Activity 5: Generating Statistics from a CSV File

1. Load the necessary libraries.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

2. Read in the Boston housing dataset (given as a .csv file) from the local directory.

In [2]:
```
boston_housing_df = pd.read_csv('Boston_housing.csv')
```

3. Check the first 10 records. Find the total number of records.

In [3]:
```
print('Total number of records in dataframe : ',len(boston_housing_df))
boston_housing_df.head(10)
```

```
Total number of records in dataframe :  506
```

Out[3]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
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
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43	22.9
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.90	19.15	27.1
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311	15.2	386.63	29.93	16.5
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.10	18.9

4. Create a smaller DataFrame with columns that do not include CHAS, NOX, B, and LSTAT.

In [4]:
```
boston_housing_df_2 = boston_housing_df.loc[:, ~boston_housing_df.columns.isin(['CHAS','
boston_housing_df_2
```

Out[4]:

	CRIM	ZN	INDUS	RM	AGE	DIS	RAD	TAX	PTRATIO	PRICE
0	0.00632	18.0	2.31	6.575	65.2	4.0900	1	296	15.3	24.0
1	0.02731	0.0	7.07	6.421	78.9	4.9671	2	242	17.8	21.6
2	0.02729	0.0	7.07	7.185	61.1	4.9671	2	242	17.8	34.7
3	0.03237	0.0	2.18	6.998	45.8	6.0622	3	222	18.7	33.4
4	0.06905	0.0	2.18	7.147	54.2	6.0622	3	222	18.7	36.2
...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	6.593	69.1	2.4786	1	273	21.0	22.4
502	0.04527	0.0	11.93	6.120	76.7	2.2875	1	273	21.0	20.6

503	0.06076	0.0	11.93	6.976	91.0	2.1675	1	273	21.0	23.9
504	0.10959	0.0	11.93	6.794	89.3	2.3889	1	273	21.0	22.0
505	0.04741	0.0	11.93	6.030	80.8	2.5050	1	273	21.0	11.9

506 rows × 10 columns

5. Check the last seven records of the new DataFrame you just created.

In [5]:
```
boston_housing_df_2.tail(7)
```

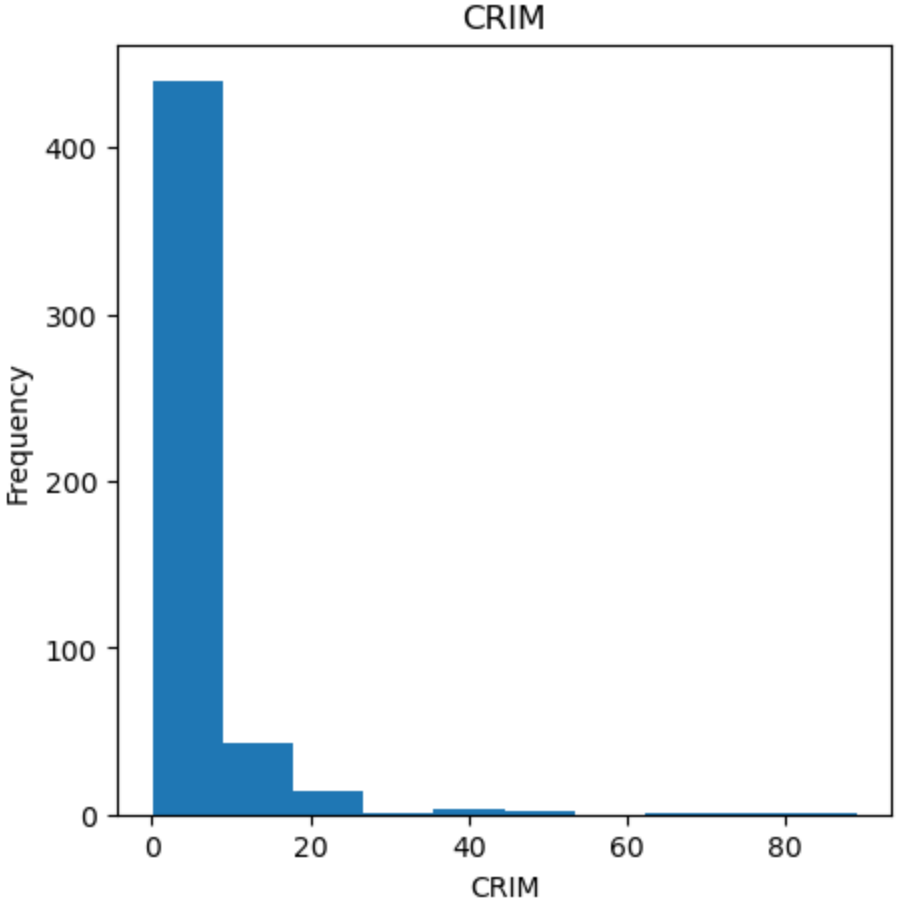
Out[5]:

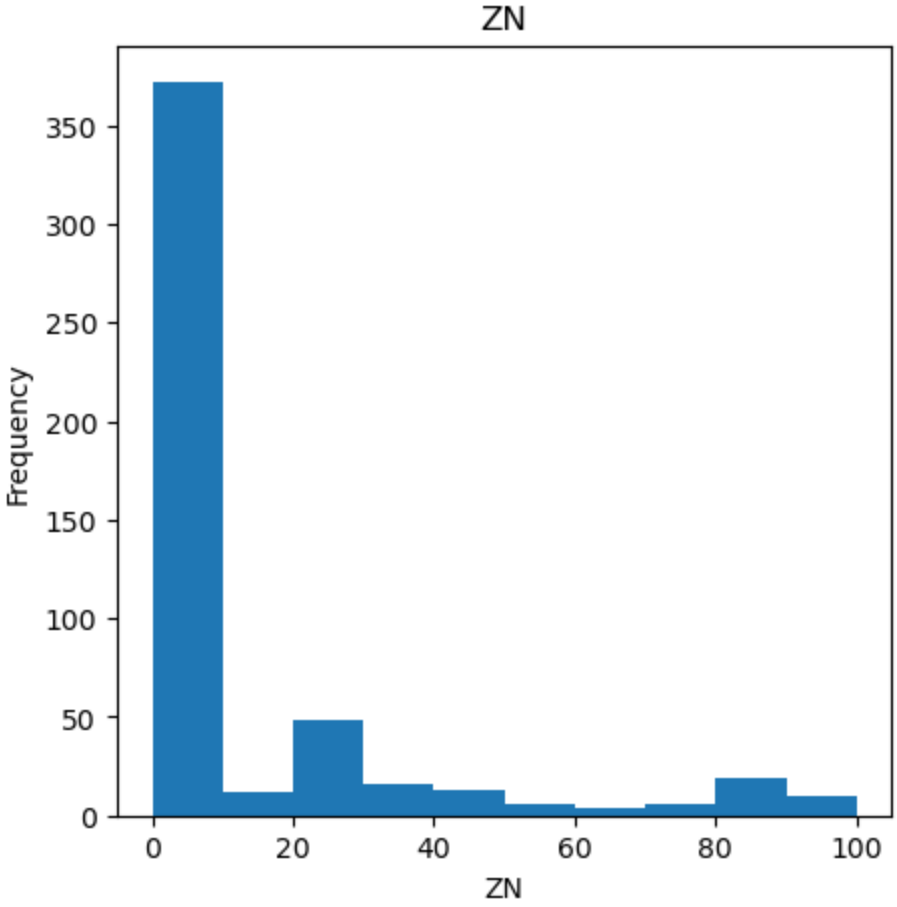
	CRIM	ZN	INDUS	RM	AGE	DIS	RAD	TAX	PTRATIO	PRICE
499	0.17783	0.0	9.69	5.569	73.5	2.3999	6	391	19.2	17.5
500	0.22438	0.0	9.69	6.027	79.7	2.4982	6	391	19.2	16.8
501	0.06263	0.0	11.93	6.593	69.1	2.4786	1	273	21.0	22.4
502	0.04527	0.0	11.93	6.120	76.7	2.2875	1	273	21.0	20.6
503	0.06076	0.0	11.93	6.976	91.0	2.1675	1	273	21.0	23.9
504	0.10959	0.0	11.93	6.794	89.3	2.3889	1	273	21.0	22.0
505	0.04741	0.0	11.93	6.030	80.8	2.5050	1	273	21.0	11.9

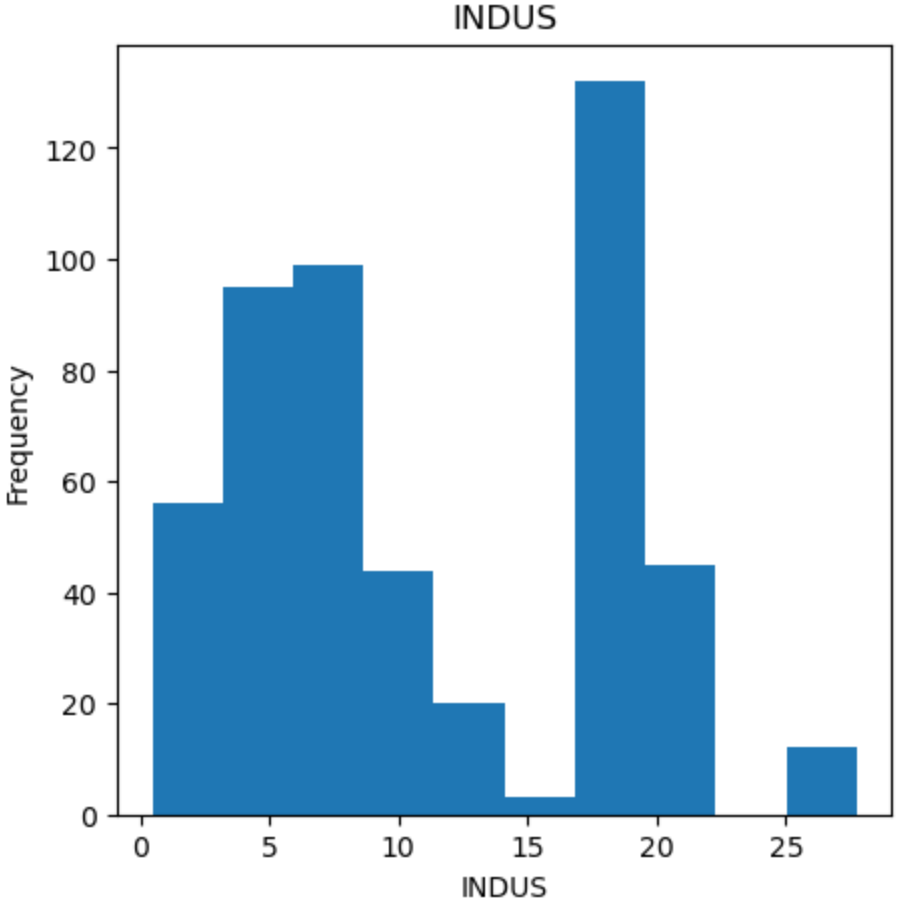
6. Plot the histograms of all the variables (columns) in the new DataFrame.

