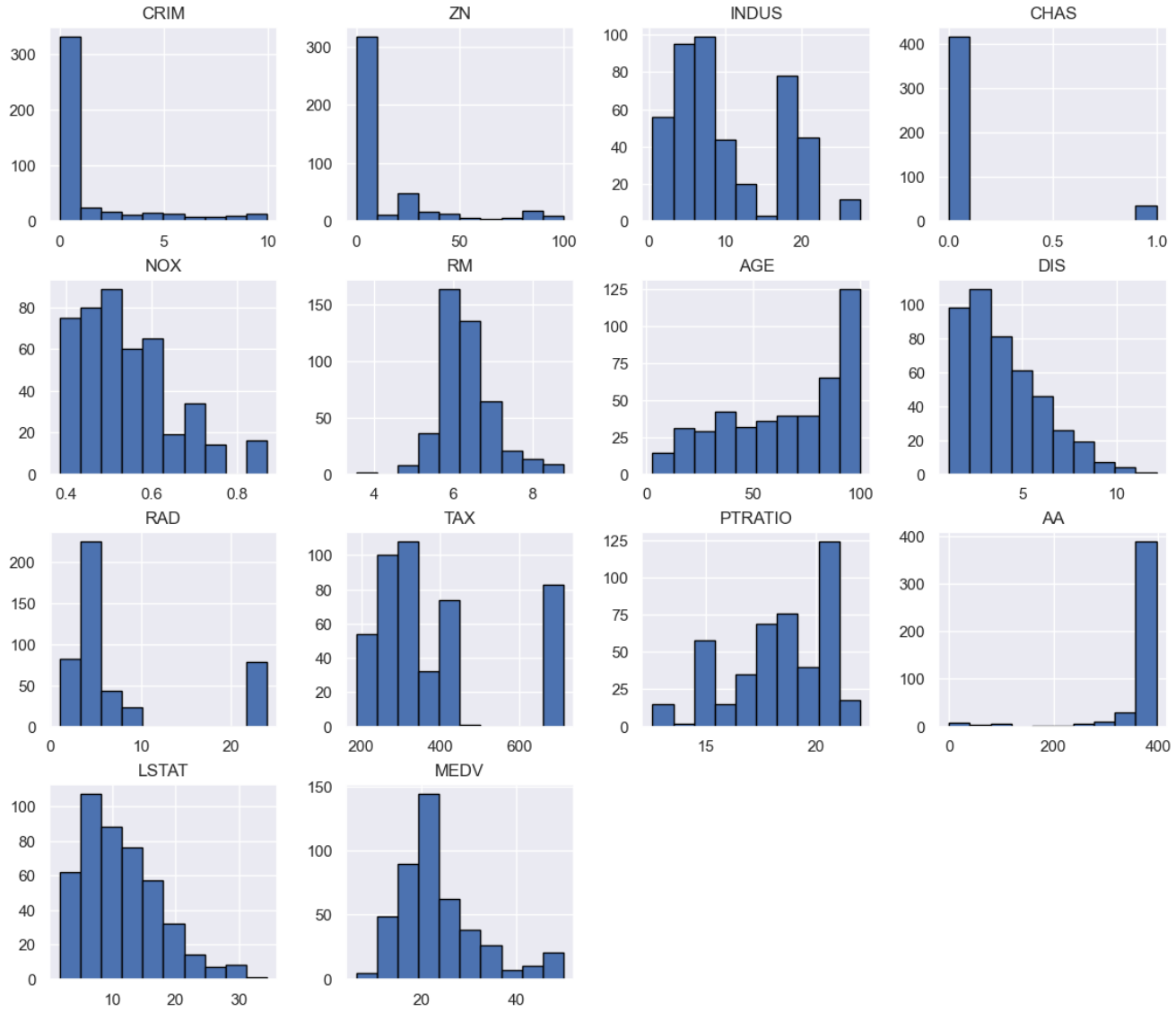
```
CRIM       float64
ZN         float64
INDUS      float64
CHAS         int64
NOX        float64
RM         float64
AGE        float64
DIS        float64
RAD          int64
TAX          int64
PTRATIO    float64
AA         float64
LSTAT      float64
MEDV       float64
dtype: object
```

