Exploratory Data Analysis and Visualization Part 1

Introduction

Machine learning (ML) and neural networks (NN) require working with data. There are a number of important aspects of working with data. It must be understood in terms of its statistical properties, its cleanliness (data hygine), its completness (how to impute missing data), and its visualization. Getting familiar with some basic tools and processes for handling data is critical to making sense of it. Once the data is formatted to be more managable, it can be processed using the tools and techniques of machine learning to gain insights from the data.

This first article (in the series described in [7]) focuses on covering a handful of techniques related to using tools to manage the data. The references section lists resources used in the preparation of this article. It also provides the reader with the opportunity to acquire the materials for self-practice and extension. The references give a link to Anaconda, a set of applications tailored for use in data science (DS). This article and future articles will all be generated using Anaconda Navigator's JypyterLab notebooks [4]. This series of artciles will also heavily reuse the sample code and notebooks cited [2] to demonstrate different aspects of DS. These tutorials are not fully original because they try to give readers [1] the ability to reproduce results from these tutorials and references to gain relevant experience. A series of general purpose articles clearly will not be able to cover the level of detail to satisfy the more sophisticated reader, but will aide the layman trying to gain a vocabulary and familiarity with aspects of the subject matter. The primary goal being to create a learning by doing opportunity.

The following sections will cover the basics of using the Python programming language ecosystem tailored for DS. This includes demonstrating commonly used libraries like pandas, numpy, scipy, matplotlib and others. Given the vast depth of functionality in all these libraries, it is impossible for a series of tutorials to cover them comprehensively. The objective of this series will be to provide exposure to a swath of ideas in a logical progression. Many of the mentioned packages have extensive online documentation that can be consulted to gain better understanding of all their features, how to use them effectively, and many different kinds of examples. This article will conclude with showing some visualization library functionality for plotting and drawing data.

Theory

This section covers details related to acquiring data to use in a DS project. This includes examining a few readily available sources of data and acquiring externally available data. How to handle ingesting tablular data, and similar considerations for different data file formats. Lastly, how processed data is stored and managed inside a Pandas Dataframe (or Series for time series data).

Getting data to use for a DS project can be a challenge. There are many sources of data of varying quality. Sites like Kaggle [5] host many popular data sets used in many canonical examples. Data sets for many ML tasks typically consist of tabular data organized into columns and rows. The rows represent individual examples or samples of the thing being described (e.g. houses being sold, member species of a penguin colony, different wine grapes, cancer diagnosis). The columns represent the features (or sometimes called attributes or parameters) describing the example. Since the individual examples have to be identified as belonging to some category (used in classification tasks) or having some expected numerical value (used in regression tasks), these data sets have a column (typically called a target variable) that represents the label (or target value) given to the example. Many datasets must be examined in detail to determine if they contain a clean and complete set of relevant features for the things being described. Additionally, (to be examined in future articles) the label on each data example (row) serves as the ground truth when training an ML algorithm for regression or classification. This trained model then is utilized on new unlabeled data to infer the new data's target label or value.

The sklearn library mentioned earlier, contains many built-in data sets on a variety of topics. The sklearn documentation contains useful description of:

1. Toy datasets: https://scikit-learn.org/stable/datasets/toy_dataset.html
2. Real world datasets: https://scikit-learn.org/stable/datasets/real_world.html
3. Generated datasets: https://scikit-learn.org/stable/datasets/sample_generators.html
4. How to load these datasets: https://scikit-learn.org/stable/datasets/loading_other_datasets.html

Seaborn, another visualization library also has a number of built-in datasets. A summarized listing of each one is described here: https://www.geeksforgeeks.org/seaborn-datasets-for-data-science/

Lastly, a very popular canonical dataset used in many ML models is the Boston housing data that is available in an external file provided by [3] in its Datasets folder. This dataset is used in many regression tasks that require prediciting a house's value based on a set of features that describe the houses in the listing.

Analysis

Examining a few of the datasets available shows how to process and read the data into Pandas dataframes. Using samples from [6] shows what some of this data looks like.

Loading and Examining Datasets

In [78]:
```
from sklearn.datasets import load_diabetes # Import the function to load the diabetes dataset
import pandas as pd Readers
diabetes = load_diabetes() # Load the diabetes dataset
diabetes_df = pd.DataFrame(diabetes.data,columns=diabetes.feature_names)
diabetes_df.head()
```

Out[78]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019907	-0.017646
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068332	-0.092204
2	0.085299	0.050680	0.044451	-0.005670	-0.045599	-0.034194	-0.032356	-0.002592	0.002861	-0.025930
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022688	-0.009362
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031988	-0.046641

In [80]:
```
from sklearn.datasets import load_iris # Import the function to load the Iris dataset
iris = load_iris() # Load the Iris dataset
sk_iris_df = pd.DataFrame(iris.data,columns=iris.feature_names)
sk_iris_df.head()
```

Out[80]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [82]:
```
from sklearn.datasets import load_wine  # Import the function to load the wine dataset
wine = load_wine()  # Load the wine dataset
X = wine.data  # Extract the feature data
y = wine.target  # Extract the target data
wine_df = pd.DataFrame(X, columns=wine.feature_names)
wine_df.head()
```

Out[82]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82

A very popular visualization library is matplotlib. Some examples of using it to draw plots are given below. Another similar library built upon it is Seaborn. It also has a number of built-in datasets that can be examined. Some examples (repeat from other sources) include:

In [58]:
```
import seaborn as sns
# a dataset on identifying iris species
iris_df = sns.load_dataset('iris')
# examine the first few and last few items
iris_df.head()
```

Out[58]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [60]:

```
# a dataset on identifying tips given for restaurant meals by customers of different characteristics
tips_df = sns.load_dataset('tips')
# examine the first few and last few items
tips_df.head()
```

Out[60]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [70]:
```
# a dataset of fmri (functional magnetic resonance image)
fmri_df = sns.load_dataset('fmri')
fmri_df.head()
```

Out[70]:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

In [72]:
```
# a dataset on titanic ship disaster survivors (alive is the target variable, repeated in the survived colu
titanic_df = sns.load_dataset('titanic')
titanic_df.head()
```

Out[72]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	al
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	F
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	F
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	T
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	F
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	T

The pandas library has a number of built in methods that allow for reading different dataset file types. These files can be csv (comma separate values), excel, html, json, sql and many others. The built in method for csv files can be used to parse the Boston housing data file into a pandas dataframe as follows: (See [2] for detailed definition of columns of the Boston housing data.)

In [3]:
```
import pandas as pd
housing_df = pd.read_csv('HousingData.csv')

# examine the first few rows
housing_df.head()
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	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	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	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	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	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	NaN	36.2

In [5]:
```
# determine how many rows and columns are in the dataframe
housing_df.shape
```

Out[5]:
```
(506, 14)
```

The columns represent the parameters of interest that are the values each individual example takes on. This can be seen for any dataframe with the following command.

In [8]:
```
housing_df.columns
```

Out[8]:
```
Index(['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD', 'TAX',
       'PTRATIO', 'B', 'LSTAT', 'MEDV'],
      dtype='object')
```

Examining Metrics

A number of built in methods exist to provide some simple analysis of the dataframes extracted. These methods can be used to get descriptive statistics (describe) about the data. They can be used to get meta-data (information) related to the data types of each column, the quantity of data, how many rows might have null column values, and others. Additionally, some methods can be used to transform the stored data or fill in blanks once they are identified.

In [12]:
```
housing_df.describe()
```

Out[12]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	
count	486.000000	486.000000	486.000000	486.000000	506.000000	506.000000	486.000000	506.000000	506.000000	506.000
mean	3.611874	11.211934	11.083992	0.069959	0.554695	6.284634	68.518519	3.795043	9.549407	408.237
std	8.720192	23.388876	6.835896	0.255340	0.115878	0.702617	27.999513	2.105710	8.707259	168.537
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000
25%	0.081900	0.000000	5.190000	0.000000	0.449000	5.885500	45.175000	2.100175	4.000000	279.000
50%	0.253715	0.000000	9.690000	0.000000	0.538000	6.208500	76.800000	3.207450	5.000000	330.000
75%	3.560263	12.500000	18.100000	0.000000	0.624000	6.623500	93.975000	5.188425	24.000000	666.000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000

In [14]:
```
housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     486 non-null    float64
 1   ZN       486 non-null    float64
 2   INDUS    486 non-null    float64
 3   CHAS     486 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      486 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    int64  
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    486 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