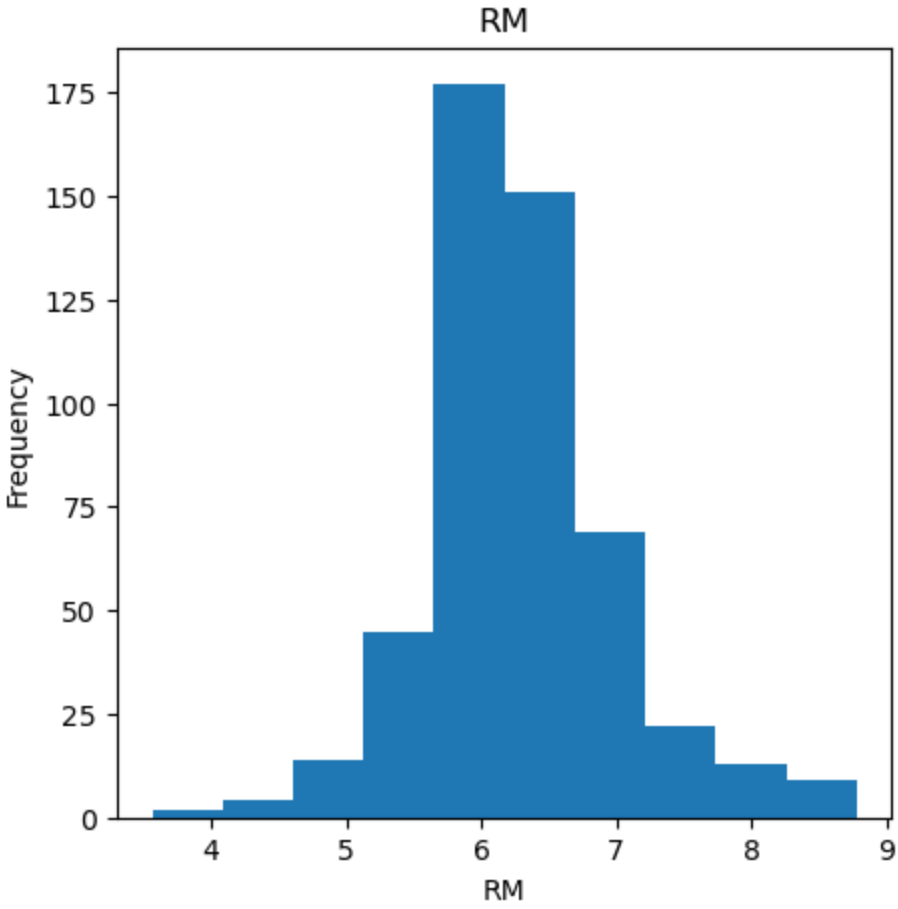
In [6]:
```
fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.CRIM)
plt.xlabel("CRIM")
plt.ylabel("Frequency")
plt.title('CRIM')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.ZN)
plt.xlabel("ZN")
plt.ylabel("Frequency")
plt.title('ZN')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.INDUS)
plt.xlabel("INDUS")
plt.ylabel("Frequency")
plt.title('INDUS')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.RM)
plt.xlabel("RM")
plt.ylabel("Frequency")
plt.title('RM')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.AGE)
plt.xlabel("AGE")
plt.ylabel("Frequency")
plt.title('AGE')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.DIS)
plt.xlabel("DIS")
plt.ylabel("Frequency")
```

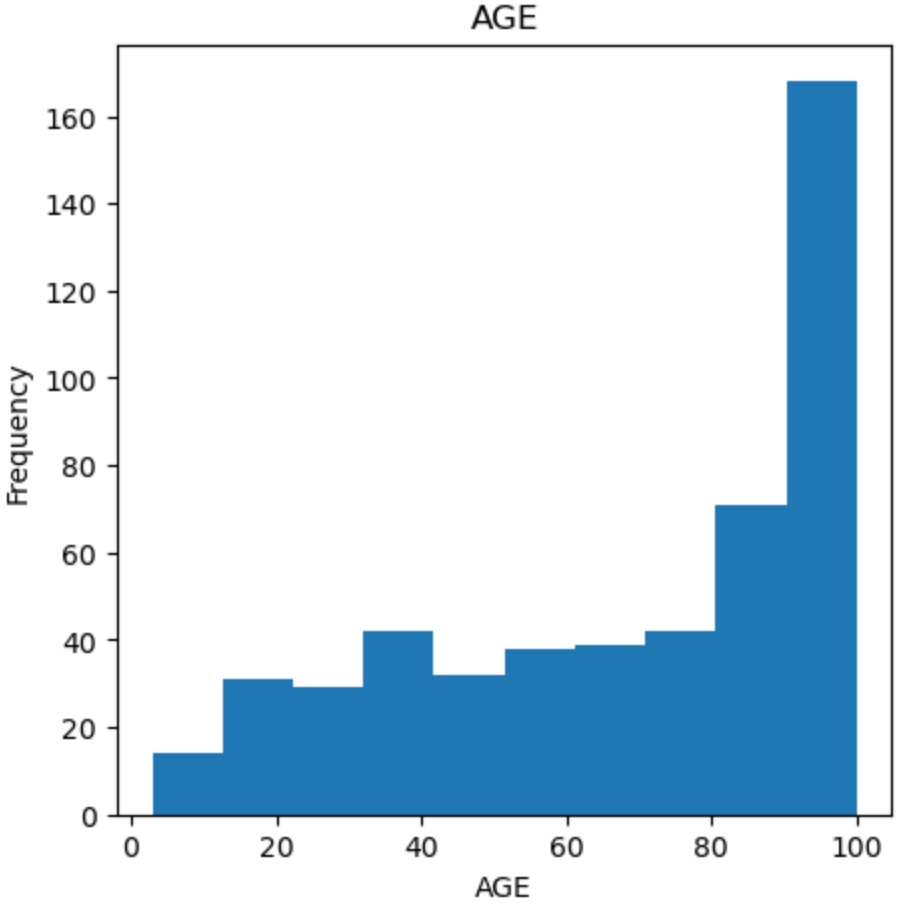
```
plt.title('DIS')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.RAD)
plt.xlabel("RAD")
plt.ylabel("Frequency")
plt.title('RAD')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.TAX)
plt.xlabel("TAX")
plt.ylabel("Frequency")
plt.title('TAX')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.PTRATIO)
plt.xlabel("PTRATIO")
plt.ylabel("Frequency")
plt.title('PTRATIO')

##fig, ax = plt.subplots(figsize =(5, 5))
##ax.hist(boston_housing_df_2.B)
##plt.xlabel("B")
##plt.ylabel("Frequency")
##plt.title('B')

##fig, ax = plt.subplots(figsize =(5, 5))
##ax.hist(boston_housing_df_2.LSTAT)
##plt.xlabel("LSTAT")
##plt.ylabel("Frequency")
##plt.title('LSTAT')

fig, ax = plt.subplots(figsize =(5, 5))
ax.hist(boston_housing_df_2.PRICE)
plt.xlabel("PRICE")
plt.ylabel("Frequency")
plt.title('PRICE')

# Show plot
plt.show()
```