In [49]:
```
# To summarize the statistics of the data

print(df.describe())
```

```
             CRIM      ZN   INDUS    CHAS     NOX      RM     AGE     DIS     RAD  \
count 452.000 452.000 452.000 452.000 452.000 452.000 452.000 452.000 452.000
mean    1.421  12.721  10.305   0.077   0.541   6.344  65.558   4.044   7.823
std     2.496  24.326   6.797   0.268   0.114   0.667  28.127   2.090   7.543
min     0.006   0.000   0.460   0.000   0.385   3.561   2.900   1.130   1.000
25%     0.070   0.000   4.930   0.000   0.447   5.927  40.950   2.355   4.000
50%     0.191   0.000   8.140   0.000   0.519   6.229  71.800   3.550   5.000
75%     1.211  20.000  18.100   0.000   0.605   6.635  91.625   5.401   7.000
max     9.967 100.000  27.740   1.000   0.871   8.780 100.000  12.127  24.000

              TAX  PTRATIO      AA   LSTAT    MEDV
count 452.000  452.000 452.000 452.000 452.000
mean  377.442   18.247 369.827  11.442  23.750
std   151.328    2.200  68.554   6.156   8.809
min   187.000   12.600   0.320   1.730   6.300
25%   276.750   16.800 377.718   6.588  18.500
50%   307.000   18.600 392.080  10.250  21.950
75%   411.000   20.200 396.157  15.105  26.600
max   711.000   22.000 396.900  34.410  50.000
```