In [16]:
```
# determine if any of the dataframe entries contain null values (i.e. blanks)
housing_df.isnull().any()
```

Out[16]:
```
CRIM        True
ZN          True
INDUS       True
CHAS        True
NOX        False
RM         False
AGE         True
DIS        False
RAD        False
TAX        False
PTRATIO    False
B          False
LSTAT       True
MEDV       False
dtype: bool
```

In [18]:
```
# find the locations of particular rows containing null values.
# This is to show the first five rows and all the columns.
housing_df.loc[:5, housing_df.isnull().any()]
```

Out[18]:

	CRIM	ZN	INDUS	CHAS	AGE	LSTAT
0	0.00632	18.0	2.31	0.0	65.2	4.98
1	0.02731	0.0	7.07	0.0	78.9	9.14
2	0.02729	0.0	7.07	0.0	61.1	4.03
3	0.03237	0.0	2.18	0.0	45.8	2.94
4	0.06905	0.0	2.18	0.0	54.2	NaN
5	0.02985	0.0	2.18	0.0	58.7	5.21

In [20]:
```
# examine descriptive statistics about the null data
housing_df.loc[:5, housing_df.isnull().any()].describe()
```

Out[20]:

	CRIM	ZN	INDUS	CHAS	AGE	LSTAT
count	6.000000	6.000000	6.000000	6.0	6.000000	5.000000
mean	0.032032	3.000000	3.831667	0.0	60.650000	5.260000
std	0.020401	7.348469	2.508907	0.0	11.134586	2.346838
min	0.006320	0.000000	2.180000	0.0	45.800000	2.940000
25%	0.027295	0.000000	2.180000	0.0	55.325000	4.030000
50%	0.028580	0.000000	2.245000	0.0	59.900000	4.980000
75%	0.031740	0.000000	5.880000	0.0	64.175000	5.210000
max	0.069050	18.000000	7.070000	0.0	78.900000	9.140000

Lastly, data hygiene must be applied to have a clean dataset. In future articles, the ML and NN algorithms that will be used require a complete set of data to operate on. Some algorithms are more flexible to missing data or different scales and units of data values. Depending on what the analysis plan is, will determine how much data massaging needs to be done.

Null values that are in the dataset can be filled. This is highly dependent on the data fields themselves. The fields could be numeric or text-based or a choice from a limited set of values in a category. So depending on the type of missing value and what makes the most sense for filling them in, a strategy has to be developed for the dataset to be cleaned up. In some cases, the null or blank values can be replaced with an appropriate value that represents an approximation of what data can be filled.

The data might be an interpolation of other values or an average of all values in the column or possibly a 0 if it makes sense to have such a value for a feature entry.

In [23]:
```
# See [2] (pgs 414-418).
# Replace the null values in the AGE column with its mean value
housing_df['AGE'] = housing_df['AGE'].fillna(housing_df.mean())
# Replace the null values in the CHAS column with 0
housing_df['CHAS'] = housing_df['CHAS'].fillna(0)
# Replace all remaining null values with median for the respective columns
housing_df = housing_df.fillna(housing_df.median())
# examine the updated values now have replaced all null values.  Compare with previous info
housing_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    float64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64  
 9   TAX      506 non-null    int64  
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(12), int64(2)
memory usage: 55.5 KB
```

This demonstrates a number of datasets that can be easily extracted from known sources and provide an initial clean set of information to utilize in ML exercises. The last data on Boston housing data provided a real world example of data with incomplete or non-uniform data values. The techniques shown can be utilized to compensate for poor data and fill in compensating values. The set of operations to execute on the Pandas dataframe are documented beyond the examples shown.

The descriptive statistics given earlier show the dataframe's methods to provide common functions for mean, mode, median, min, max. These are useful when trying to examine the data for outliers and get an idea of what are the rough bounds on values in the dataset.

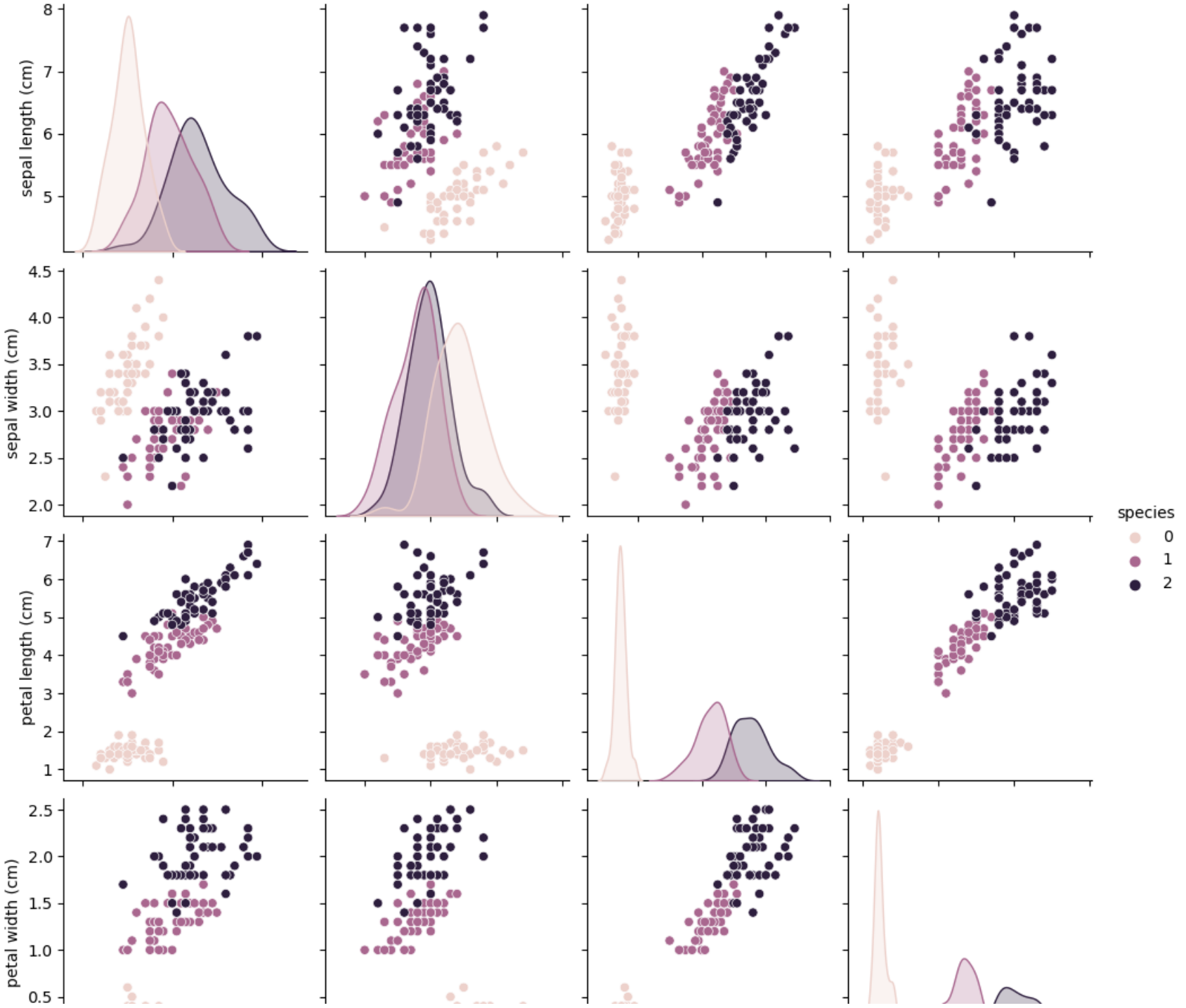
Experimental Results

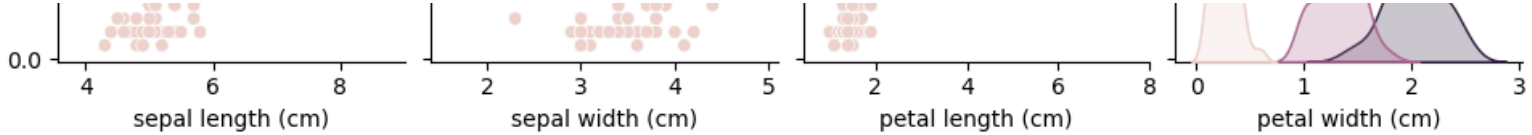
This section covers details of displaying and graphing the datasets using some graphics libraries. These figures show another way to explore the dataset to better understand the individual values contained in the dataframe, but also possibly other properties like correlation amongst values. These values are can be displayed in different types of graphs that come built into the libaries. Many of these examples are reproduced from [2] (pgs 419-437) and [6].