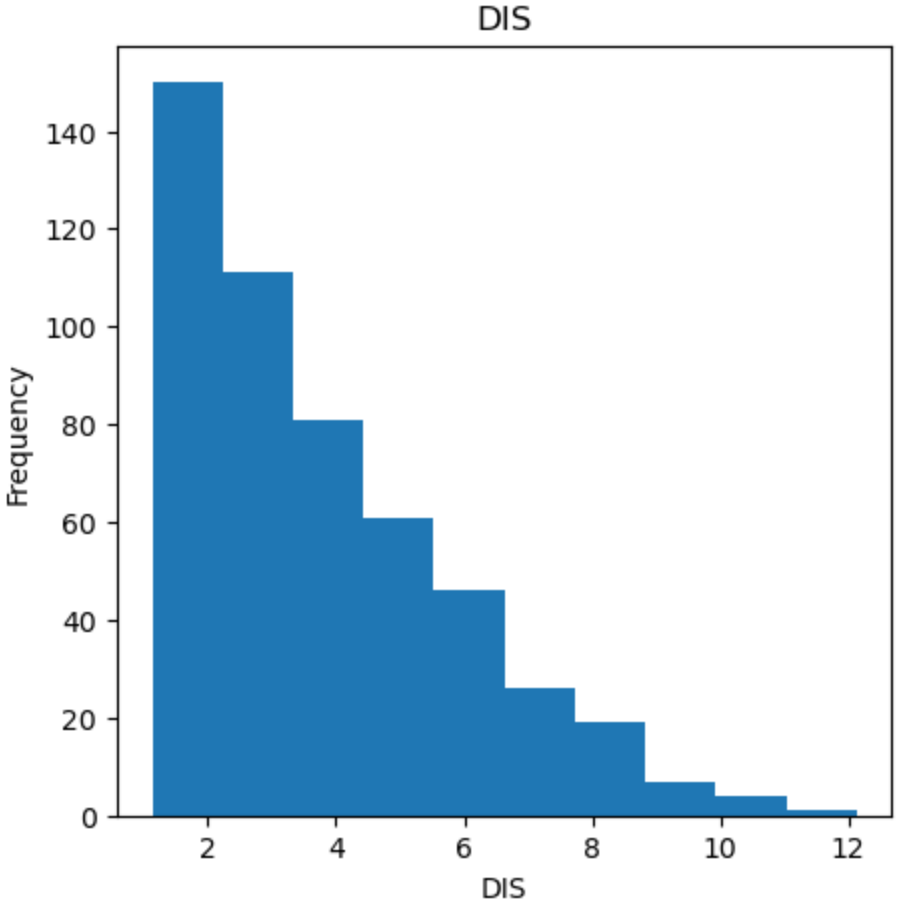

CRIM
Frequency
0
100
200
300
400
0
20
40
60
80
CRIM

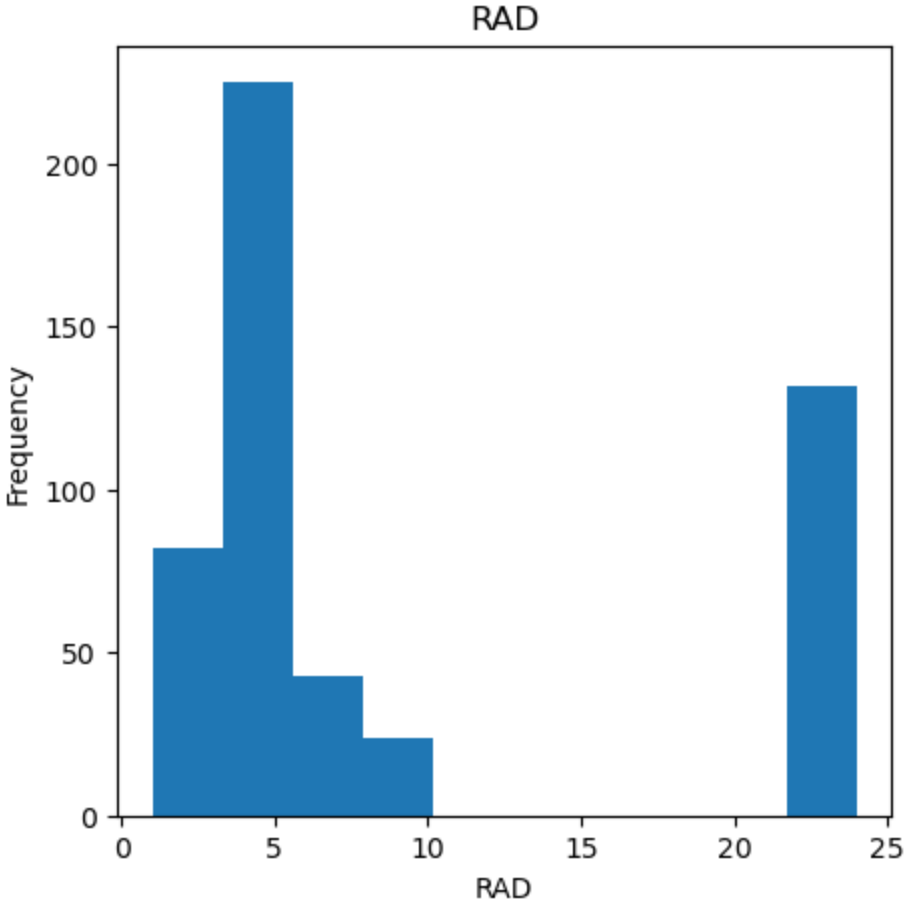

ZN
Frequency
0
50
100
150
200
250
300
350
0
20
40
60
80
100
ZN

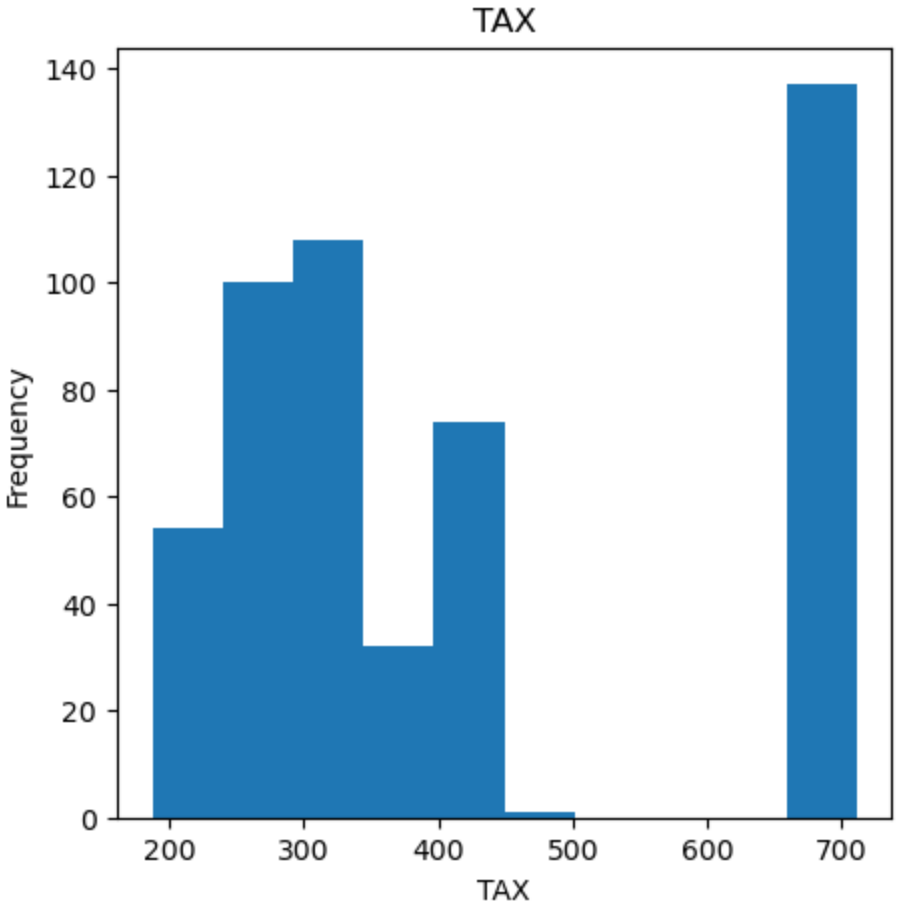

INDUS
Frequency
0
20
40
60
80
100
120
0
5
10
15
20
25
INDUS

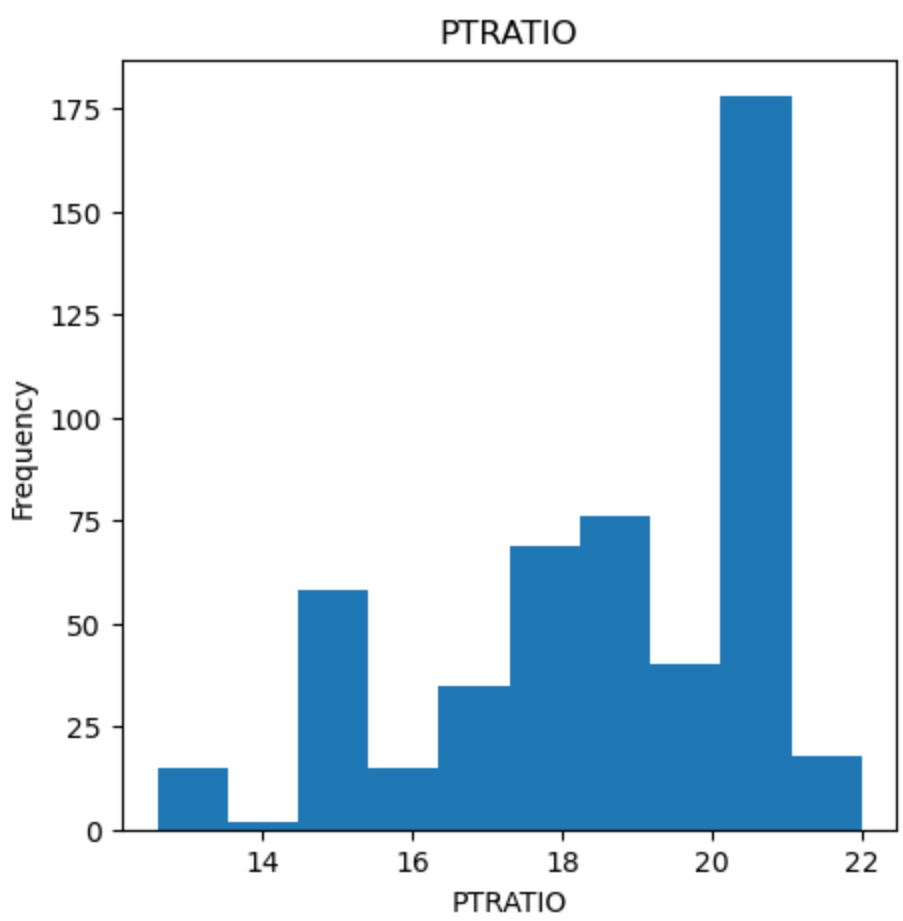

RM
Frequency
0
25
50
75
100
125
150
175
4
5
6
7
8
9
RM

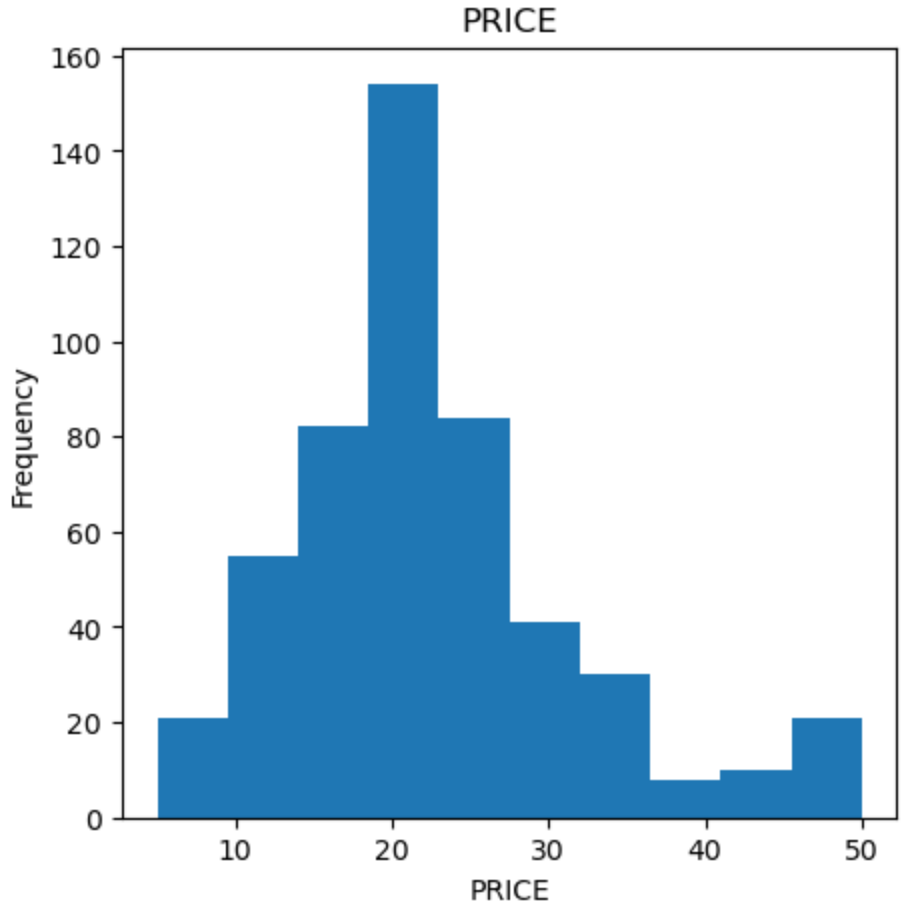

AGE
Frequency
0
20
40
60
80
100
120
140
160
0
20
40
60
80
100
AGE


DIS
Frequency
0
20
40
60
80
100
120
140
2
4
6
8
10
12