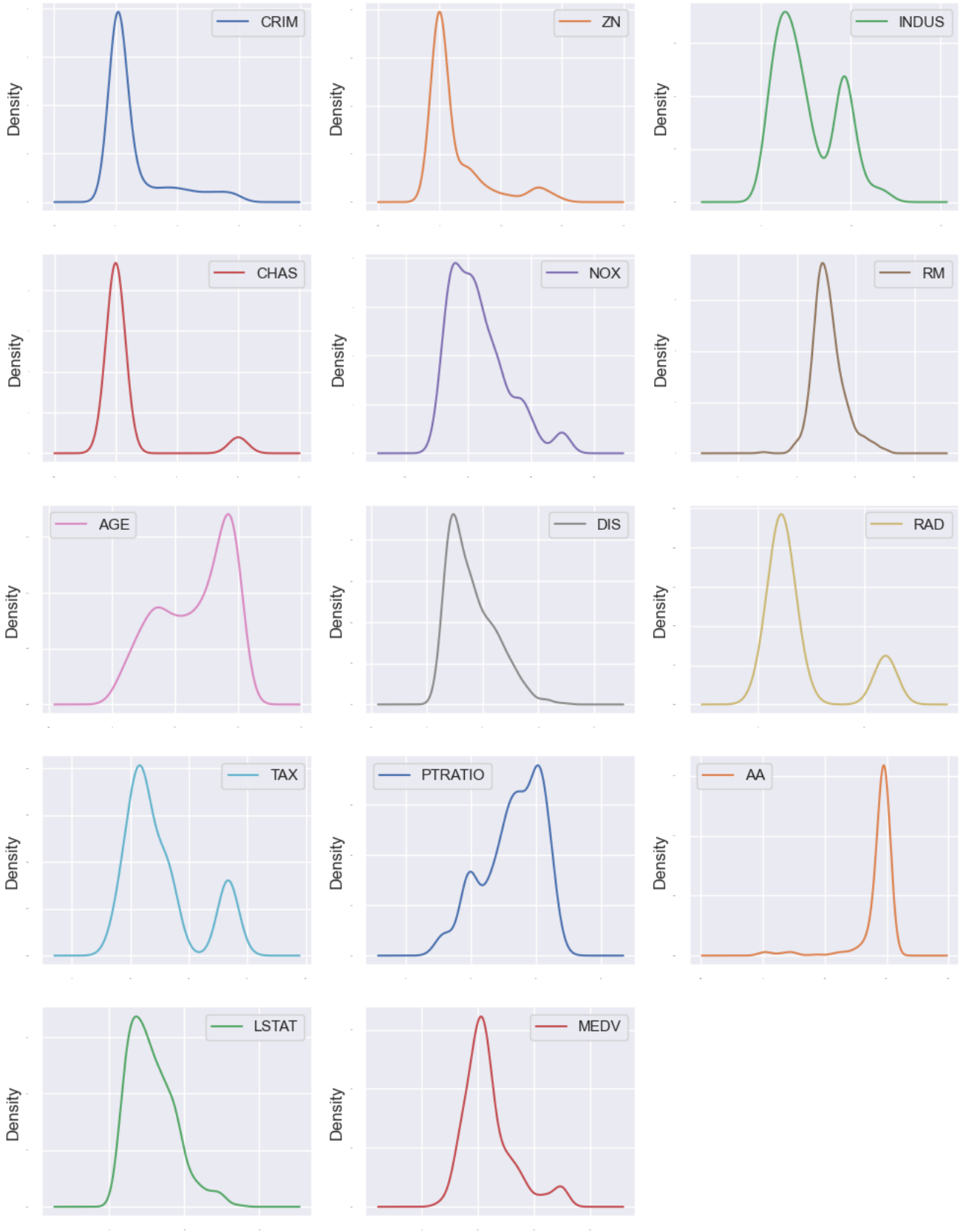
Creating a Histogram

In [50]:
```
# To plot the histogram graph for each variable.

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



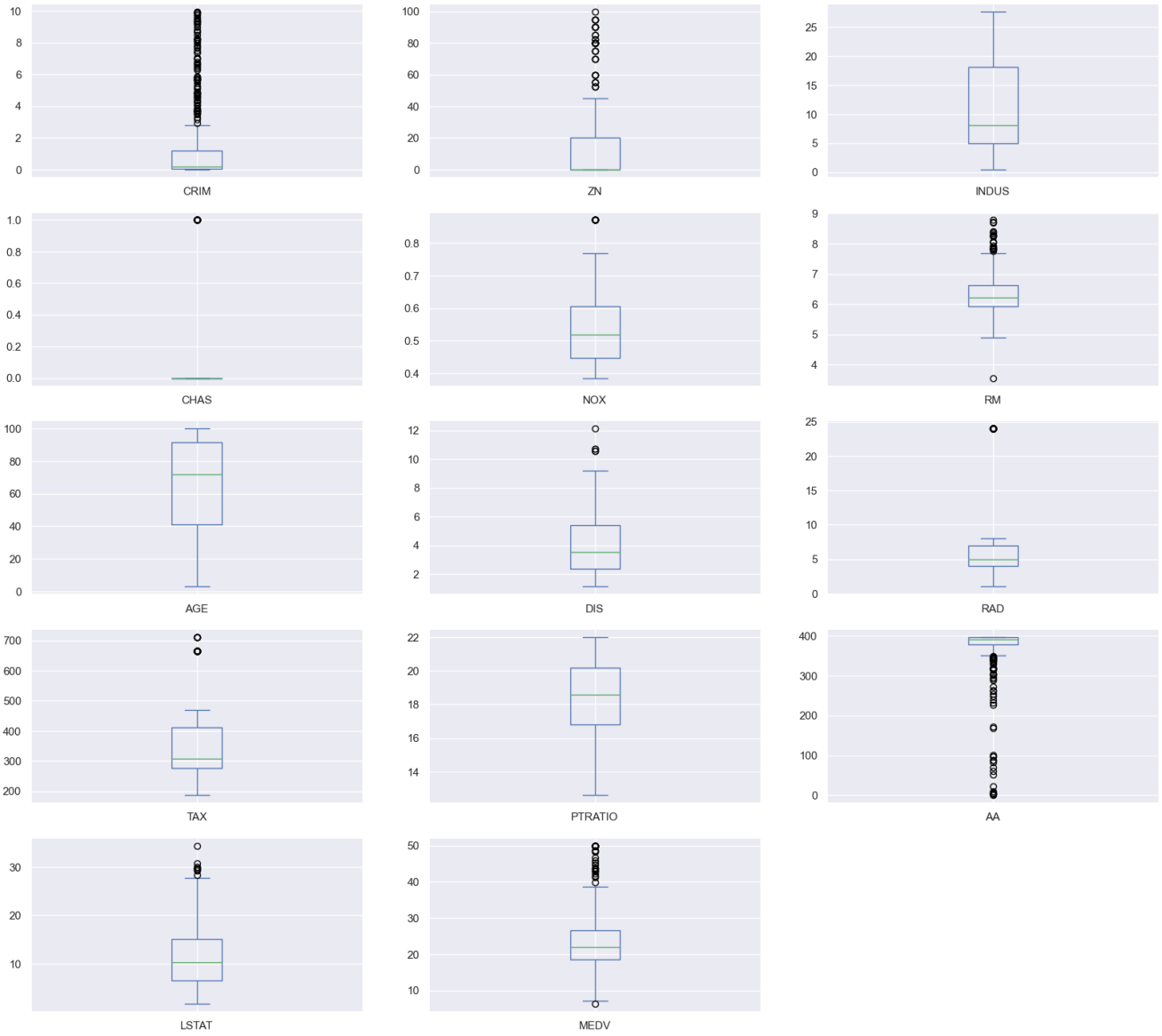
Creating a Density Plot

In [51]:
```
# Density plots
#Here we have 14 numeric variable, at least 14 plots, layout (5,3): 5 rows, each row with 3 plots
# In this code we are trying to plot the density for the variables of dataframe object using the kind=density

