Assignment_week_3 & 4_Raghuwanshi_Prashant_DSC540

September 23, 2021

Assignment: Week 3 & Week 4 Exercise, Understanding Packages

Name: Prashant Raghuwanshi

Date: 9/20/2021

Course: DSC540-T301 Data Preparation (2221-1) Data Wrangling with Python: Activity 5, page 116

[1]:
```
# Import libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[2]:
```
# read source file into dataframe
boston_house_df = pd.read_csv("C:/Users/dell/Documents/docker/Boston_housing.
 ↪csv")
# display firt 5 records
boston_house_df.head(5)
```

[2]:

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

	B	LSTAT	PRICE
0	396.90	4.98	24.0
1	396.90	9.14	21.6
2	392.83	4.03	34.7
3	394.63	2.94	33.4
4	396.90	5.33	36.2

[3]:
```
# display total records & cloumns counts
boston_house_df.shape
```

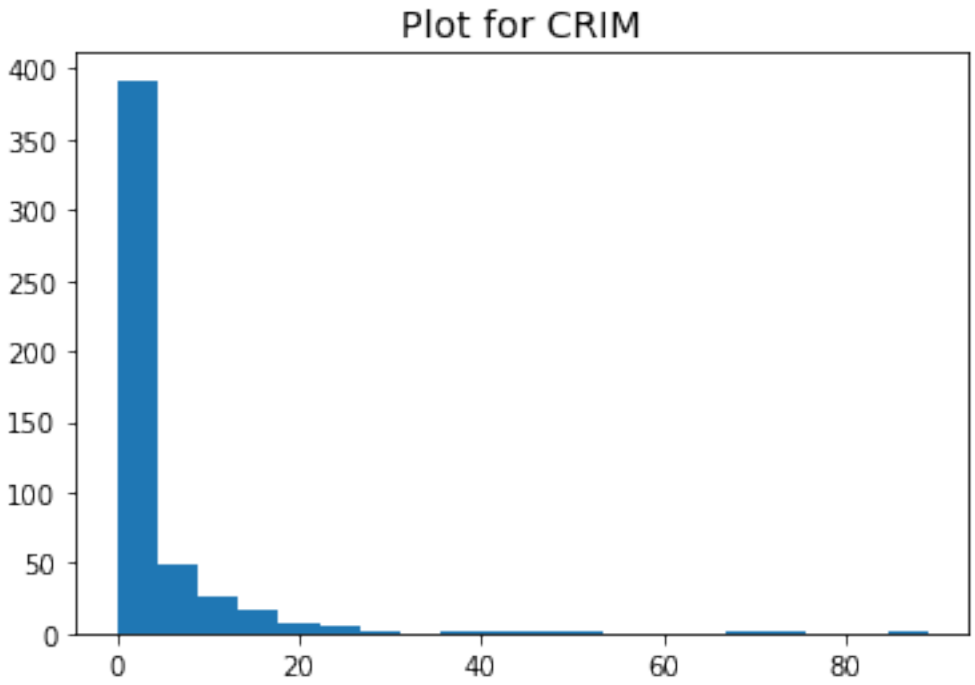
[3]: (506, 14)

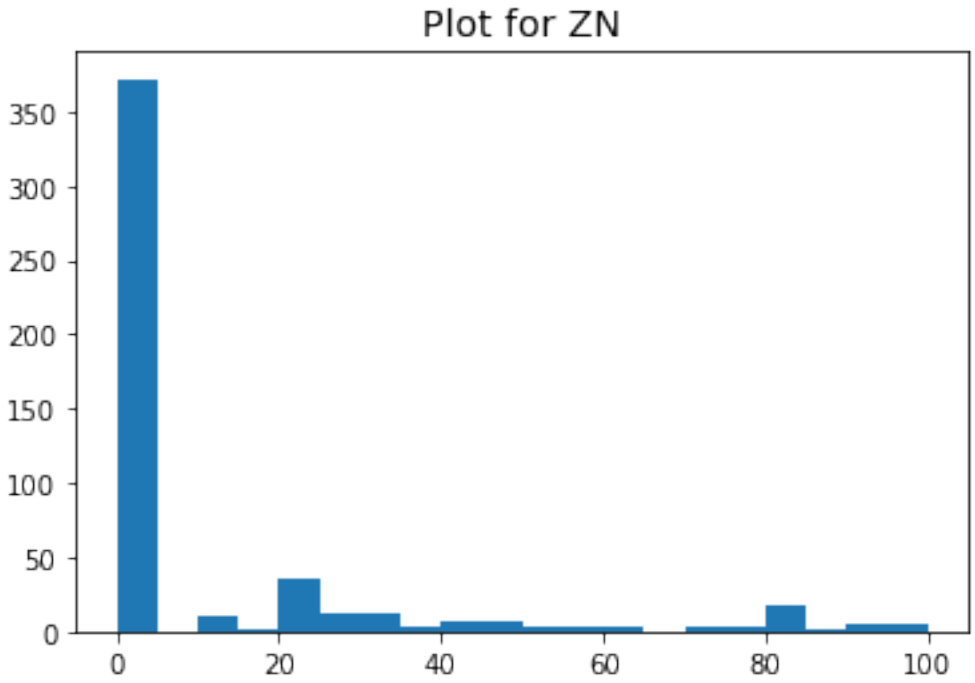
[4]:
```
# create new dataframe which do not include 'CHAS', 'NOX', 'B', and 'LSTAT'
# subsetting the datasets with required columns
boston_house_df1 =␣
 ↪boston_house_df[['CRIM','ZN','INDUS','RM','AGE','DIS','RAD','TAX','PTRATIO','PRICE']]
# display last 7 records from new dataframe
boston_house_df1.tail(7)
```

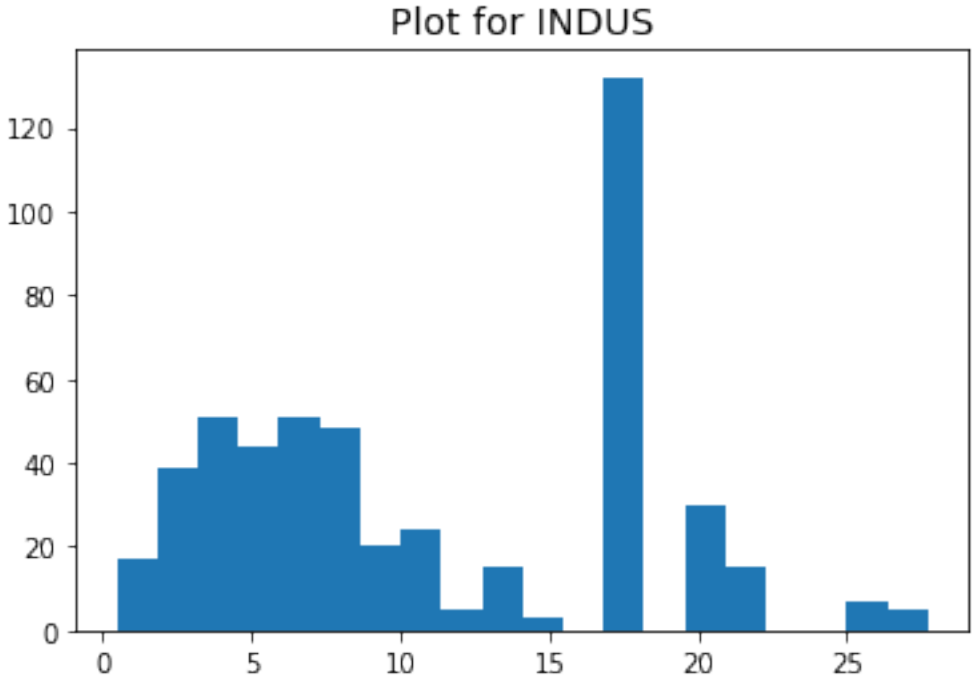
[4]:

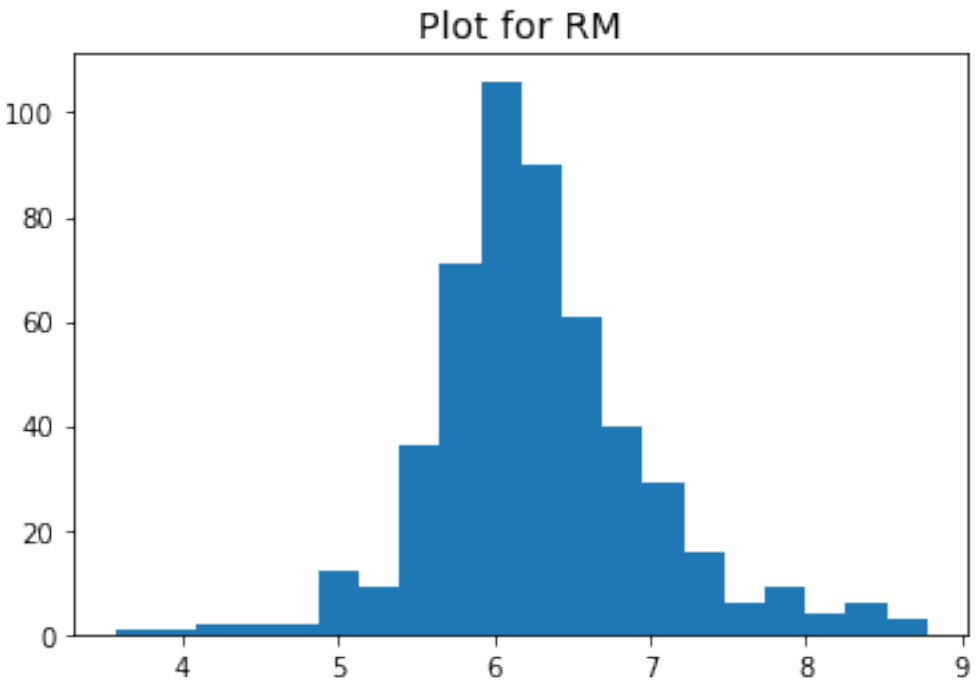
	CRIM	ZN	INDUS	RM	AGE	DIS	RAD	TAX	PTRATIO	PRICE