DIS


RAD
Frequency
0
50
100
150
200
0
5
10
15
20
25
RAD


TAX
Frequency
0
20
40
60
80
100
120
140
200
300
400
500
600
700
TAX

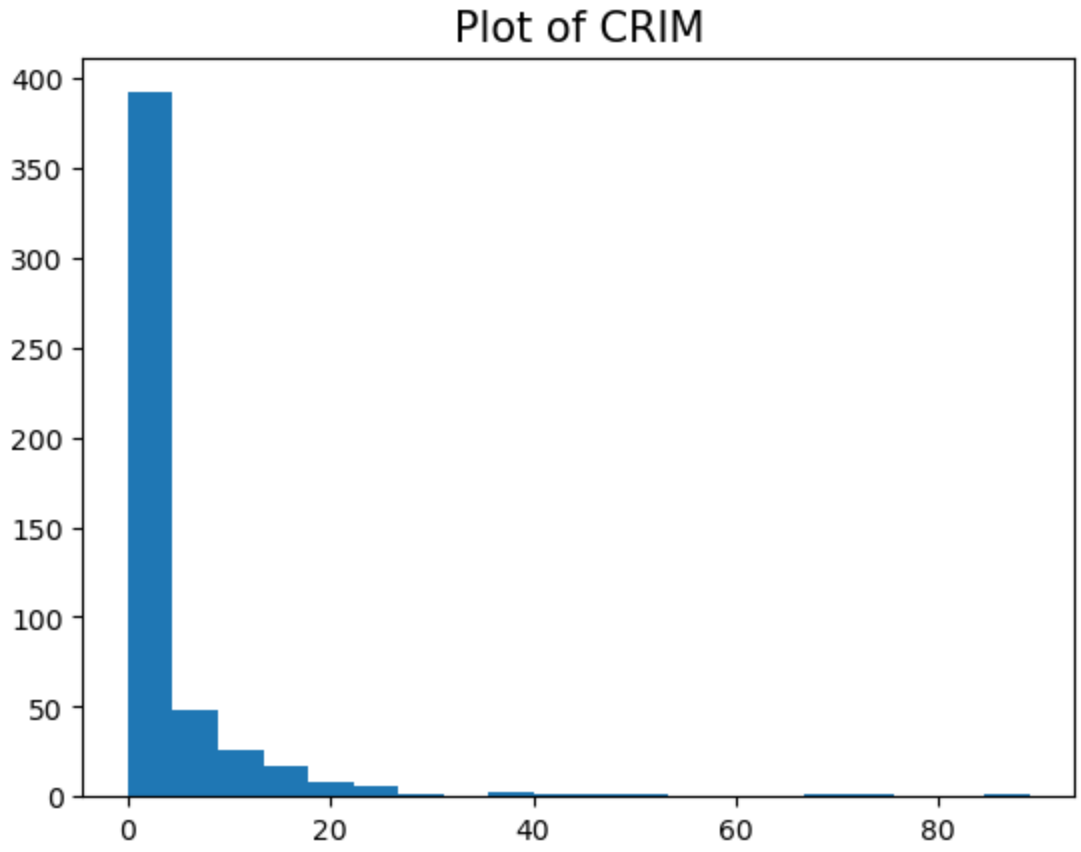


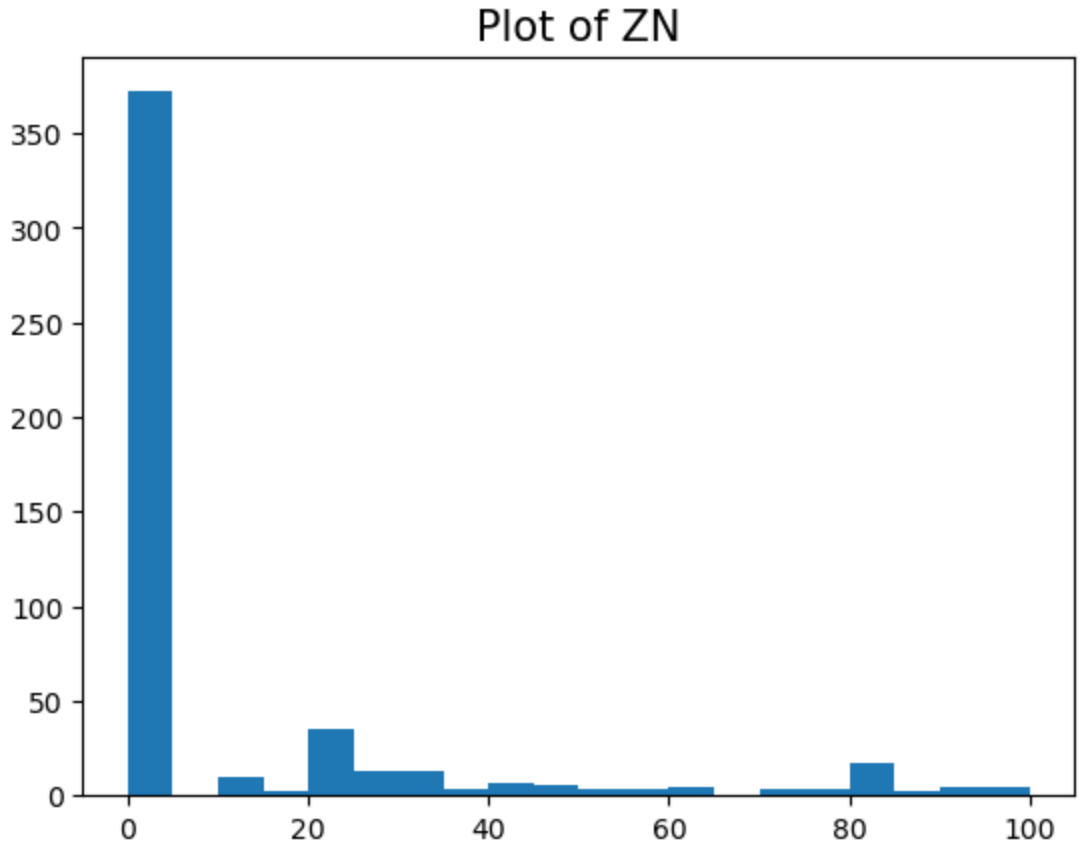


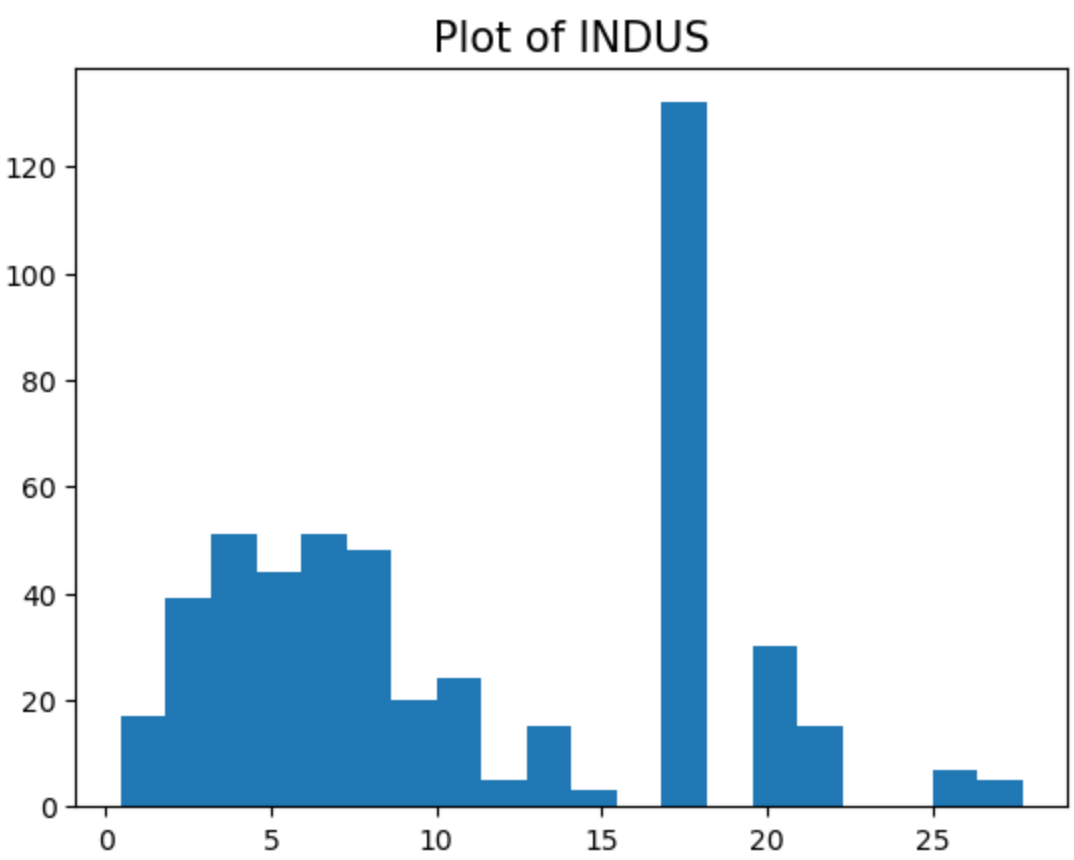
7. Plot them all at once using a for loop. Try to add a unique title to a plot.

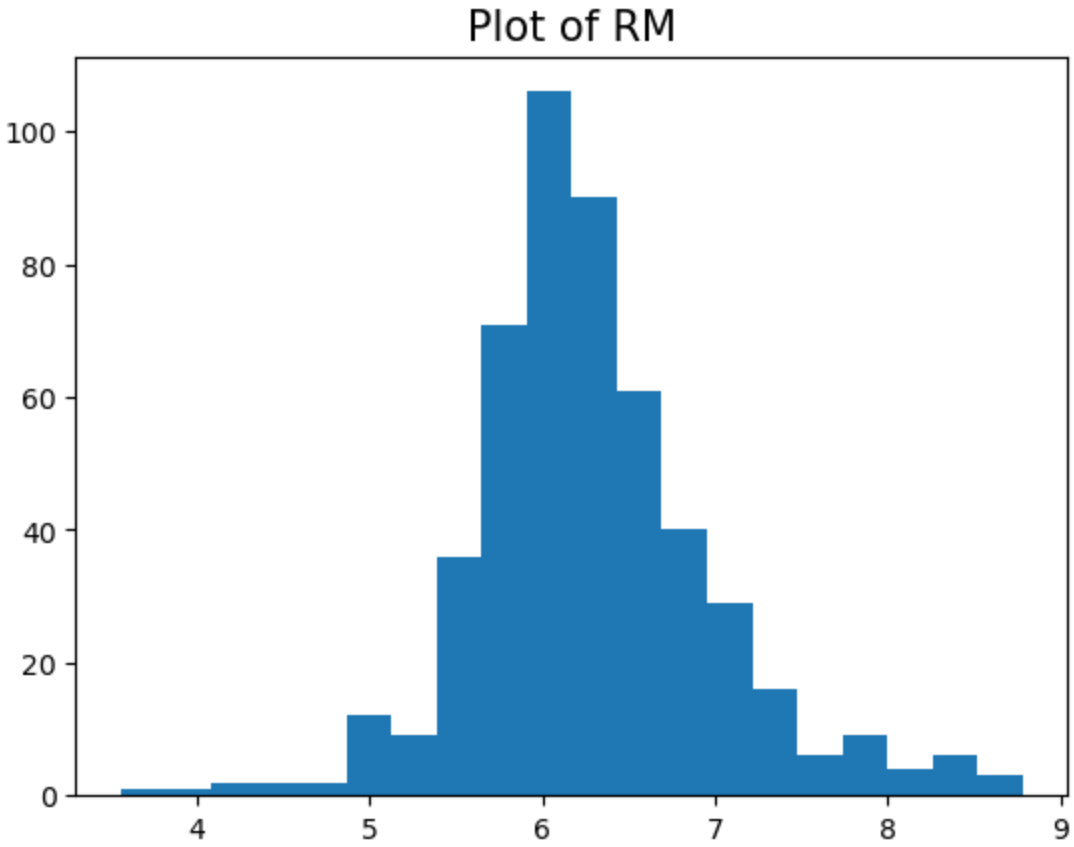
In [7]:
```
#fig, axes = plt.subplots(ncols=len(boston_housing_df.columns), figsize=(10,5))
for col,row in boston_housing_df_2.items():
    plt.hist(boston_housing_df_2[col],bins=20)
```