df.plot(kind='density', subplots=True, layout= (5,3), sharex=False, legend=True, fontsize=1, figsize= (12,16))
plt.show()
```



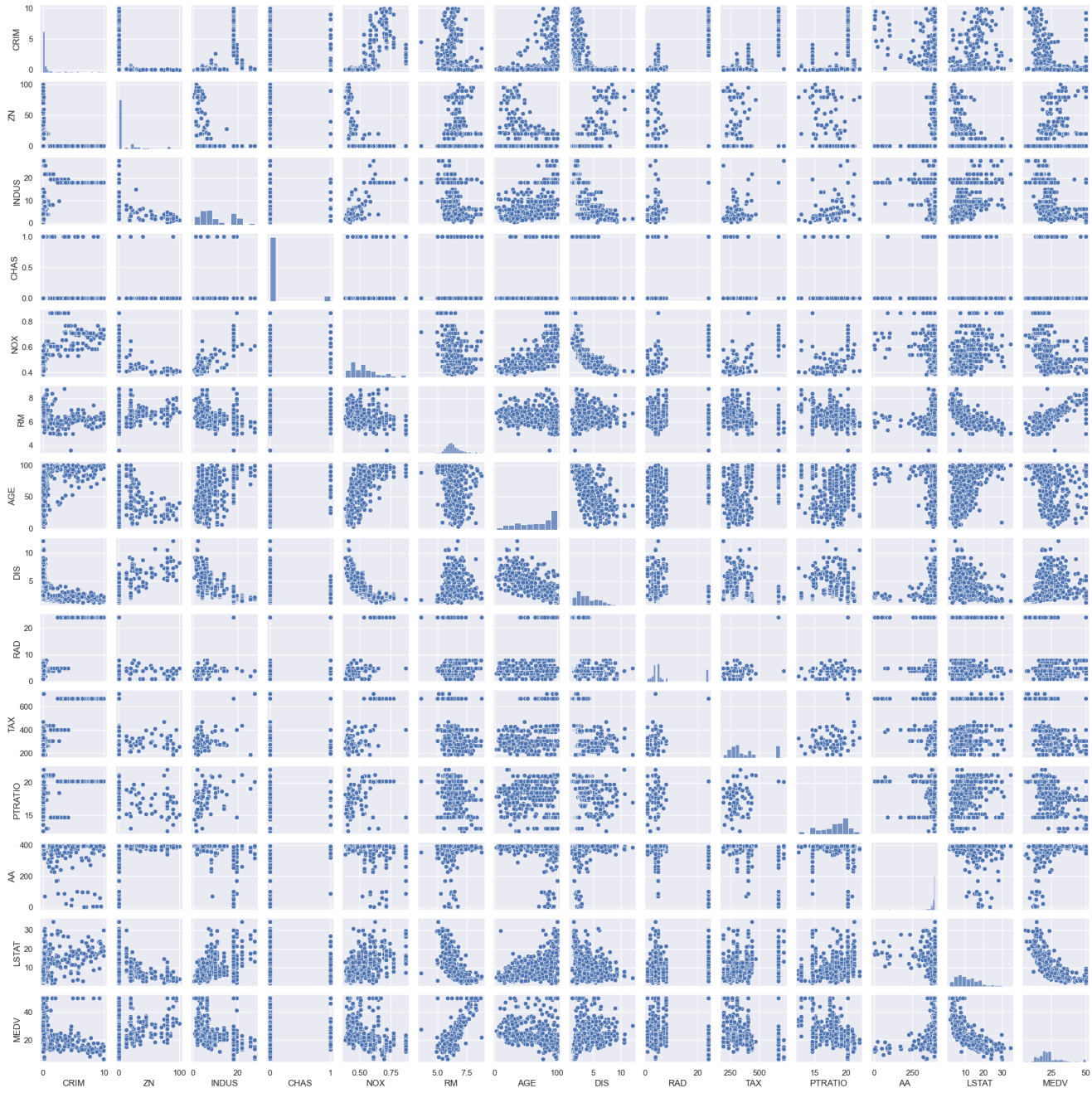
Creating a Box Plot

In [52]:
```
# In the below code we are plotted the box plot by passing the argument as the kind=box