499	0.17783	0.0	9.69	5.569	73.5	2.3999	6	391	19.2	17.5
500	0.22438	0.0	9.69	6.027	79.7	2.4982	6	391	19.2	16.8
501	0.06263	0.0	11.93	6.593	69.1	2.4786	1	273	21.0	22.4
502	0.04527	0.0	11.93	6.120	76.7	2.2875	1	273	21.0	20.6
503	0.06076	0.0	11.93	6.976	91.0	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

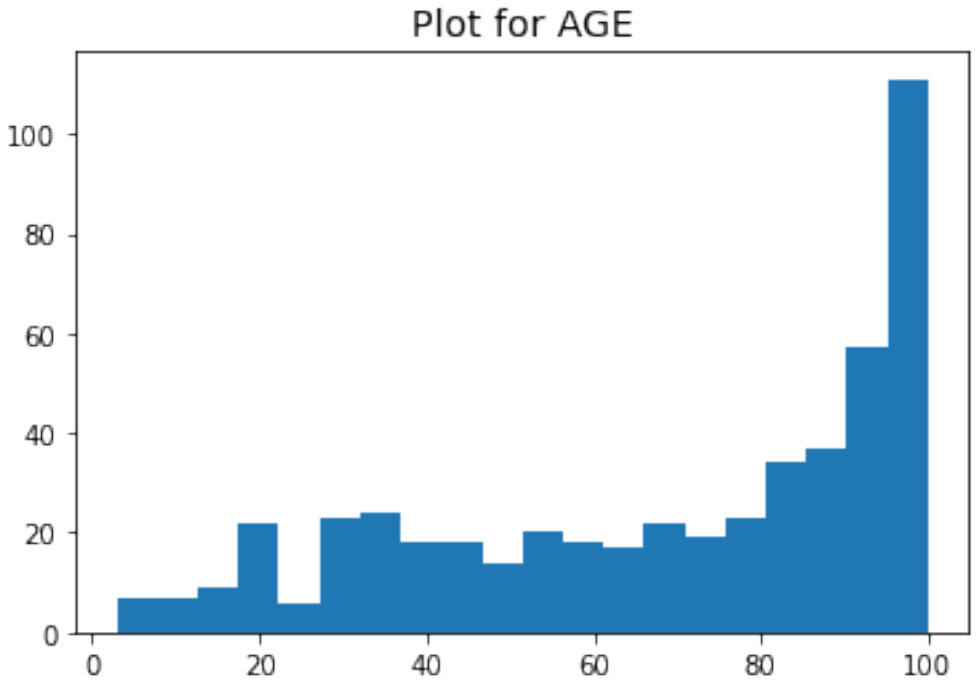
[5]:
```
# plot histograms of all the variables (columns) in the new DataFrame by uing a␣
 ↪for loop
# reading each column one by one in a for loop and plot the histogram graphs␣
 ↪for each columns
for col in boston_house_df1.columns:
    plt.title("Plot for "+col,fontsize=14)
    plt.hist(boston_house_df1[col],bins=20)