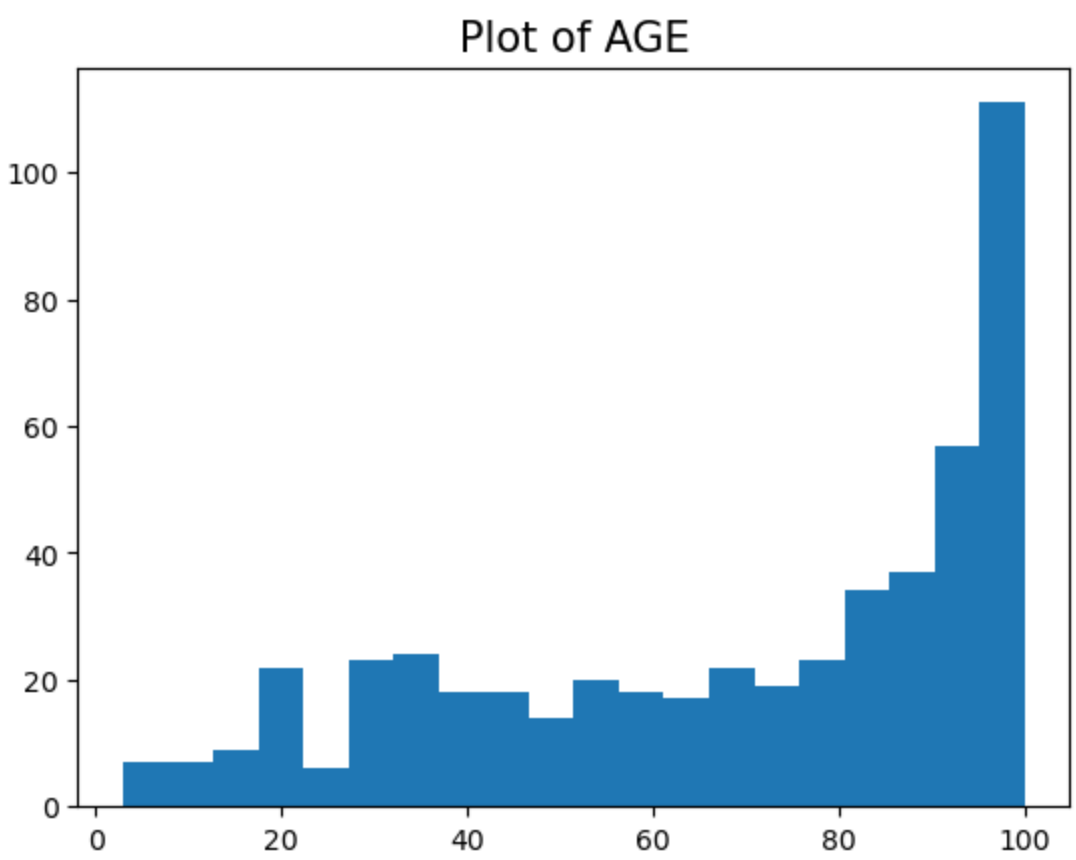
```
    plt.title("Plot of "+ col,fontsize=15)
    plt.show()
```