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



Correlation Analysis and Feature Selection

In [53]:
```
#Here we are using the seaborn to use the pairplot for the dataframe

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



Correlations

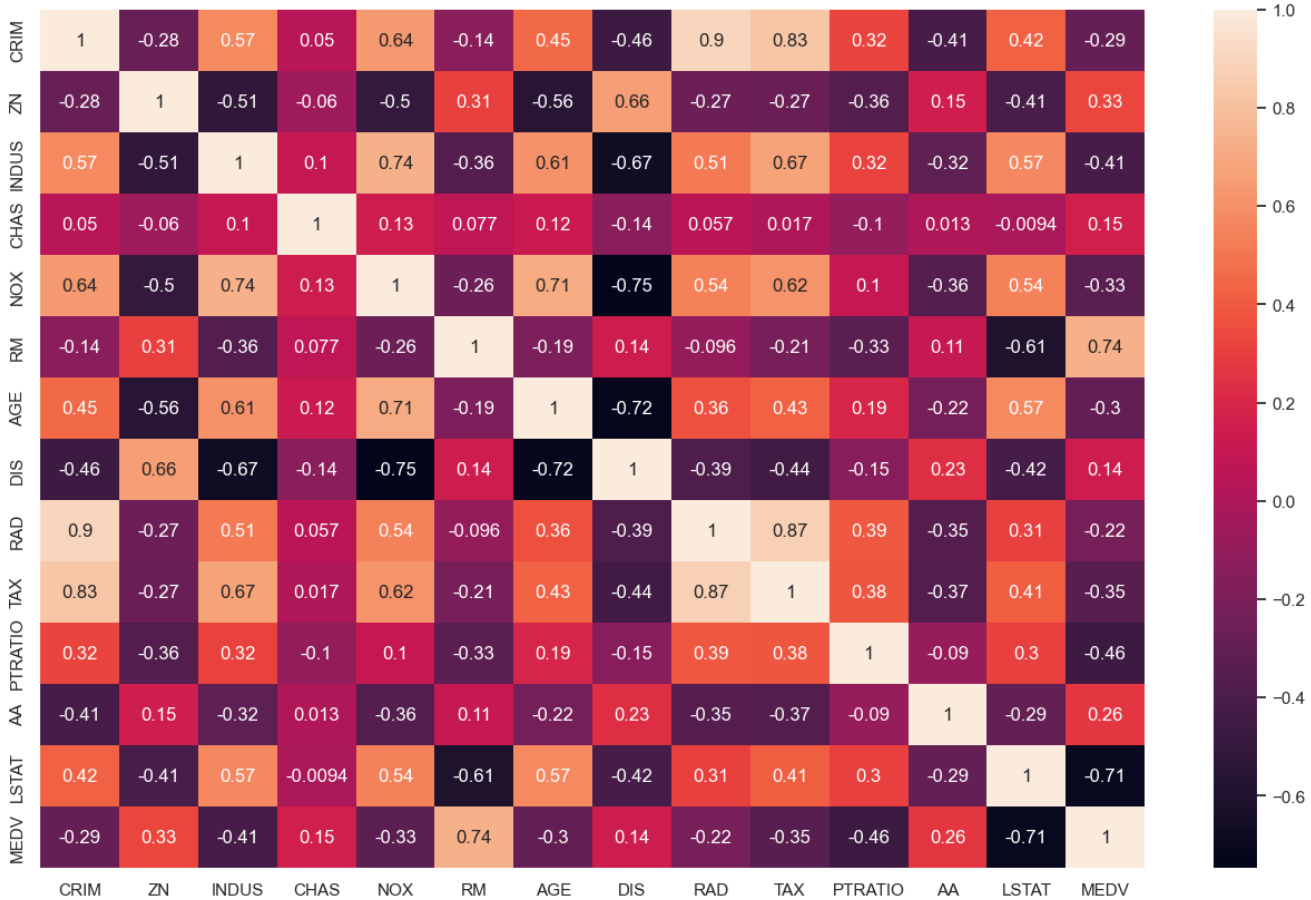
In [54]:
```
# To format the data in the dataframe to round the float upto 3 decimals

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

In [55]:
```
# Here we will get the correlations, with only 3 decimals.

df.corr()
```

Out[55]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	AA	LSTAT	MEDV
CRIM	1.000	-0.281	0.574	0.050	0.637	-0.142	0.448	-0.462	0.898	0.826	0.319	-0.413	0.425	-0.286
ZN	-0.281	1.000	-0.514	-0.060	-0.501	0.307	-0.556	0.656	-0.267	-0.269	-0.364	0.150	-0.411	0.332
INDUS	0.574	-0.514	1.000	0.103	0.739	-0.365	0.606	-0.669	0.513	0.673	0.317	-0.317	0.565	-0.412
CHAS	0.050	-0.060	0.103	1.000	0.134	0.077	0.123	-0.141	0.057	0.017	-0.100	0.013	-0.009	0.154
NOX	0.637	-0.501	0.739	0.134	1.000	-0.265	0.707	-0.746	0.542	0.615	0.103	-0.358	0.537	-0.333
RM	-0.142	0.307	-0.365	0.077	-0.265	1.000	-0.188	0.139	-0.096	-0.215	-0.334	0.108	-0.607	0.740
AGE	0.448	-0.556	0.606	0.123	0.707	-0.188	1.000	-0.720	0.359	0.427	0.193	-0.224	0.573	-0.300
DIS	-0.462	0.656	-0.669	-0.141	-0.746	0.139	-0.720	1.000	-0.388	-0.444	-0.152	0.234	-0.424	0.139
RAD	0.898	-0.267	0.513	0.057	0.542	-0.096	0.359	-0.388	1.000	0.873	0.387	-0.353	0.310	-0.218
TAX	0.826	-0.269	0.673	0.017	0.615	-0.215	0.427	-0.444	0.873	1.000	0.385	-0.367	0.411	-0.346
PTRATIO	0.319	-0.364	0.317	-0.100	0.103	-0.334	0.193	-0.152	0.387	0.385	1.000	-0.090	0.303	-0.461
AA	-0.413	0.150	-0.317	0.013	-0.358	0.108	-0.224	0.234	-0.353	-0.367	-0.090	1.000	-0.291	0.265
LSTAT	0.425	-0.411	0.565	-0.009	0.537	-0.607	0.573	-0.424	0.310	0.411	0.303	-0.291	1.000	-0.706
MEDV	-0.286	0.332	-0.412	0.154	-0.333	0.740	-0.300	0.139	-0.218	-0.346	-0.461	0.265	-0.706	1.000

In [56]:
```
# We could simply look at the correlations but a heatmap is a great way to present to the general audience.

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



In [57]:
```
# Now let's say we want to decrease the amount of variables in our heatmap.  We would use the following code.  This
#  Remember how to make a subset.  Try using different variables.