    # A histogram is a graph showing frequency distributions.
    # It is a graph showing the number of observations within each given␣
 ↪interval.
    plt.show()
```

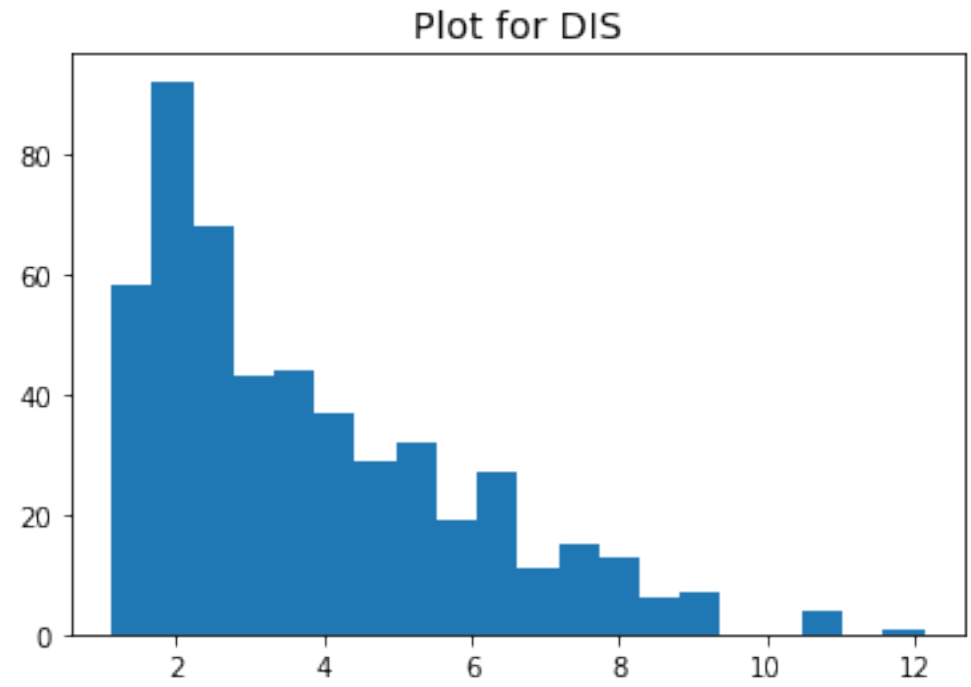