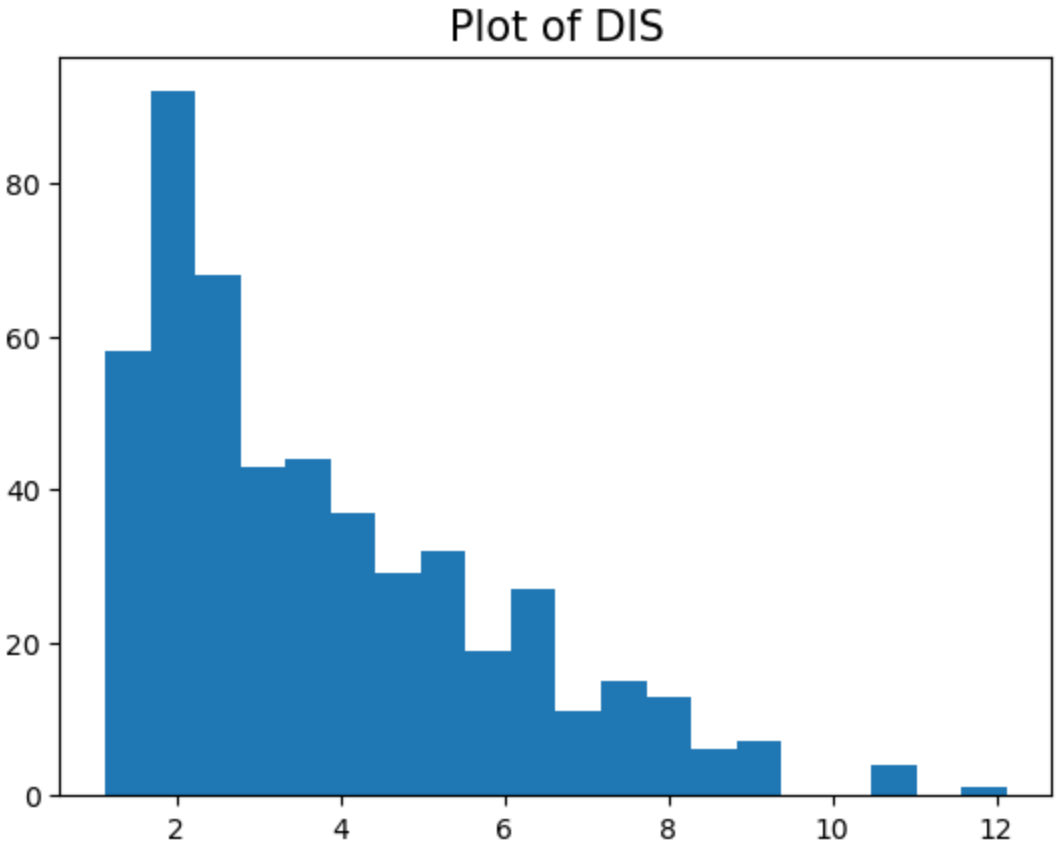


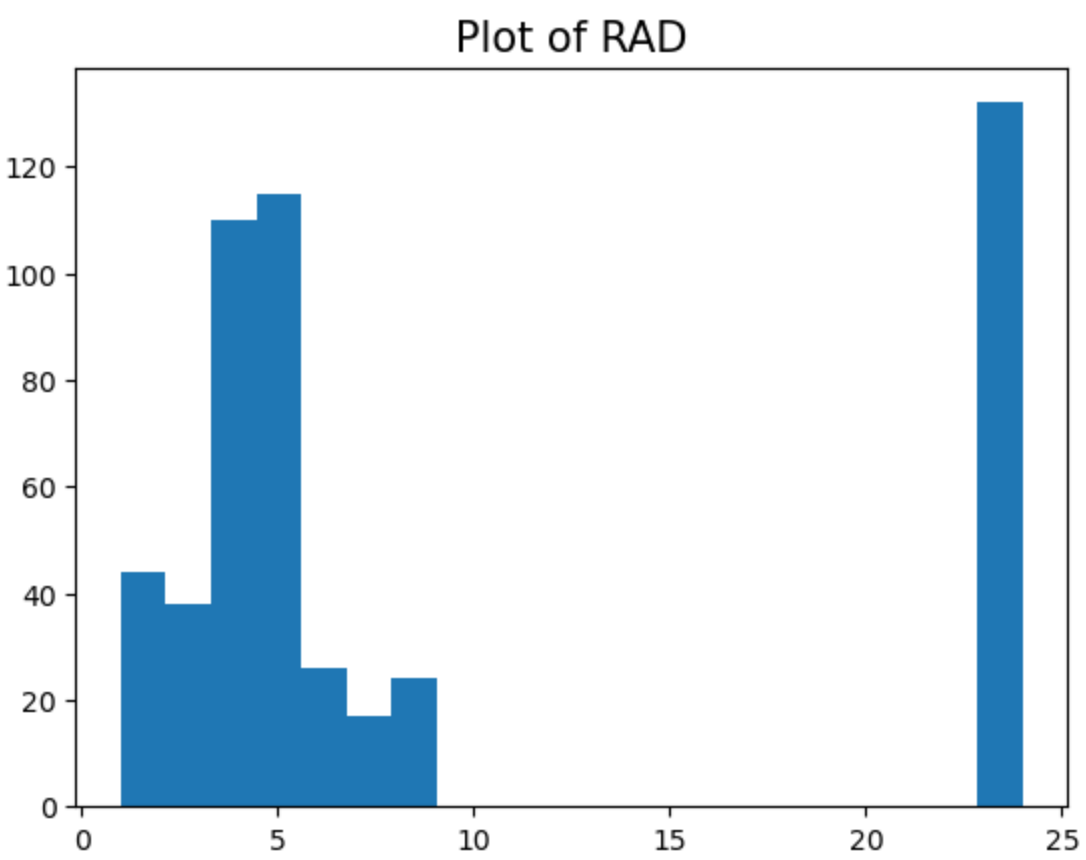


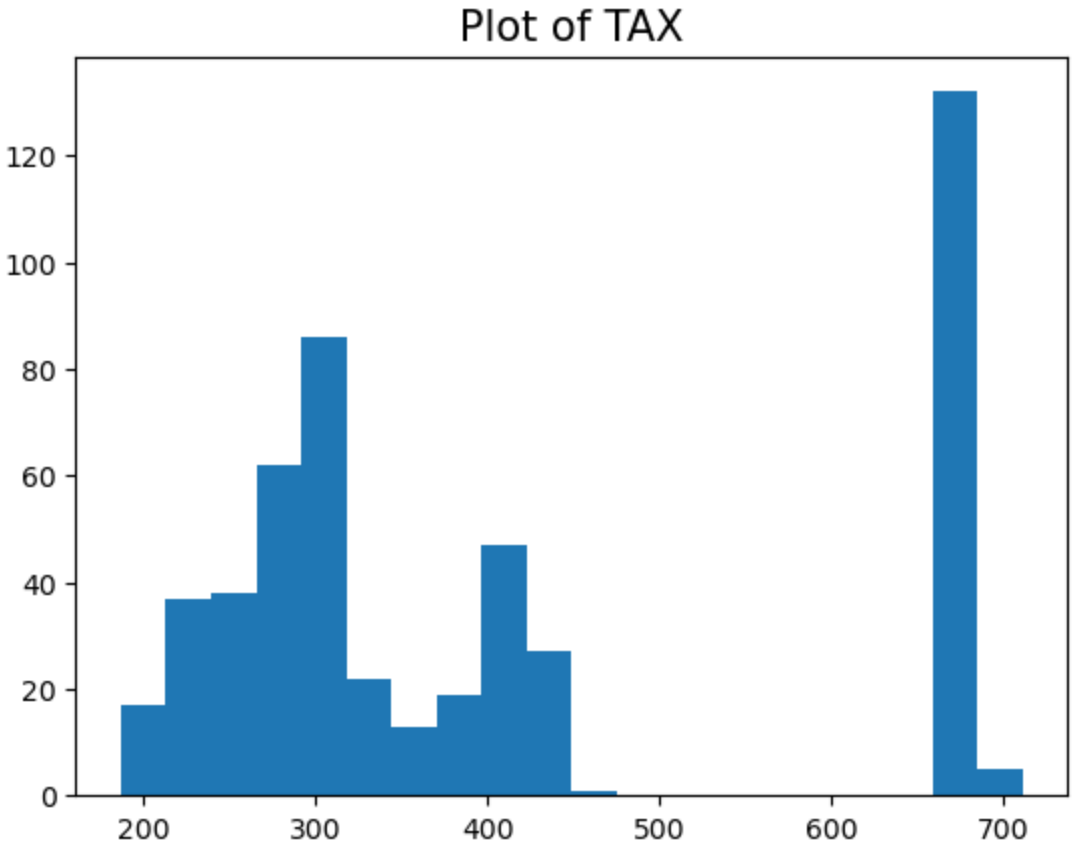

Plot of INDUS
120
100
80
60
40
20
0
0
5
10
15
20
25

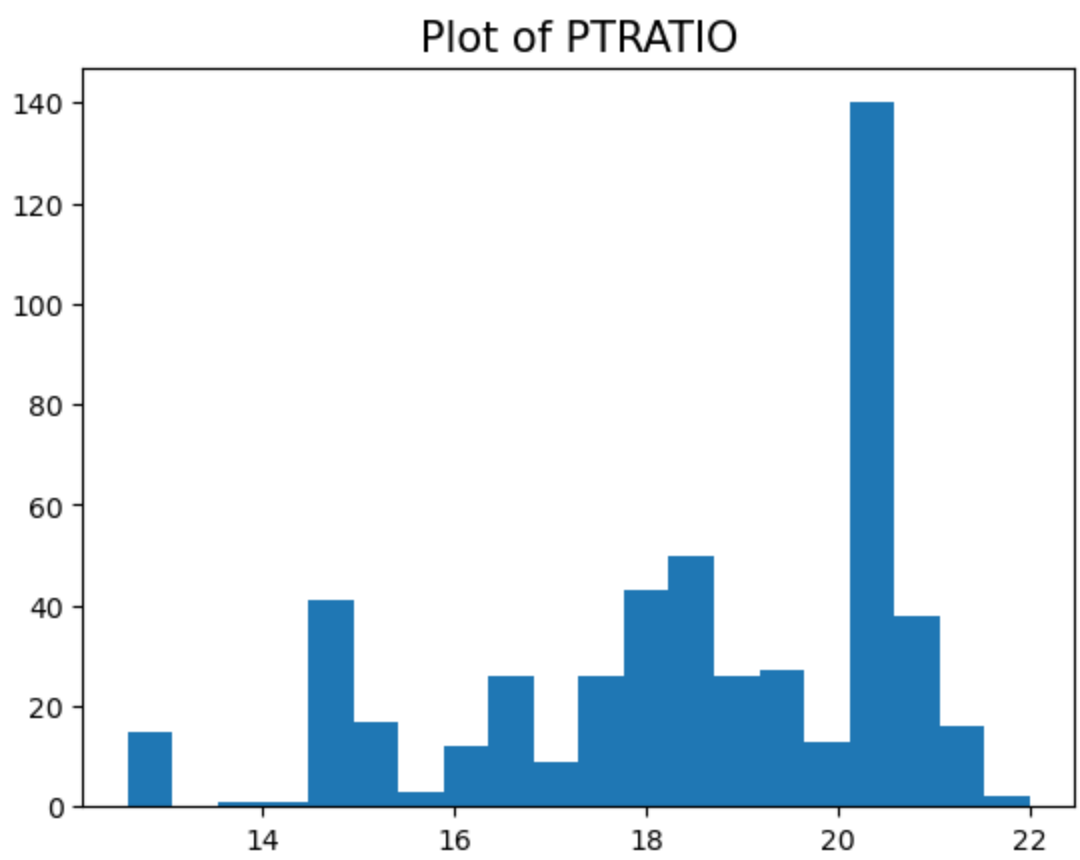

Plot of RM
100
80
60
40
20
0
4
5
6
7
8
9

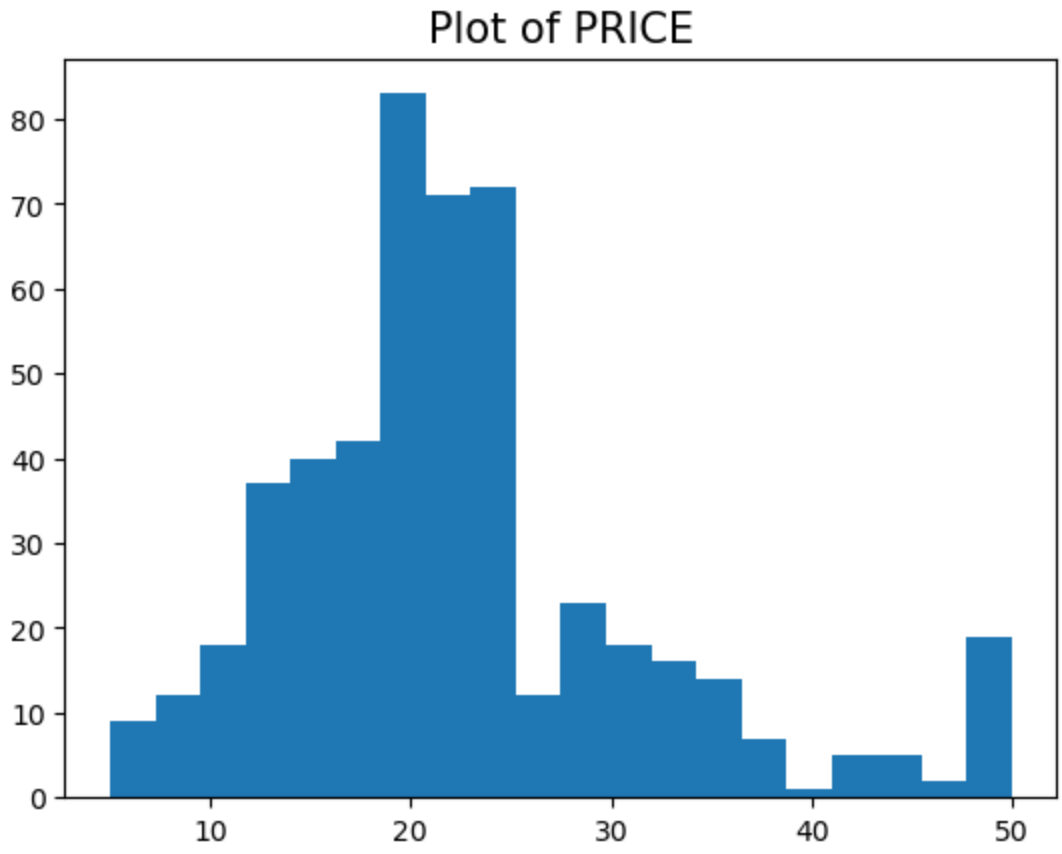

Plot of AGE
100
80
60
40
20
0
0
20
40
60
80
100


Plot of DIS
80
60
40
20
0
2
4
6
8
10
12


Plot of RAD
120
100
80
60
40
20
0
0
5
10
15
20
25


Plot of TAX
120
100
80
60
40
20
0
200
300
400
500
600
700

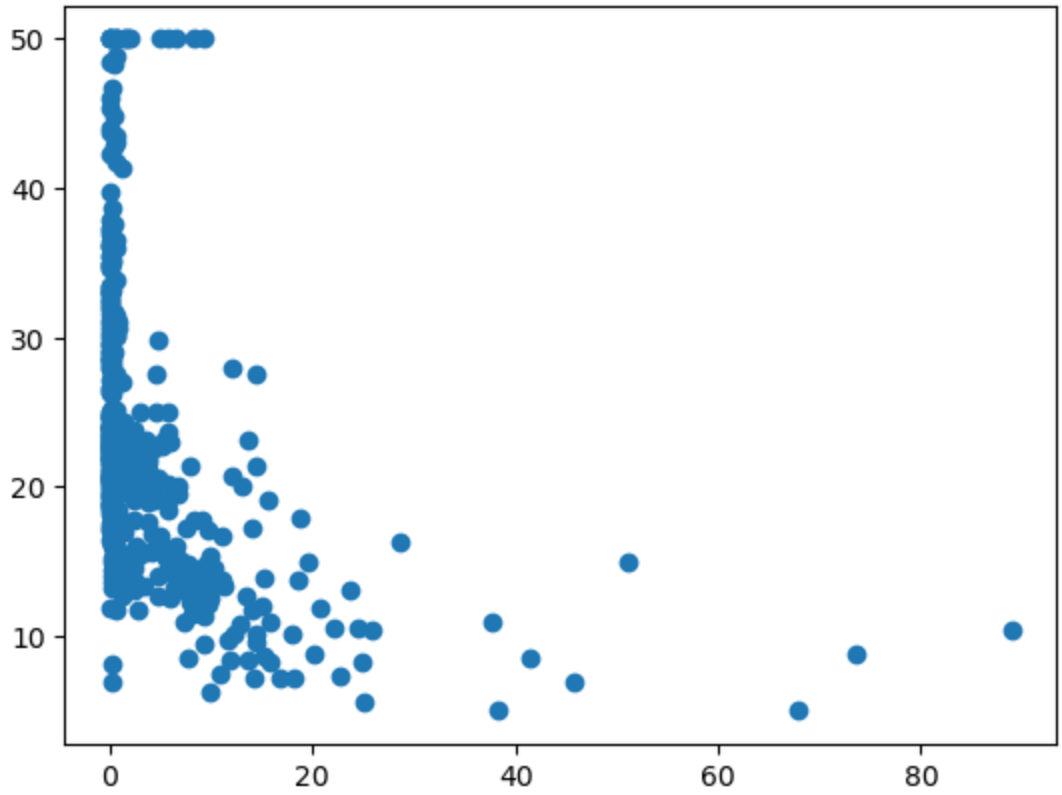




8. Create a scatter plot of crime rate versus price.

In [8]:
```
plt.scatter(boston_housing_df.CRIM, boston_housing_df.PRICE)
```

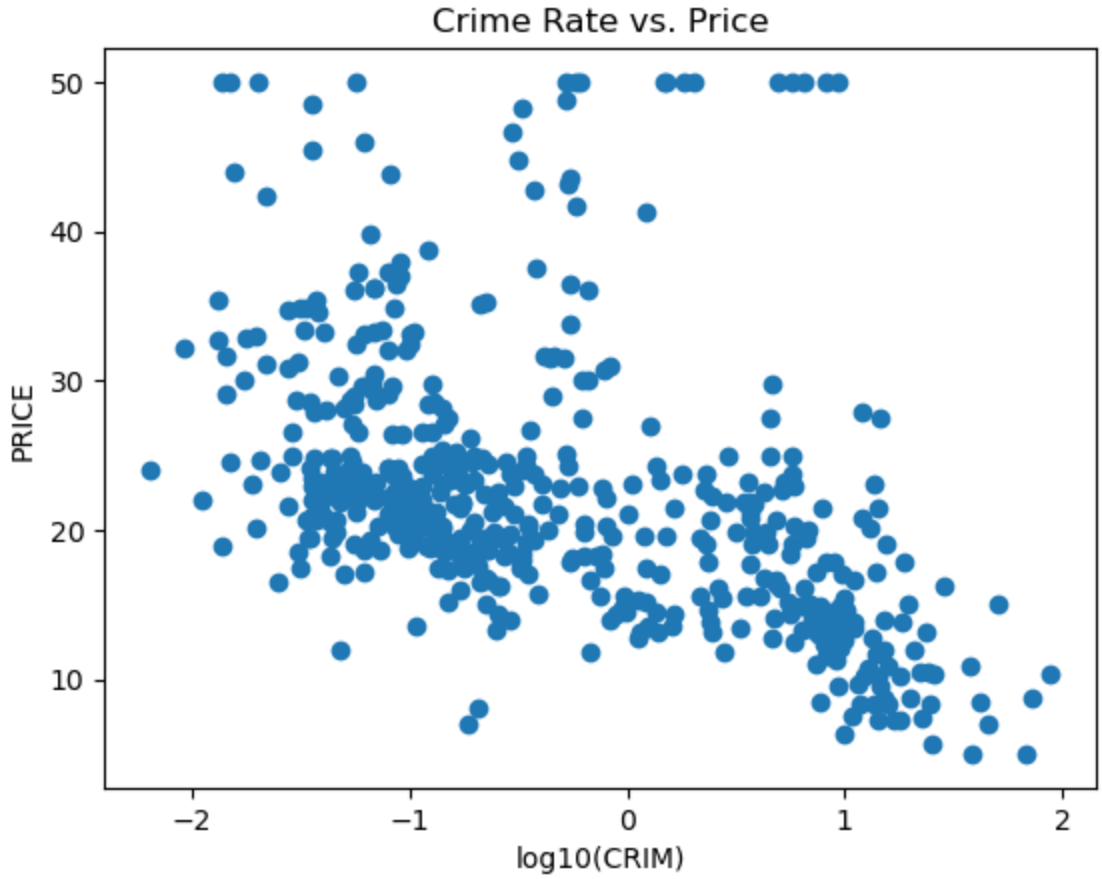
Out[8]:
```
<matplotlib.collections.PathCollection at 0x20e862791c0>
```



9. Plot using log10(crime) versus price.

In [9]:
```
import numpy as np

plt.scatter(np.log10(boston_housing_df.CRIM),boston_housing_df.PRICE )
plt.title("Crime Rate vs. Price")
plt.xlabel("log10(CRIM)")
plt.ylabel("PRICE")
```

Out[9]:
```
Text(0, 0.5, 'PRICE')
```



10. Calculate some useful statistics, such as mean rooms per dwelling, median age, mean distances to five Boston employment centers, and the percentage of houses with a low price (< $20,000).