Graphing data

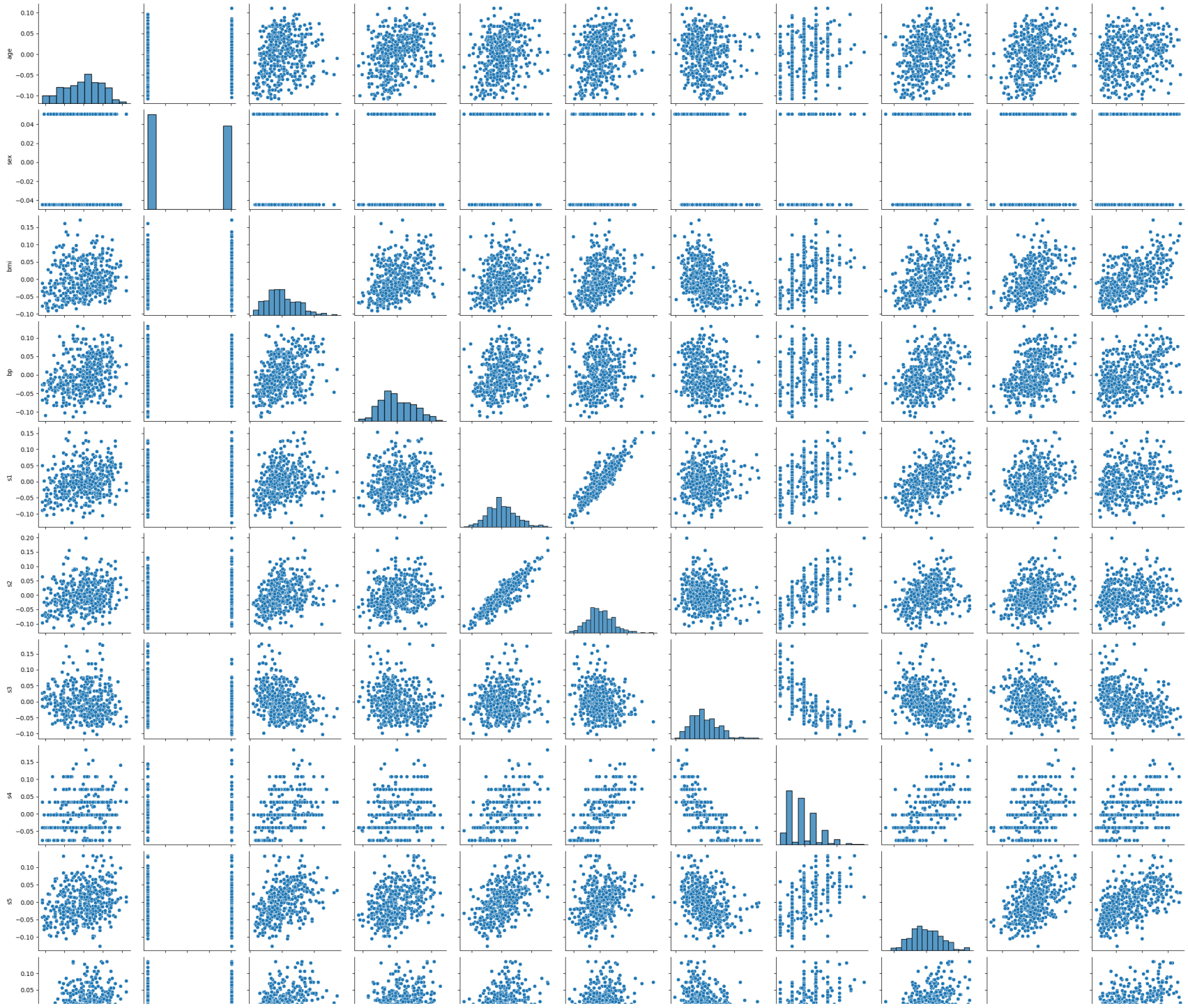
This next set of figures show a common built-in graph that demonstrates pair-wise scatter plots amongst the parameters found in the dataframe. These pair-wise plots allow one to visualize correlation amongst the parameters systematically to give a comprehensive look at the relationships amongst the values. Given the number of parameters, this set of plots can grow very quickly. It can make it very difficult to plot out all the combinations. This requires either using a different method to demonstrate the relationships or to reduce the number of parameters selected to plot.

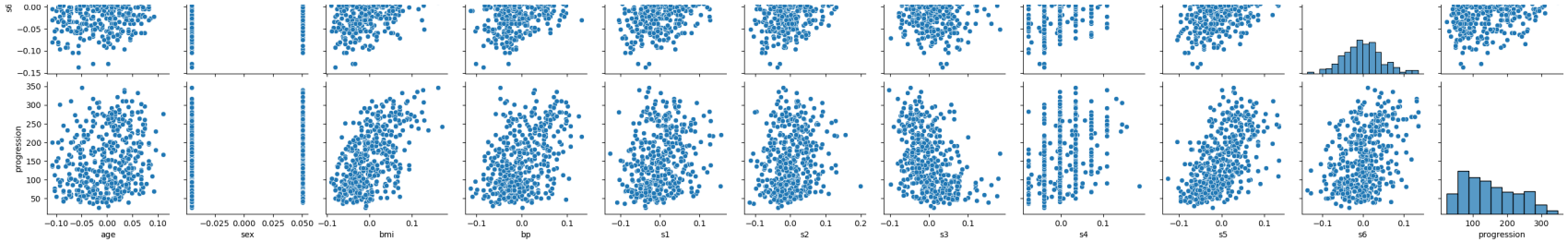
In [86]:
```
import matplotlib.pyplot as plt
sk_iris_df['species'] = iris.target
sns.pairplot(sk_iris_df, hue='species')
plt.show()
```


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
species
0
1
2



In [88]:
```
diabetes_df['progression'] = diabetes.target
sns.pairplot(diabetes_df)
plt.show()
```


age
sex
bmi
bp
s1
s2
s3
s4
s5

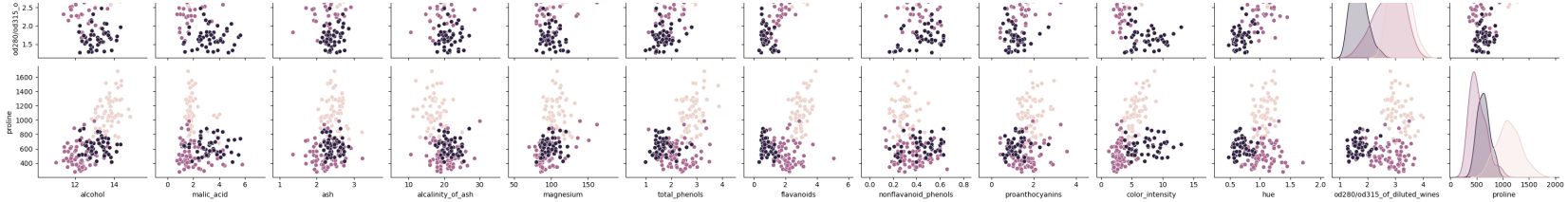


In [90]:

```
wine_df['cultivar'] = y

sns.pairplot(wine_df, hue='cultivar')
plt.show()
```