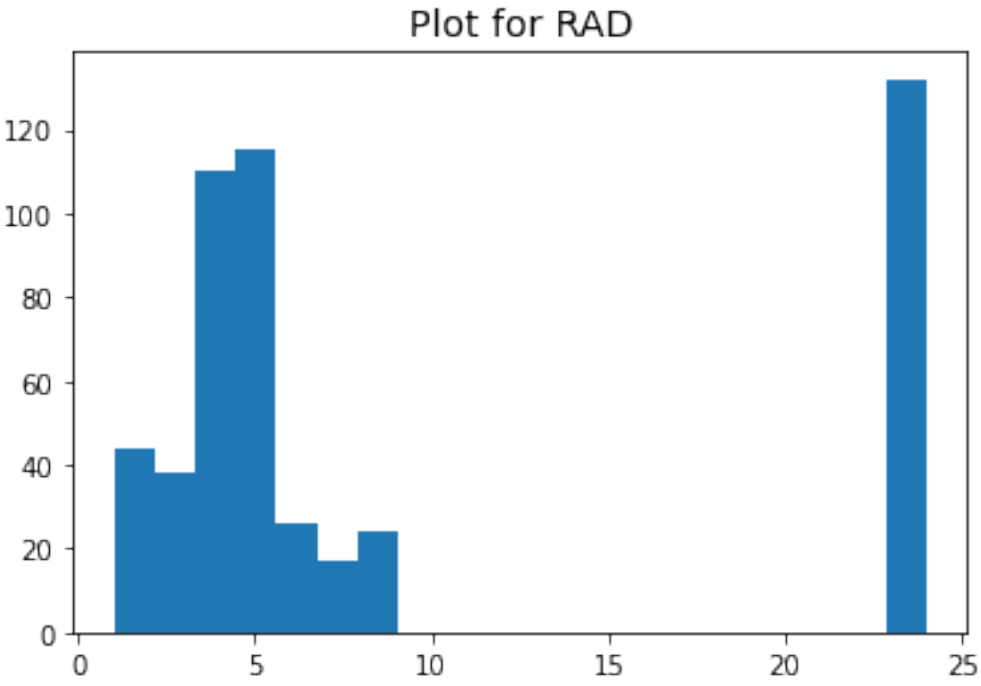

Plot for ZN
350
300
250
200
150
100
50
0
0
20
40
60
80
100

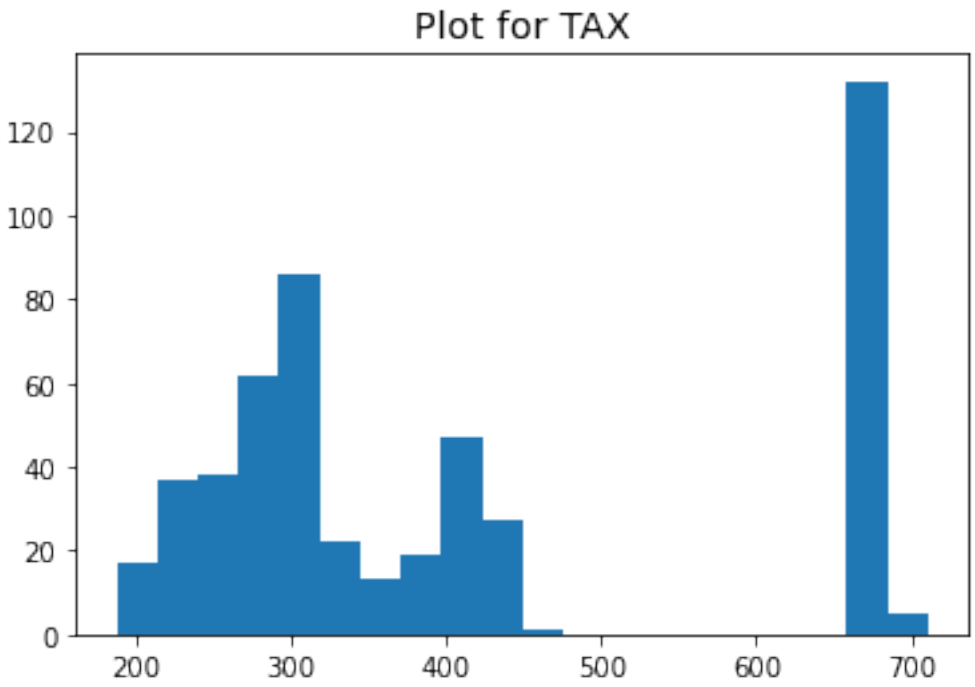

Plot for INDUS
120
100
80
60
40
20
0
0
5
10
15
20
25