In [10]:
```
print('Mean rooms:',boston_housing_df.RM.mean())
print('Median Age:',boston_housing_df.AGE.median())
print('Mean Distance:',boston_housing_df.DIS.mean())

print('Number of houses with price < $20,000:',len(boston_housing_df[boston_housing_df.P
print('Total number of houses:',len(boston_housing_df.PRICE))
print('Percentage of houses with low price:',round(len(boston_housing_df[boston_housing_
```

```
Mean rooms: 6.284634387351779
Median Age: 77.5
Mean Distance: 3.795042687747036
Number of houses with price < $20,000: 210
Total number of houses: 506
Percentage of houses with low price: 41.5 %
```

Activity 6: Generating Statistics from a CSV File

1. Load the necessary libraries.

In [11]:
```
import os.path
```

2. Read the adult income dataset from the following URL: https://github.com/TrainingByPackt/Data-Wrangling-with-Python/blob/master/Chapter04/Activity06/.

In [12]:
```
adult_income_df = pd.read_csv('https://raw.githubusercontent.com/TrainingByPackt/Data-Wr
adult_income_df
```

39	**State-**	**77516**	**Bachelors**	**13**	**Never-**	**Adm-**	**Not-in-**	**Male**	**2174**	**0**	**40**	**United-**	**<=5**

Out[12]:

		gov				married	clerical	family					States	
0	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Male	0	0	13	United-States	<=5
1	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	Male	0	0	40	United-States	<=5
2	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Male	0	0	40	United-States	<=5
3	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Female	0	0	40	Cuba	<=5
4	37	Private	284582	Masters	14	Married-civ-spouse	Exec-managerial	Wife	Female	0	0	40	United-States	<=5
...	...	...	...	...	...	...	...	...	...	...	...	...	...	
32555	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	Female	0	0	38	United-States	<=5
32556	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	Male	0	0	40	United-States	>5
32557	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	Female	0	0	40	United-States	<=5
32558	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	Male	0	0	20	United-States	<=5
32559	52	Self-emp-inc	287927	HS-grad	9	Married-civ-spouse	Exec-managerial	Wife	Female	15024	0	40	United-States	>5

32560 rows × 14 columns

3. Create a script that will read a text file line by line.

In [13]:
```
lines=[]
with open('adult_income_names.txt', 'r') as file_data:  # using with to handle file clos
    for line in file_data:
        file_data.readline() #Read each line
        lines.append(line) #Adding each line to a list
lines
```

Out[13]:
```
['age: continuous.\n',
 'workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov,
Without-pay, Never-worked.\n',
 'fnlwgt: continuous.\n',
 'education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm, Assoc-voc,
9th, 7th-8th, 12th, Masters, 1st-4th, 10th, Doctorate, 5th-6th, Preschool.\n',
 'education-num: continuous.\n',
 'marital-status: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Marri
ed-spouse-absent, Married-AF-spouse.\n',
 'occupation: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-sp
ecialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-
moving, Priv-house-serv, Protective-serv, Armed-Forces.\n',
 'relationship: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.\n',
```

```
 'sex: Female, Male.\n',
 'capital-gain: continuous.\n',
 'capital-loss: continuous.\n',
 'hours-per-week: continuous.\n',
 'native-country: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlyi
ng-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippi
nes, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Repub
lic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Th
ailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.']
```

4. Add a name of Income for the response variable to the dataset.

In [14]:
```
nameOfIncome = [lines[line].split(':')[0] for line in range(len(lines))] #Split string u
nameOfIncome.append('income') #Adding column 'income'
nameOfIncome
```

Out[14]:
```
['age',
 'workclass',
 'fnlwgt',
 'education',
 'education-num',
 'marital-status',
 'occupation',
 'relationship',
 'sex',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'native-country',
 'income']
```

5. Find the missing values.

In [15]:
```
adult_income_df.isnull()
```

Out[15]:

	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	Male	2174	0	40	United-States	<=50K
0	False	False	False	False	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
32555	False	False	False	False	False	False	False	False	False	False	False	False	False	False
32556	False	False	False	False	False	False	False	False	False	False	False	False	False	False
32557	False	False	False	False	False	False	False	False	False	False	False	False	False	False
32558	False	False	False	False	False	False	False	False	False	False	False	False	False	False
32559	False	False	False	False	False	False	False	False	False	False	False	False	False	False

32560 rows × 14 columns

In [16]:
```
for cols in adult_income_df.columns:
    missing = adult_income_df[cols].isnull().sum() #Get the count of missing values by c
    if missing>0:
```

```
        print("{} has {} missing value(s)".format(cols,missing)) #Print column name and
    else:
        print("{} has NO missing value!".format(cols)) #Print column name with no missin
```

```
39 has NO missing value!
 State-gov has NO missing value!
77516 has NO missing value!
 Bachelors has NO missing value!
13 has NO missing value!
 Never-married has NO missing value!
 Adm-clerical has NO missing value!
 Not-in-family has NO missing value!
 Male has NO missing value!
2174 has NO missing value!
0 has NO missing value!
40 has NO missing value!
 United-States has NO missing value!
 <=50K has NO missing value!
```

In [17]:
```
# Alternatively, converting list to dict. Identify missing values using pd.isna for ever
#Replacing new line with empty string and splitting string by ':' for every list item (l
incomenames = dict(lines[line].replace('\n','').split(':') for line in range(len(lines))
for key, value in incomenames.items():
    if pd.isna(key):
        print(key)
    if pd.isna(value):
        print(value)
#Nothing is printed implying there are no missing values.
```

In [18]:
```
adult_income_df.isna().sum() #Another method of identifying missing values for the entir
```

Out[18]:
```
39                0
 State-gov        0
77516             0
 Bachelors        0
13                0
 Never-married    0
 Adm-clerical     0
 Not-in-family    0
 Male             0
2174              0
0                 0
40                0
 United-States    0
 <=50K            0
dtype: int64
```

6. Create a DataFrame with only age, education, and occupation by using subsetting.

In [19]:
```
new_income_df = pd.read_csv('https://raw.githubusercontent.com/TrainingByPackt/Data-Wran
```

In [20]:
```
income_df = new_income_df[['age','education','occupation']]
income_df
```

Out[20]:

	age	education	occupation
0	39	Bachelors	Adm-clerical
1	50	Bachelors	Exec-managerial
2	38	HS-grad	Handlers-cleaners
3	53	11th	Handlers-cleaners
4	28	Bachelors	Prof-specialty

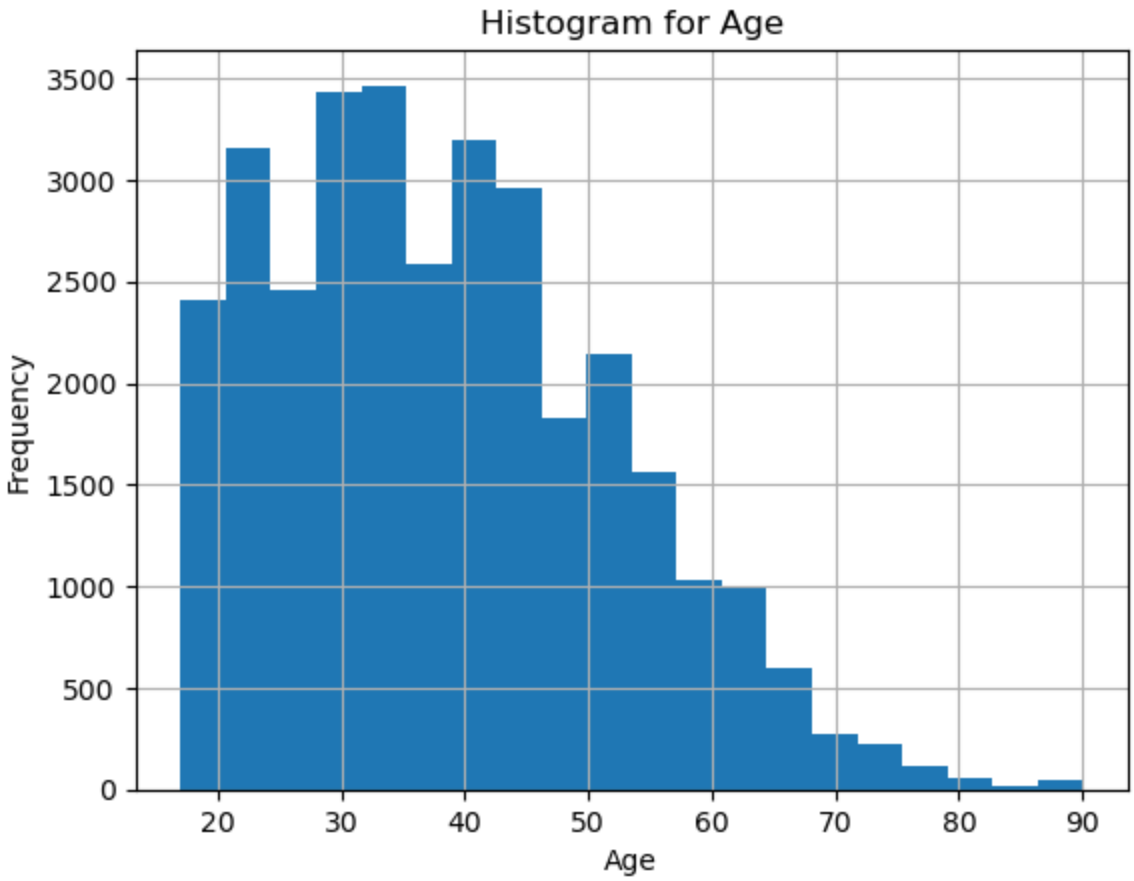
...	...	...	...
32556	27	Assoc-acdm	Tech-support
32557	40	HS-grad	Machine-op-inspct
32558	58	HS-grad	Adm-clerical
32559	22	HS-grad	Adm-clerical
32560	52	HS-grad	Exec-managerial

32561 rows × 3 columns

7. Plot a histogram of age with a bin size of 20.

In [34]:
```
income_df.age.hist(bins=20)
plt.xlabel("Age")
plt.ylabel("Frequency")
plt.title("Histogram for Age")
```

Out[34]:
```
Text(0.5, 1.0, 'Histogram for Age')
```



8. Create a function to strip the whitespace characters.

In [35]:
```
def StripWhiteSpaces(text):
    str = ''
    if pd.notna(text):
        str = text.strip()
    return str
```

9. Use the apply method to apply this function to all the columns with string values,create a new column, copy the values from this new column to the old column,and drop the new column.