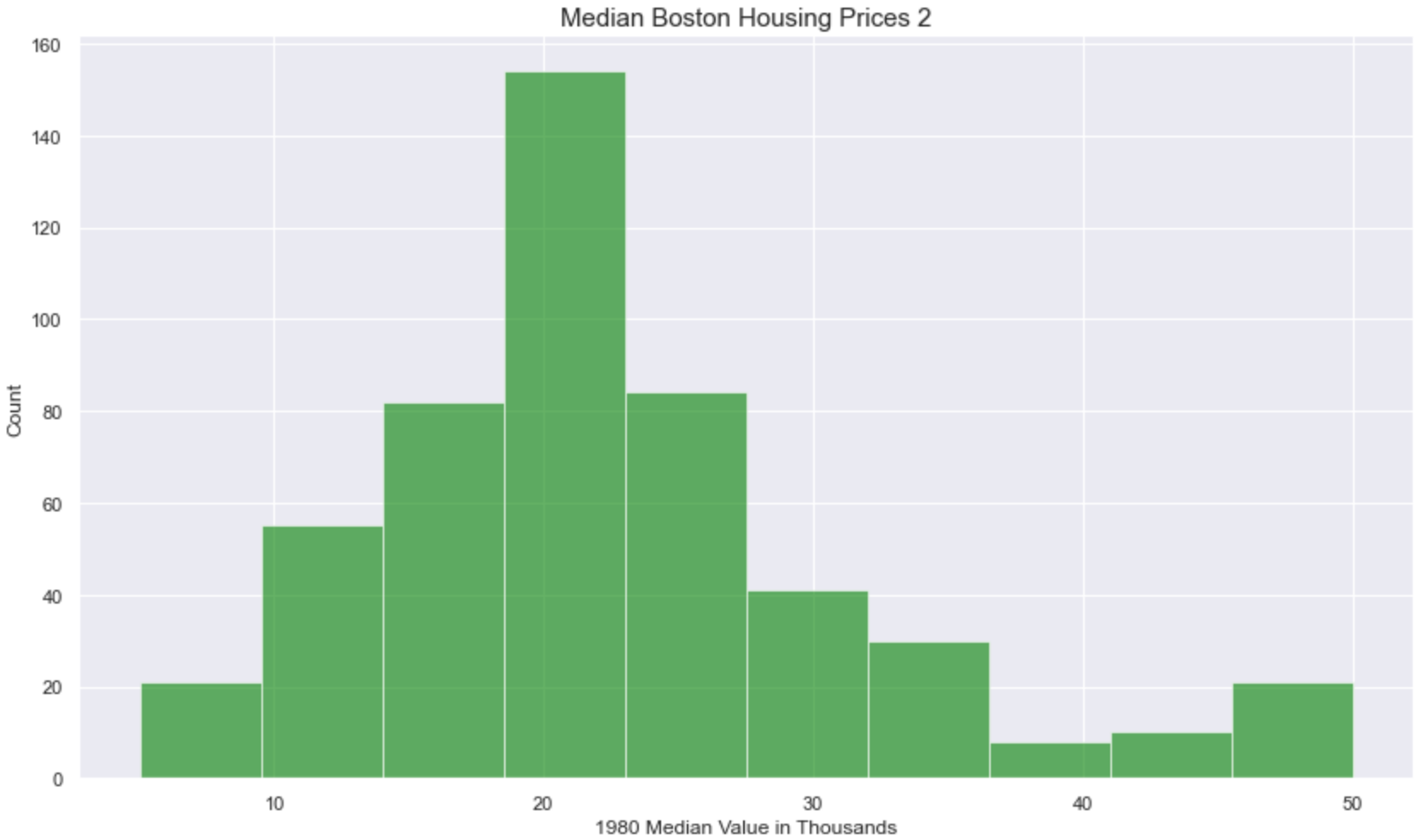

alcohol
malic_acid
ash
alcalinity_of_ash
magnesium
total_phenols
flavanoids
nonflavanoid_phenols
proanthocyanins
color_intensity
hue
od280/od315_of_diluted_wines
cultivar
0
1
2



Simple visualization of statistical descriptions.

This section covers basic graphing of one of the datasets supplied as a csv file. The Boston housing data is a good one for creating some simple graphs from the many different types of graphs available. A simple histogram of the median prices is drawn as follows (see the appendix for additional ways to draw):

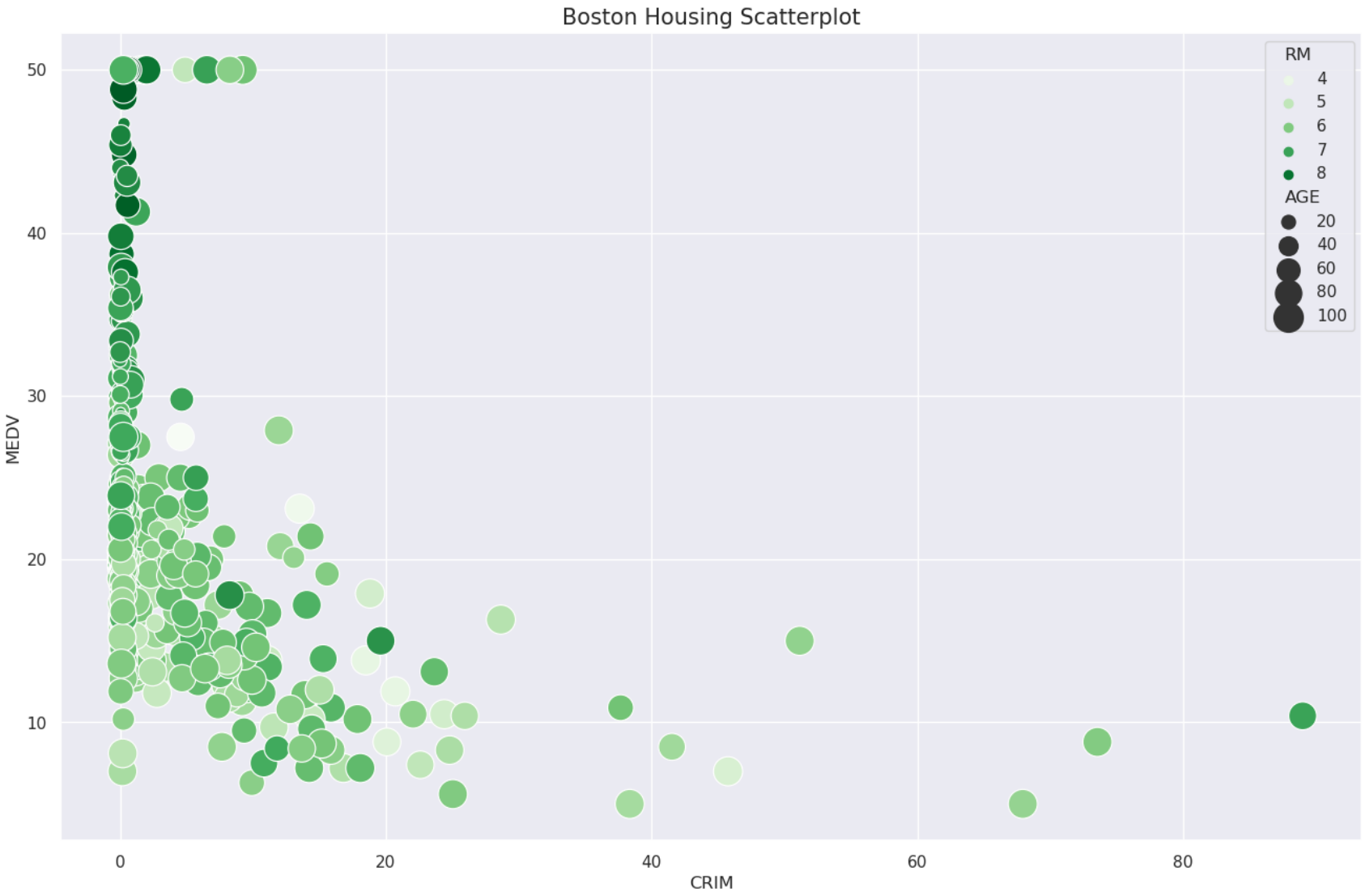
In [2]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
title = 'Median Boston Housing Prices 2'
plt.figure(figsize=(14,8))
plt.hist(housing_df['MEDV'], color='green', alpha=0.6)
plt.title(title, fontsize=15)
plt.xlabel('1980 Median Value in Thousands')
plt.ylabel('Count')
plt.savefig(title, dpi=300) # this line causes the graph to be saved with the name supplied.
plt.show()
```



Another very common type of plot is the scatter plot. Examining the specific columns of data plotted out shows some interesting properties in this dataset. The crime parameter has a noticable impact on the house median value parameter. The number of rooms is similar. The age does not have the same effect.

In [46]:
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
plt.figure(figsize=(16,10))
my_title='Boston Housing Scatterplot'
plt.title(my_title, size=15)
```