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

In [58]:
```
# Here we will look at the correlations for only the variables in df2.

df2.corr()
```

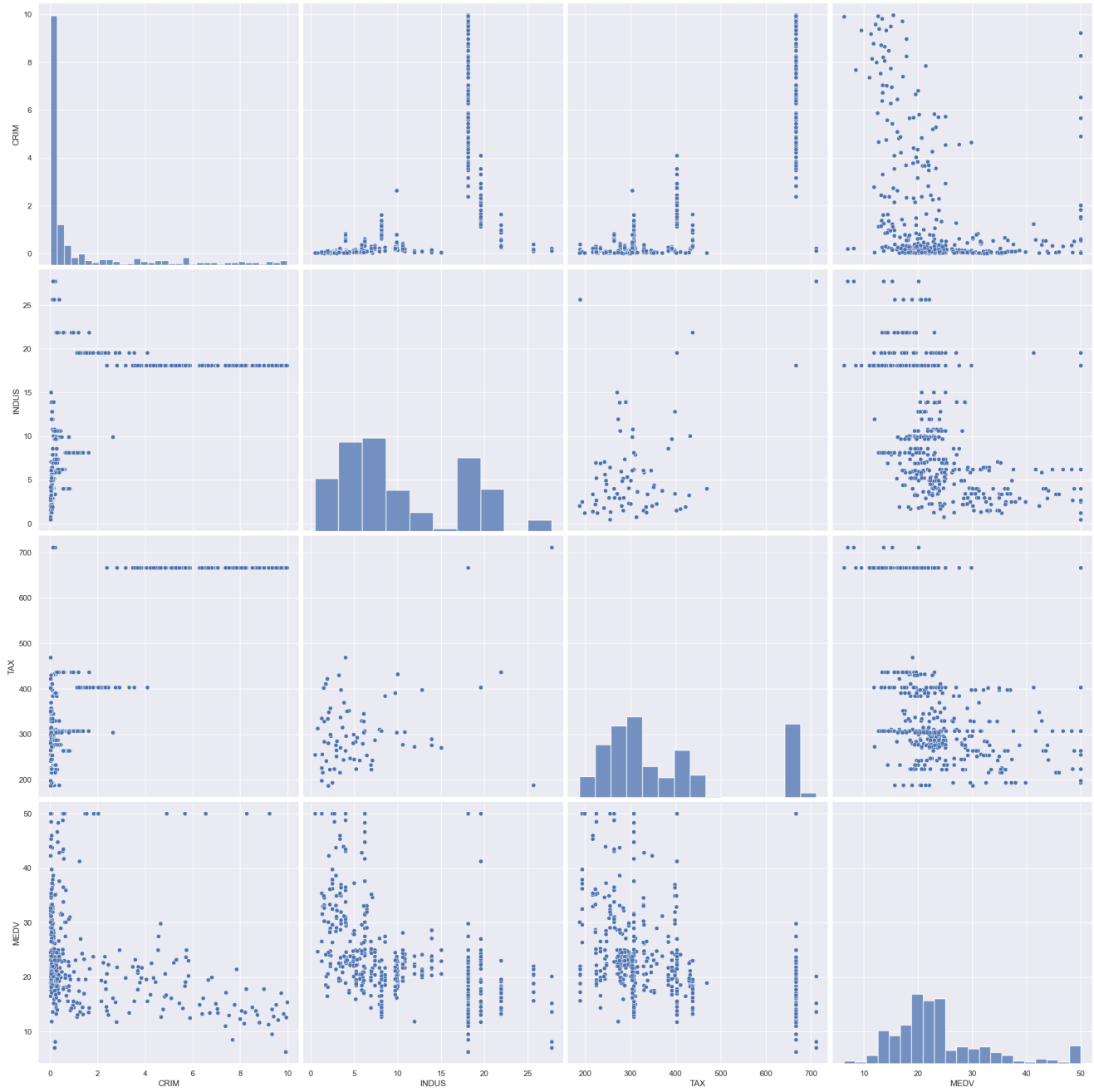
Out[58]:

	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

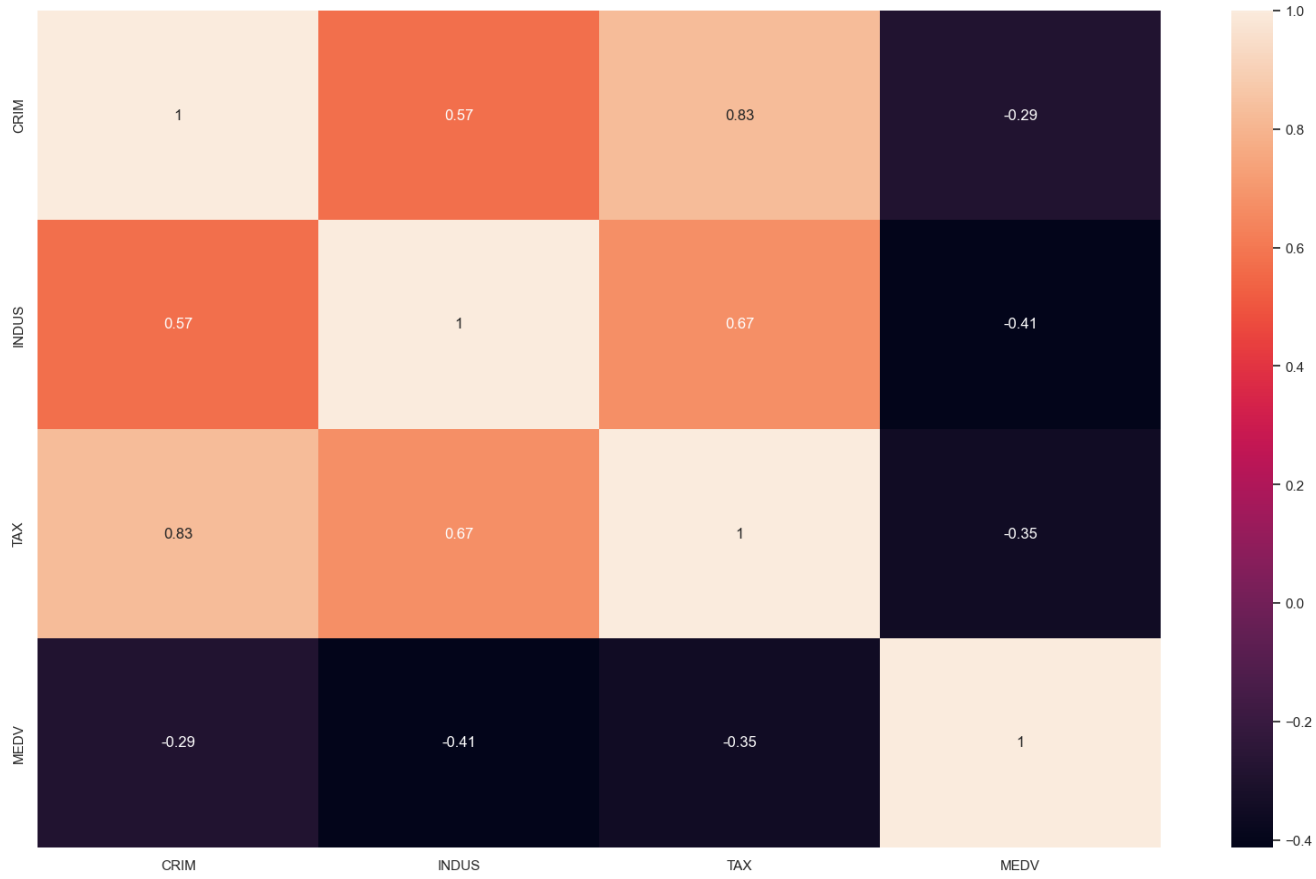
Creating a Heat Map

In [59]:
```
# Let's try the pairplot with only the variables in df2

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

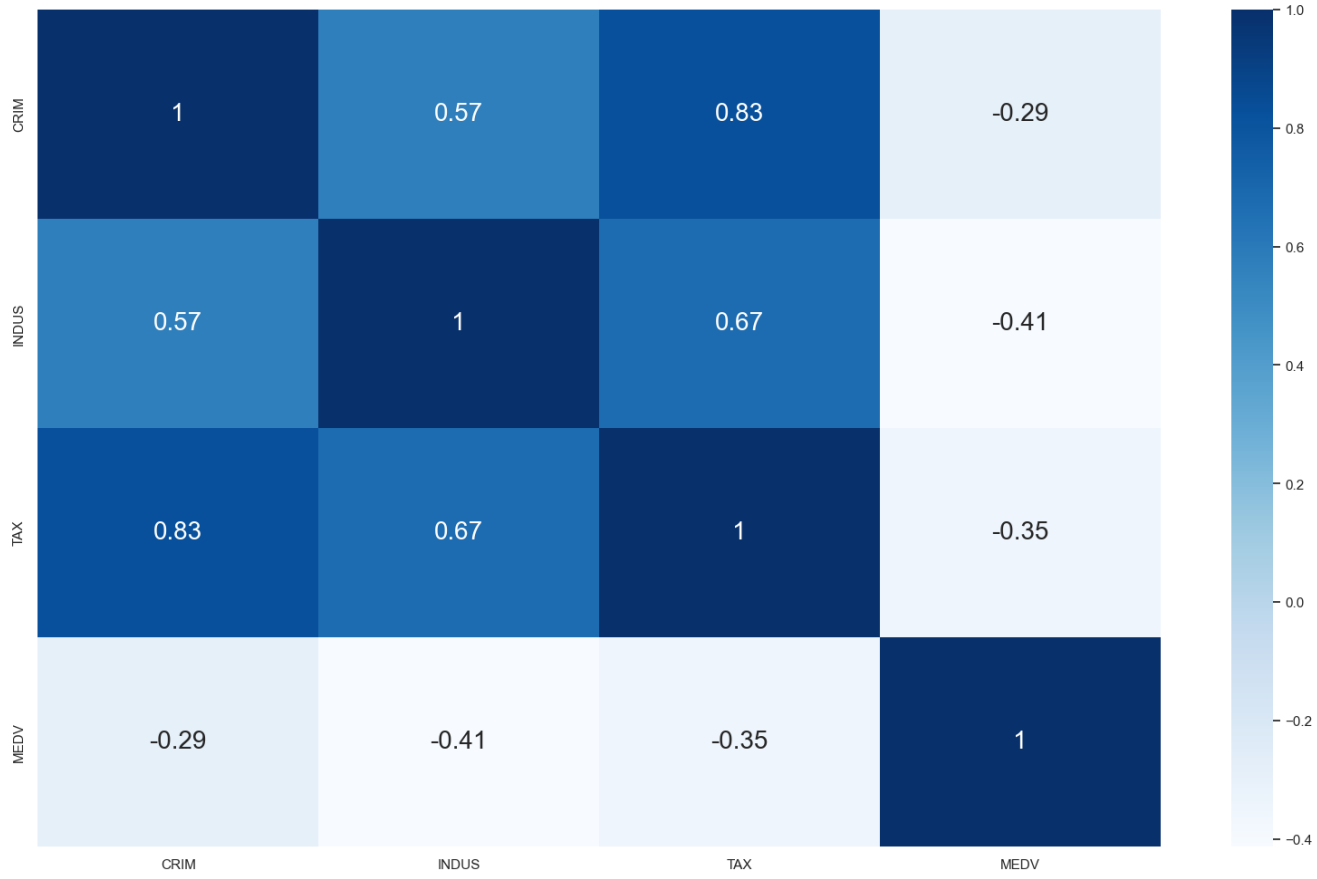


In [60]:
```
# Now we will make a heatmap with only the variables in df2 subset.  Again, it is very important to understand this
plt.figure(figsize =(20,12))
plt.figure(figsize = (20,12))
sns.heatmap(df2.corr(), annot=True)
plt.show()
```

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



In [61]:
```
#If you want to change the color and font, to make the labels easier to read, use this code.

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



Separate the Dataset into Input & Output NumPy Arrays

In [62]:
```
from sklearn.model_selection import train_test_split
```

In [63]:
```
# 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 your homework)
# 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 [64]:
```
# 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 the for seed radomization

seed = 7

# Split the dataset (both input & output) into training/testing datasets
# if random_state = None : Calling the function multiple times will produce different results.
# 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=seed)
```

Build and Train the Model

In [65]:
```
# Build the model

model=LinearRegression()

# Train the model using the training sub-dataset

model.fit(X_train, Y_train)

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

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

```
Intercept: 31.393427670412905
Coefficients: [ 0.09859287 -0.42388844 -0.00931847]
```

In [66]:
```
# If we want to print out the list of the coefficients with their correspondent variable name
# 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', -0.4238884417716135)

('CRIM', 0.09859287239143877)

('TAX', -0.009318474474503347)
```

In [67]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1 )
```

Out[67]:
```
LinearRegression
LinearRegression(n_jobs=1)
```

Calculate R-Squared

In [68]:
```
R_squared = model.score(X_test, Y_test)
print("R-squared: ", R_squared)
```

```
R-squared:  0.15473313373590392
```

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 [69]:
```
model.predict([[12,10,450]])
```

Out[69]:
```
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 USD 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 [70]:
```
# We are evaluating the algorithm
# we need to provide 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 squared error:",results.mean())
```

```
Average of all results from the K-fold Cross Validation, using negative 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 [71]:
```
# We are evaluating the algorithm
# we need to provide the K-size

num_folds = 10

# Now, we are fixing 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

# Splitting 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/algorithm

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)

# Printing 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 variance:",results.mean())
```

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

In [ ]:
```

```