In [36]:
```
income_df['education'] = income_df['education'].apply(StripWhiteSpaces)
income_df['occupation'] = income_df['occupation'].apply(StripWhiteSpaces)
income_df
```

```
C:\Users\aarti\AppData\Local\Temp\ipykernel_39372\3363460757.py:1: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  income_df['education'] = income_df['education'].apply(StripWhiteSpaces)
C:\Users\aarti\AppData\Local\Temp\ipykernel_39372\3363460757.py:2: SettingWithCopyWarnin
g:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_
guide/indexing.html#returning-a-view-versus-a-copy
  income_df['occupation'] = income_df['occupation'].apply(StripWhiteSpaces)
```

Out[36]:

	age	education	occupation
0	39	Bachelors	Adm-clerical
1	50	Bachelors	Exec-managerial
2	38	HS-grad	Handlers-cleaners
3	53	11th	Handlers-cleaners
4	28	Bachelors	Prof-specialty
...	...	...	...
32556	27	Assoc-acdm	Tech-support
32557	40	HS-grad	Machine-op-inspct
32558	58	HS-grad	Adm-clerical
32559	22	HS-grad	Adm-clerical
32560	52	HS-grad	Exec-managerial

32561 rows × 3 columns

10. Find the number of people who are aged between 30 and 50.

In [24]:
```
income_df[(income_df.age >= 30) & (income_df.age <= 50)]
```

Out[24]:

	age	education	occupation
0	39	Bachelors	Adm-clerical
1	50	Bachelors	Exec-managerial
2	38	HS-grad	Handlers-cleaners
5	37	Masters	Exec-managerial
6	49	9th	Other-service
...	...	...	...
32550	43	Some-college	Craft-repair
32551	32	10th	Handlers-cleaners

32552	43	Assoc-voc	Sales
32553	32	Masters	Tech-support
32557	40	HS-grad	Machine-op-inspct

16390 rows × 3 columns

11. Group the records based on age and education to find how the mean age is distributed. 172 | A Deep Dive into Data Wrangling with Python

In [25]:
```
income_df.groupby(['education']).mean(numeric_only=True)
```

Out[25]:

education	age
10th	37.429796
11th	32.355745
12th	32.000000
1st-4th	46.142857
5th-6th	42.885886
7th-8th	48.445820
9th	41.060311
Assoc-acdm	37.381443
Assoc-voc	38.553546
Bachelors	38.904949
Doctorate	47.702179
HS-grad	38.974479
Masters	44.049913
Preschool	42.764706
Prof-school	44.746528
Some-college	35.756275

12. Group by occupation and show the summary statistics of age. Find which profession has the oldest workers on average and which profession has its largest share of the workforce above the 75th percentile.

In [26]:
```
income_df.groupby(['occupation']).describe()['age']
```

Out[26]:

occupation	count	mean	std	min	25%	50%	75%	max
?	1843.0	40.882800	20.336350	17.0	21.0	35.0	61.0	90.0
Adm-clerical	3770.0	36.964456	13.362998	17.0	26.0	35.0	46.0	90.0
Armed-Forces	9.0	30.222222	8.089774	23.0	24.0	29.0	34.0	46.0
Craft-repair	4099.0	39.031471	11.606436	17.0	30.0	38.0	47.0	90.0

Exec-managerial	4066.0	42.169208	11.974548	17.0	33.0	41.0	50.0	90.0
Farming-fishing	994.0	41.211268	15.070283	17.0	29.0	39.0	52.0	90.0
Handlers-cleaners	1370.0	32.165693	12.372635	17.0	23.0	29.0	39.0	90.0
Machine-op-inspct	2002.0	37.715285	12.068266	17.0	28.0	36.0	46.0	90.0
Other-service	3295.0	34.949621	14.521508	17.0	22.0	32.0	45.0	90.0
Priv-house-serv	149.0	41.724832	18.633688	17.0	24.0	40.0	57.0	81.0
Prof-specialty	4140.0	40.517633	12.016676	17.0	31.0	40.0	48.0	90.0
Protective-serv	649.0	38.953775	12.822062	17.0	29.0	36.0	47.0	90.0
Sales	3650.0	37.353973	14.186352	17.0	25.0	35.0	47.0	90.0
Tech-support	928.0	37.022629	11.316594	17.0	28.0	36.0	44.0	73.0
Transport-moving	1597.0	40.197871	12.450792	17.0	30.0	39.0	49.0	90.0

i. Exec-managerial has the Oldest workers on Average. The mean is 42.16 which is the highest in the dataset.

ii. No profession has its workshare above 75th percentile. Craft-repair has a larger number but is not more than 75th percentile.

13. Use subset and groupby to find outliers.

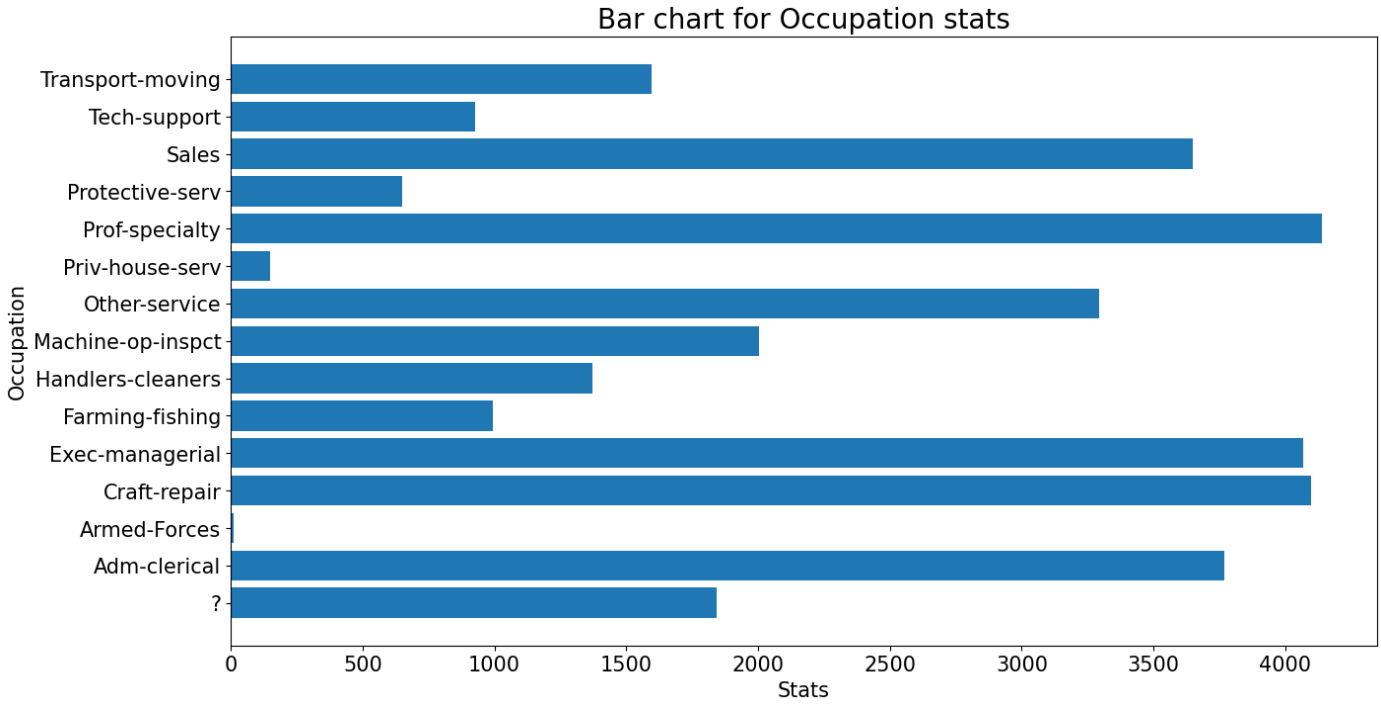
In [27]:
```
occupation_stats= income_df.groupby('occupation').describe()['age']
occupation_stats
```

Out[27]:

occupation	count	mean	std	min	25%	50%	75%	max
?	1843.0	40.882800	20.336350	17.0	21.0	35.0	61.0	90.0
Adm-clerical	3770.0	36.964456	13.362998	17.0	26.0	35.0	46.0	90.0
Armed-Forces	9.0	30.222222	8.089774	23.0	24.0	29.0	34.0	46.0
Craft-repair	4099.0	39.031471	11.606436	17.0	30.0	38.0	47.0	90.0
Exec-managerial	4066.0	42.169208	11.974548	17.0	33.0	41.0	50.0	90.0
Farming-fishing	994.0	41.211268	15.070283	17.0	29.0	39.0	52.0	90.0
Handlers-cleaners	1370.0	32.165693	12.372635	17.0	23.0	29.0	39.0	90.0
Machine-op-inspct	2002.0	37.715285	12.068266	17.0	28.0	36.0	46.0	90.0
Other-service	3295.0	34.949621	14.521508	17.0	22.0	32.0	45.0	90.0
Priv-house-serv	149.0	41.724832	18.633688	17.0	24.0	40.0	57.0	81.0
Prof-specialty	4140.0	40.517633	12.016676	17.0	31.0	40.0	48.0	90.0
Protective-serv	649.0	38.953775	12.822062	17.0	29.0	36.0	47.0	90.0
Sales	3650.0	37.353973	14.186352	17.0	25.0	35.0	47.0	90.0
Tech-support	928.0	37.022629	11.316594	17.0	28.0	36.0	44.0	73.0
Transport-moving	1597.0	40.197871	12.450792	17.0	30.0	39.0	49.0	90.0

14. Plot the values on a bar chart.

In [28]:
```
plt.figure(figsize=(15,8))
plt.barh(y=occupation_stats.index,width=occupation_stats['count'])
plt.title("Bar chart for Occupation stats",fontsize=20)
plt.ylabel("Occupation",fontsize=15)
plt.xlabel("Stats",fontsize=15)
plt.xticks(fontsize=15)
plt.yticks(fontsize=15)
plt.show()
```



15. Merge the data using common keys.

In [37]:
```
df1=new_income_df[['age','workclass','occupation']].sample(5,random_state=101)
df2=new_income_df[['education','occupation']].sample(5,random_state=101)
pd.merge(df1,df2,on='occupation',how='inner')
```

Out[37]:

	age	workclass	occupation	education
0	51	Private	Machine-op-inspct	HS-grad
1	19	Private	Sales	11th
2	40	Private	Exec-managerial	HS-grad
3	17	Private	Handlers-cleaners	10th
4	61	Private	Craft-repair	7th-8th

3. Create a series and practice basic arithmetic steps

a. Series 1 = 7.3, -2.5, 3.4, 1.5

i. Index = ‘a’, ‘c’, ‘d’, ‘e’

b. Series 2 = -2.1, 3.6, -1.5, 4, 3.1

i. Index = ‘a’, ‘c’, ‘e’, ‘f’, ‘g’

c. Add Series 1 and Series 2 together and print the results

d. Subtract Series 1 from Series 2 and print the results

In [30]:
```
first_series_data=np.array([7.3, -2.5, 3.4, 1.5])

first_series = pd.Series(first_series_data,index=['a', 'c', 'd', 'e'])
first_series
```

Out[30]:
```
a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64
```

In [31]:
```
second_series_data=np.array([-2.1, 3.6, -1.5, 4, 3.1])

second_series = pd.Series(second_series_data,index=['a', 'c', 'e', 'f', 'g'])
second_series
```

Out[31]:
```
a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

In [32]:
```
first_series.add(second_series,fill_value=0)
```

Out[32]:
```
a    5.2
c    1.1
d    3.4
e    0.0
f    4.0
g    3.1
dtype: float64
```

In [33]:
```
first_series.subtract(second_series,fill_value=0)
```

Out[33]:
```
a    9.4
c   -6.1
d    3.4
e    3.0
f   -4.0
g   -3.1
dtype: float64
```