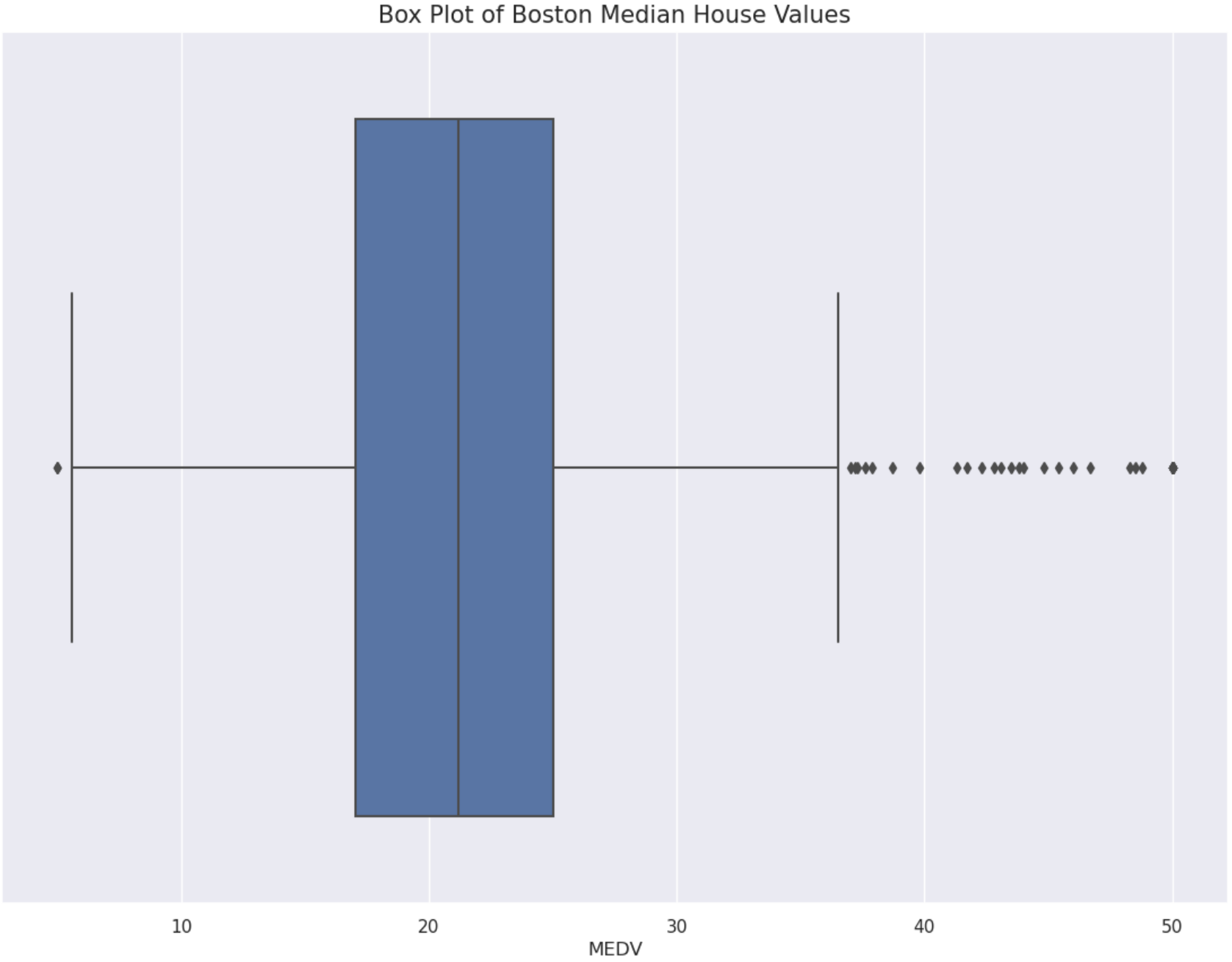
```
sns.scatterplot(x=housing_df['CRIM'], y=housing_df['MEDV'],
                hue=housing_df['RM'], size=housing_df['AGE'],
                sizes=(20, 400), palette='Greens')
# plt.savefig(my_title, dpi=300)  # uncomment this line to save the figure
plt.show()
```



Box Plots are another type of built in graph that can quickly show a number of statistical relationships amongst the parameters selected. It can provide a way of exploring the data to see what are the basic bounds of the dataset values. The median, the 25th

and 75th percentiles are shown. Additionally, the 1.5 times the interquartile range is given by the end-bars. The diamonds are outliers.

In [59]:
```
plt.figure(figsize=(14, 10))
title='Box Plot of Boston Median House Values'
plt.title(title, size=15)
sns.boxplot(x = housing_df['MEDV'])
# plt.savefig(title, dpi=300) to save the figure
plt.show()
```



There is a built in function showing correlation amongst the values as a dataframe chart. This is easier to visualize as a heatmap

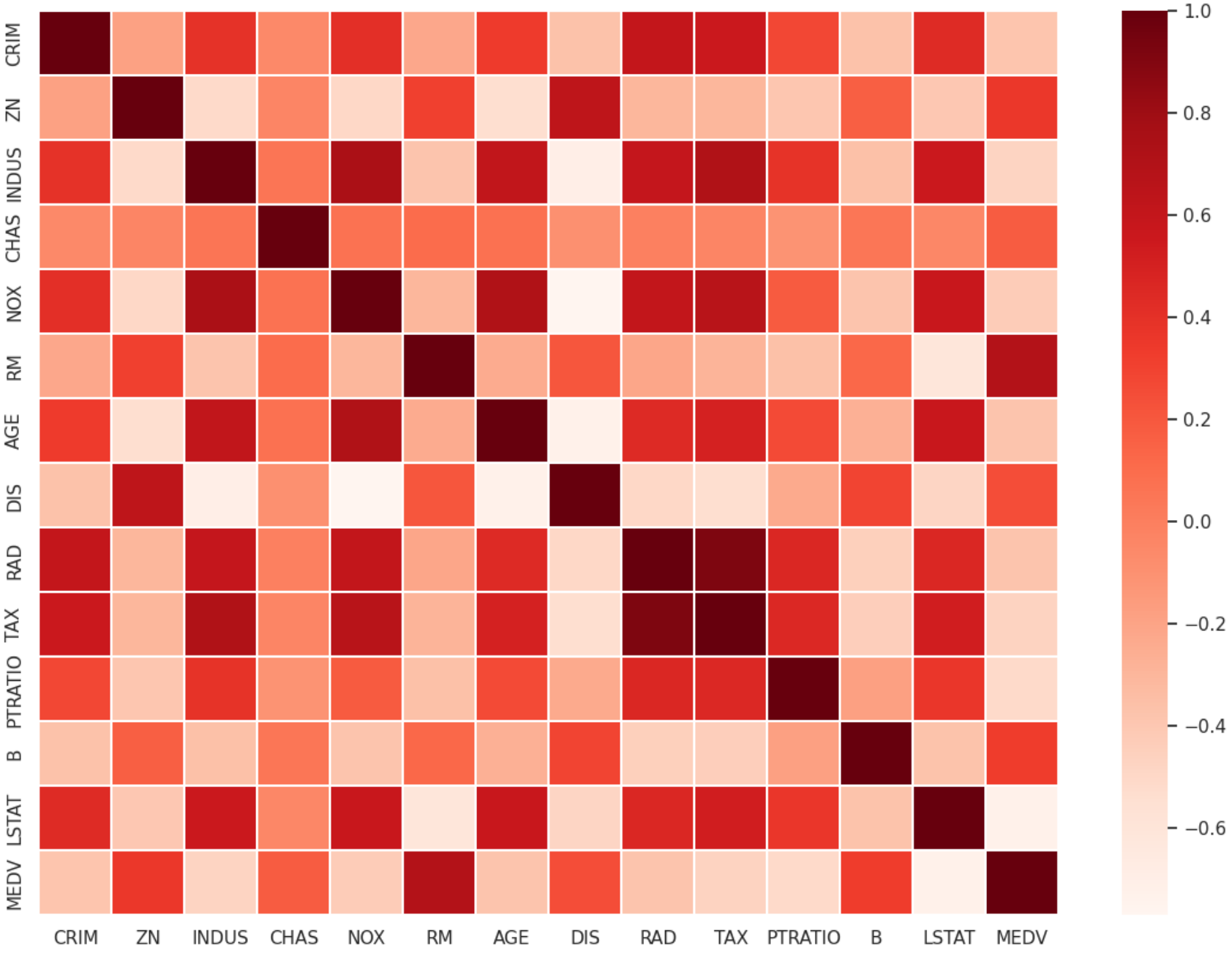
and that is also drawn. Lastly, a regression line and plot are shown with the shaded region representing the 95% confidence interval. This figure is limited because only two parameters (number of rooms and median price) are selected to show regression between them in a scatter plot. Visually it is easier to see, but lacks some details. To demonstrate another way of generating more statistical data is by using the StatsModel library and showing the ordinary least squares method of getting those statistics.

In [64]:
```
housing_df.corr()
```

Out[64]:

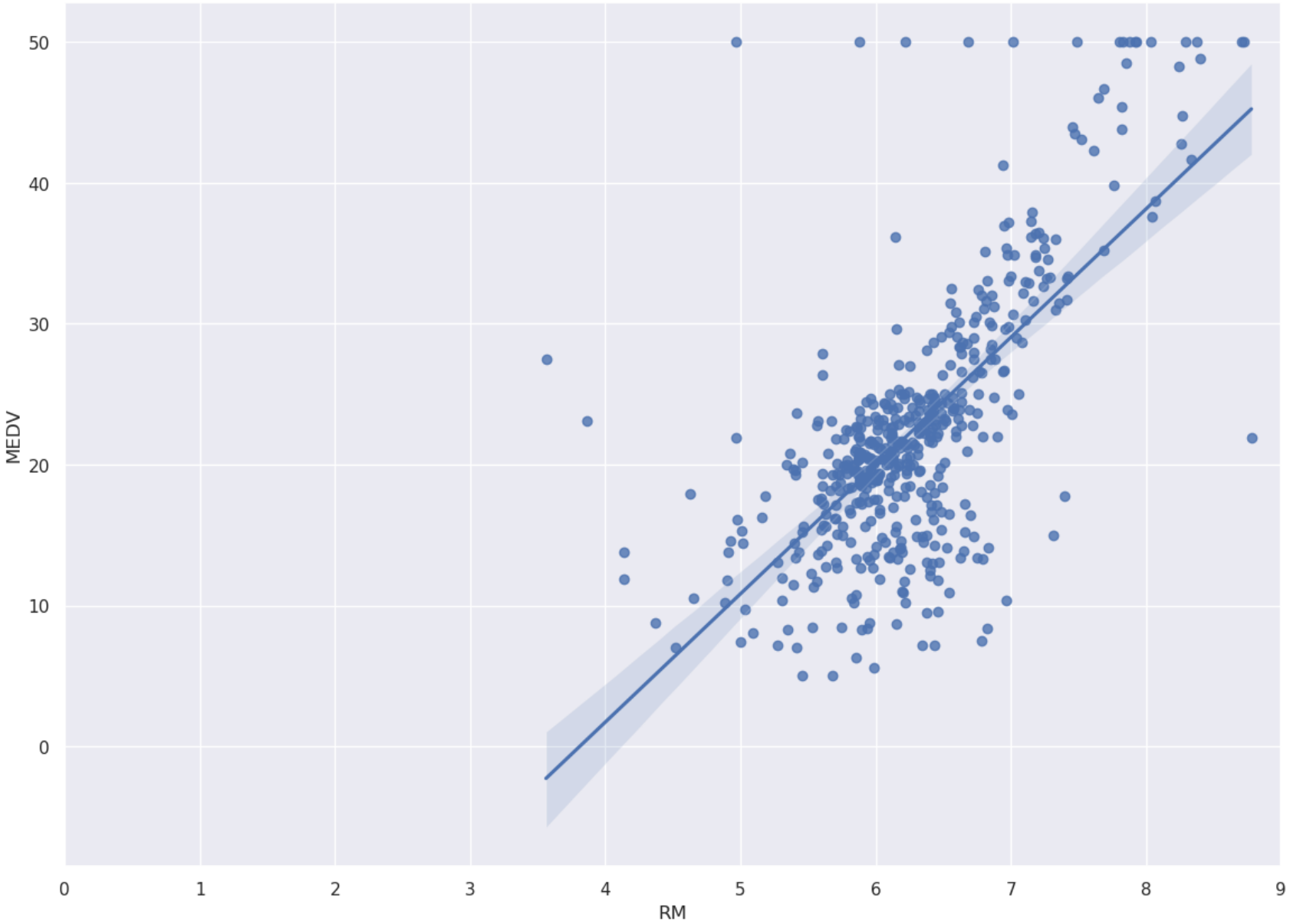
	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATI
CRIM	1.000000	-0.185359	0.392632	-0.055585	0.410971	-0.220045	0.343427	-0.366025	0.601224	0.560469	0.27796
ZN	-0.185359	1.000000	-0.507304	-0.032992	-0.498619	0.312295	-0.535341	0.632428	-0.300061	-0.304385	-0.39462
INDUS	0.392632	-0.507304	1.000000	0.054693	0.738387	-0.377978	0.614248	-0.698621	0.592735	0.716267	0.38536
CHAS	-0.055585	-0.032992	0.054693	1.000000	0.070867	0.106797	0.074984	-0.092318	-0.003339	-0.035822	-0.10945
NOX	0.410971	-0.498619	0.738387	0.070867	1.000000	-0.302188	0.711864	-0.769230	0.611441	0.668023	0.18893
RM	-0.220045	0.312295	-0.377978	0.106797	-0.302188	1.000000	-0.239518	0.205246	-0.209847	-0.292048	-0.35550
AGE	0.343427	-0.535341	0.614248	0.074984	0.711864	-0.239518	1.000000	-0.724354	0.447088	0.498408	0.26182
DIS	-0.366025	0.632428	-0.698621	-0.092318	-0.769230	0.205246	-0.724354	1.000000	-0.494588	-0.534432	-0.23247
RAD	0.601224	-0.300061	0.592735	-0.003339	0.611441	-0.209847	0.447088	-0.494588	1.000000	0.910228	0.46474
TAX	0.560469	-0.304385	0.716267	-0.035822	0.668023	-0.292048	0.498408	-0.534432	0.910228	1.000000	0.46085
PTRATIO	0.277964	-0.394622	0.385366	-0.109451	0.188933	-0.355501	0.261826	-0.232471	0.464741	0.460853	1.00000
B	-0.365336	0.170125	-0.354840	0.050608	-0.380051	0.128069	-0.268029	0.291512	-0.444413	-0.441808	-0.17738
LSTAT	0.437417	-0.398838	0.564508	-0.047279	0.573040	-0.604323	0.575022	-0.483244	0.467765	0.524156	0.37072
MEDV	-0.383895	0.362292	-0.476394	0.183844	-0.427321	0.695360	-0.377572	0.249929	-0.381626	-0.468536	-0.50778

In [67]:
```
corr = housing_df.corr()
plt.figure(figsize=(14,10))
sns.heatmap(corr, xticklabels=corr.columns.values,
            yticklabels=corr.columns.values, cmap="Reds", linewidths=1.25)
plt.show()
```



In [78]:
```
plt.figure(figsize=(14, 10))
sns.regplot(x = housing_df['RM'],y = housing_df['MEDV'])
```

```
plt.xlim(0, 9)
plt.show()
```



In [81]:
```
x = housing_df['RM']
y = housing_df['MEDV']
```

```python
# use the library to generate a model and fit it on the two parameters of data
import statsmodels.api as sm
X = sm.add_constant(x)
model = sm.OLS(y, X)
est = model.fit()
print(est.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   MEDV   R-squared:                       0.484
Model:                            OLS   Adj. R-squared:                  0.483
Method:                 Least Squares   F-statistic:                     471.8
Date:                Sat, 25 Jan 2025   Prob (F-statistic):           2.49e-74
Time:                        16:29:44   Log-Likelihood:                -1673.1
No. Observations:                 506   AIC:                             3350.
Df Residuals:                     504   BIC:                             3359.
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -34.6706      2.650    -13.084      0.000     -39.877     -29.465
RM             9.1021      0.419     21.722      0.000       8.279       9.925
==============================================================================
Omnibus:                      102.585   Durbin-Watson:                   0.684
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              612.449
Skew:                           0.726   Prob(JB):                    1.02e-133
Kurtosis:                       8.190   Cond. No.                         58.4
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

As mentioned earlier, the regression line on the scatter plot of the two parameters is being analyzed with this chart generated by the StatsModel library. The dependent variable is the median value of the home. This is the target value that is in this current dataset. It would also be the value that would be predicted if supplied (in this case) a home with the specified number of rooms.

Each individual element is not described here in detail, but can be looked up in most statistics reference. Briefly, the chart gives degrees of freedom, the residuals (which contributes to understanding the standard deviation), covariance type indicates the measure of relatedness of the two dependent random variables, the F statistic is involved in a significance test, skewness is a measure of the tail of a distribution, kurtosis is a measure of the peakedness of a distribution, and other descriptors of the

dataset.

The R^2 value represents the coefficient of determination qualifying how predictive the regression line is. Similarly, the std err column tells how far the actual values are from the line.

Conclusions

This article showed some limited details about exploratory data analysis. This was accomplished by loading different datasets from a variety of sources and formats. These dataset values were examined and cleaned up as needed before attempting to analyze them further. A number of different graphs and how to draw them based on the data were shown. These graphs were to gain insight about the dataset and relationships that exist between the different features and target variables for each example or sample.

Finally, a few methods of examining the statistical properties of the dataframe were examined. This allowed for different properties of the dataset to be examined. It also showed how to determine if attempts to find parameters that impact the target value are predictive or not based on examining the StatsModel library to analyze the selected parameters and yield different statistical measures about the values.

Similar aspects of exploratory data analysis will be the subject of a future article. This will help solidfy the common tools used to prepare data prior to using it in models for training and prediction.

References

1. It is assumed the reader is trained in basic engineering level mathematics and Python programming. Familiarity with statistics, probability, linear algebra, calculus will greatly aide understanding. Many free online articles, videos, and courses are available to fill in gap areas with an online search.
2. The Python Workshop, Second Edition. https://www.amazon.com/dp/1804610615
3. The Python Workshop, source code: https://www.packtpub.com/product/the-python-workshop-second-edition/9781804610619?utm_source=github&utm_medium=repository&utm_campaign=9781804610619
4. Anaconda Navigator (https://www.anaconda.com/products/navigator)
5. Kaggle (www.kaggle.com)

6. https://www.geeksforgeeks.org/top-inbuilt-datasets-in-scikit-learn-library/
7. https://ap20.github.io/nnj/NL/apatel/AITutorialSeries2.html

Appendix

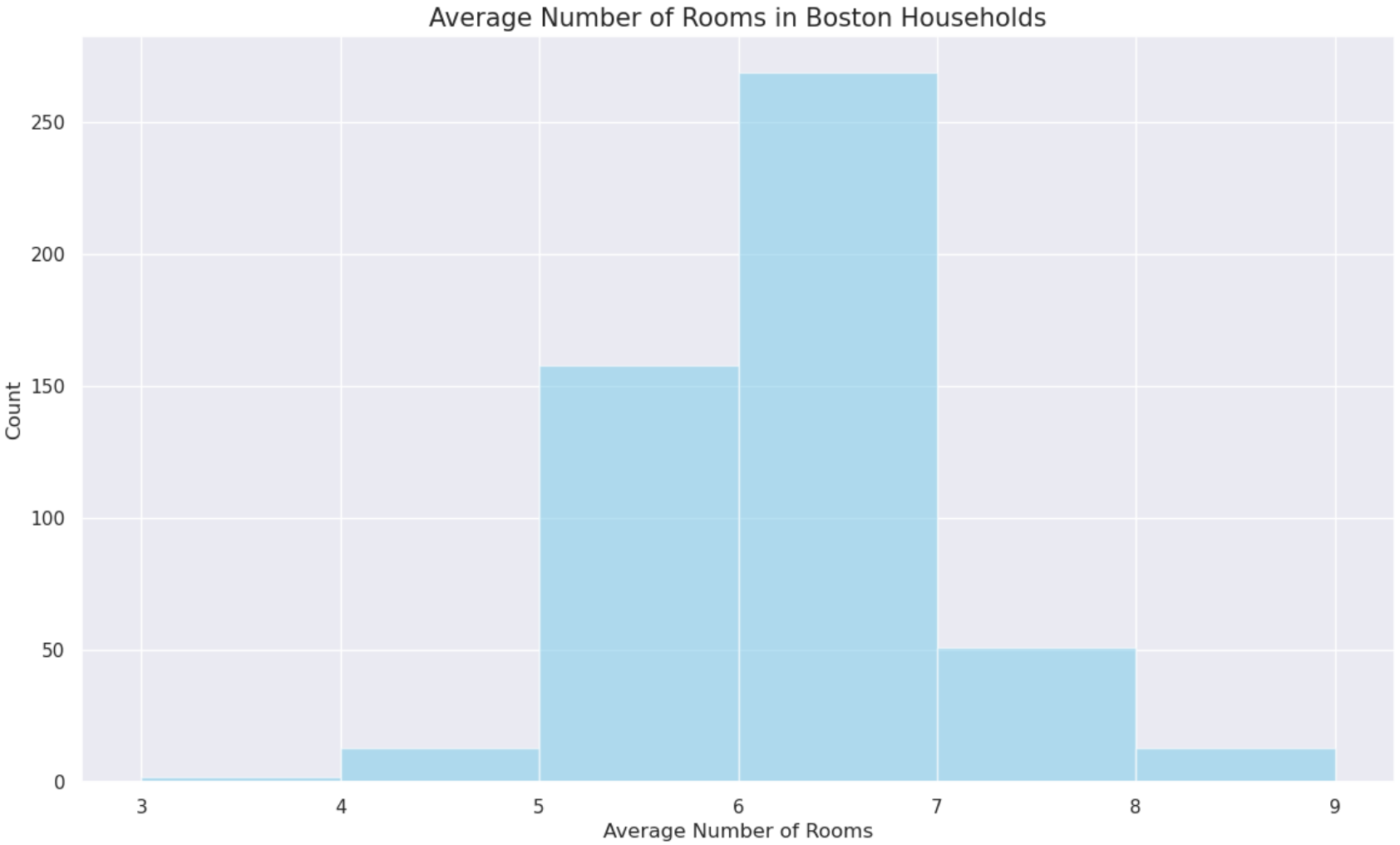
A simple function for drawing graphs

This section covers a function from [3], on showing how to draw multiple graphs. This kind of programming is necessary in order to reuse the same functionality, without having to repeat the code in multiple cells. This makes it easier to maintain and achieve the same operations. These general functions can also be saved into a library for use with other notebooks.

As can be seen in the two separate calls for plotting histograms for rooms and age attributes of a house. Simply passing the right arguments to the defined function is sufficient to customize and reuse the same function to get the required plots.

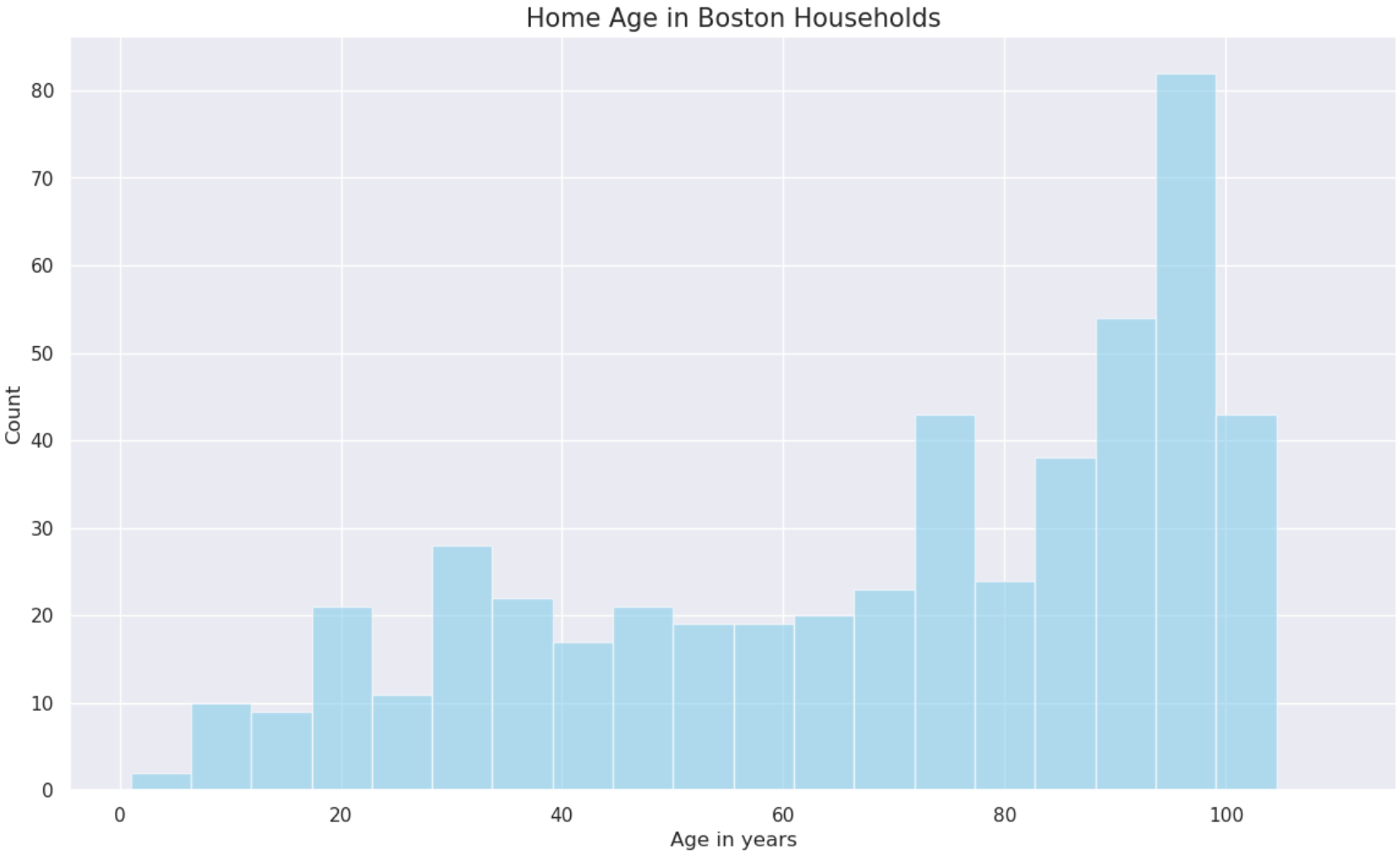
In [121…
```
def my_hist(column, title, xlab, ylab='Count', color='green', alpha=0.6, bins=10, range=None):
    import matplotlib.pyplot as plt
    import seaborn as sns
    sns.set()
    plt.figure(figsize=(14,8))
    plt.hist(column, color=color, alpha=alpha, bins=bins, range=range)
    plt.title(title, fontsize=15)
    plt.xlabel(xlab)
    plt.ylabel(ylab)
    #plt.savefig(title, dpi=300)
    plt.show()
```

In [123…
```
my_hist(housing_df['RM'], 'Average Number of Rooms in Boston Households', 'Average Number of Rooms',
        color='skyblue', bins=6, range=(3,9))
```



In [135…

```
my_hist(housing_df['AGE'], 'Home Age in Boston Households', 'Age in years',
        color='skyblue', bins=20, range=(1,110))
```



Manipulation of dataframe contents

A very common operation that is required when managing a dataset is the need to possibly remove columns that serve no purpose, or are duplicate or are inadequate for supplying into a model's algorithm. This requires changing the dataframe into another format. Lets use the same Boston Housing dataset for this purpose by examining what is in it and how useful it might be.

Looking at the correlation table and heatmap, there are some parameters (features) that stand out as not relvant to other data. The parameter has no positive correlation and could be removed.

In [11]:
```
import pandas as pd
new_housing_df = pd.read_csv('HousingData.csv')

# examine the first few rows
new_housing_df.head()
new_housing_df1 = new_housing_df
del new_housing_df1['DIS']
del new_housing_df1['RAD']
print(new_housing_df1.shape)
new_housing_df1.head()
```

```
(506, 12)
```

Out[11]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	222	18.7	396.90	NaN	36.2

Compared to the original housing dataframe of 506 rows and 14 columns. This new_housing_df has been reduced by two. It deleted the 'DIS' column, which is distance to employment centers from the neighborhood. It deleted the 'RAD' column, which is accessibility to radial highways.

Additionally, the 'CRIM' column is the per capita crime rate by town. But it might not be the right units for comparison purposes and needs to be transformed to per 1000 persons. This would require multiplying all entries by 1000 to make it more usable. It can be done as follows:

In [13]:
```
new_housing_df2 = new_housing_df
new_housing_df2['CRIM'] = new_housing_df['CRIM']*1000
new_housing_df2.head()
```

Out[13]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	TAX	PTRATIO	B	LSTAT	MEDV
0	6.32	18.0	2.31	0.0	0.538	6.575	65.2	296	15.3	396.90	4.98	24.0
1	27.31	0.0	7.07	0.0	0.469	6.421	78.9	242	17.8	396.90	9.14	21.6
2	27.29	0.0	7.07	0.0	0.469	7.185	61.1	242	17.8	392.83	4.03	34.7
3	32.37	0.0	2.18	0.0	0.458	6.998	45.8	222	18.7	394.63	2.94	33.4
4	69.05	0.0	2.18	0.0	0.458	7.147	54.2	222	18.7	396.90	NaN	36.2

Sometimes new columns of data need to be added. This could be real data obtained from observations, or it can be synthetic data created from other columns of data. The latter case applies when it is necessary to summarize a subset of columns into one summary statistic that can be used in the model as a better or more descriptive representative feature.

For example, if the 'ZN' feature (defined as proportion of residential land zoned for lots over 25K sq ft) and the 'CHAS' feature (which is 1 for tracts bound by the Charles river and 0 otherwise) could be combined to find all those large homes near the river, then that could be a new column of data. Depending on this new feature's relationship with other features (possibly via the correlation analysis seen earlier), could help determine if this subset of house prices are artificially skewing the median value of a home. This could lead to removing those two features from the overall feature set and model training.

In [32]:
```
# of the 500+ examples, determine how many of the 'ZN' and 'CHAS' features are non-zero for house entries.
print(new_housing_df2['ZN'].count())
print(new_housing_df2['CHAS'].count())
# count up the number of houses close to the river
print(new_housing_df2['CHAS'].sum())
# count up the number of houses greater than size
nhdf2 = new_housing_df2['ZN'] > 0
# produces a series from the dataframe of true and false values satifying the expression
print(nhdf2.value_counts())
# using this info, can generate a new column to capture as BHNR - big houses near river
new_housing_df2['BHNR'] = new_housing_df2['CHAS'] * new_housing_df2['ZN']
# capture those non-zero values into a new data series and get its basic stats
nhdf3 = new_housing_df2['BHNR'] >0
print(nhdf3.value_counts())
```

```
486
486
34.0
False    380
True     126
Name: ZN, dtype: int64
False    499
True       7
Name: BHNR, dtype: int64
```

These types of operations now cause the new dataframes to contain different values which can be used for plotting or different kinds of analysis. The last step yielded a data frame with the new feature 'BHNR' and it shows only 7 houses satisfy both of those criteria from the near 500 examples. With so few houses in the total set, the prices of those houses should not skew the overall median value. Similar other in-depth examination of feature values can be done to determine their impact on the model to be constructed.