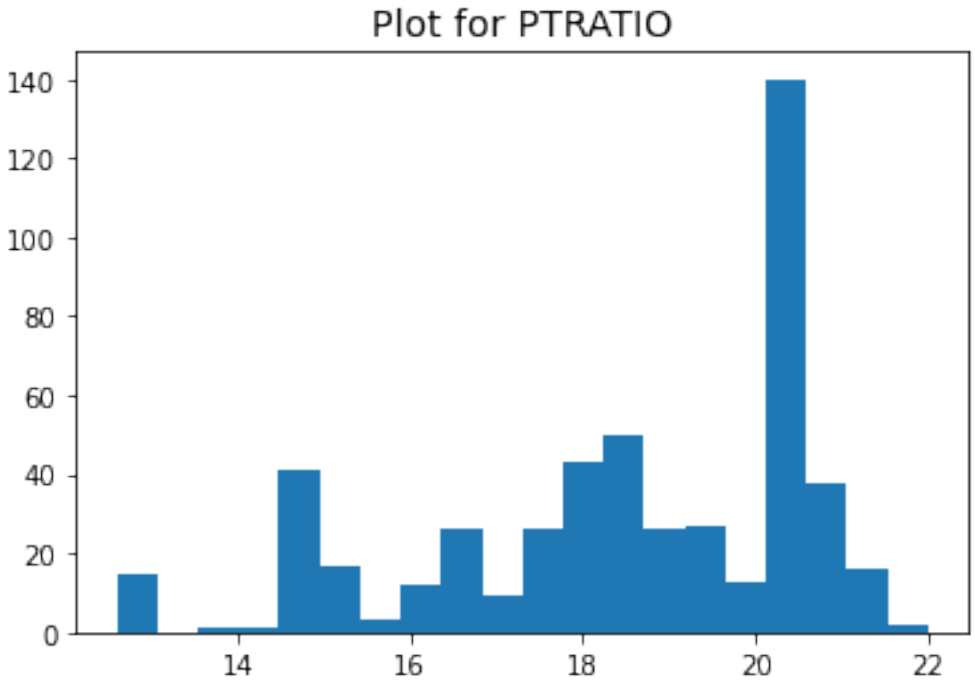

Plot for RM
100
80
60
40
20
0
4
5
6
7
8
9

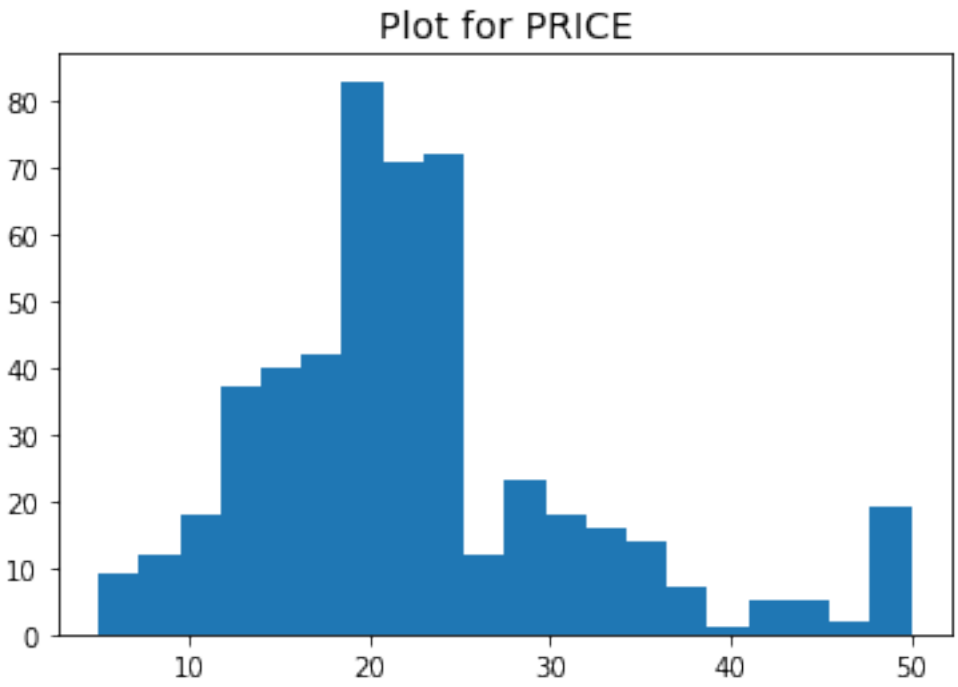

Plot for AGE
100
80
60
40
20
0
0
20
40
60
80
100


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

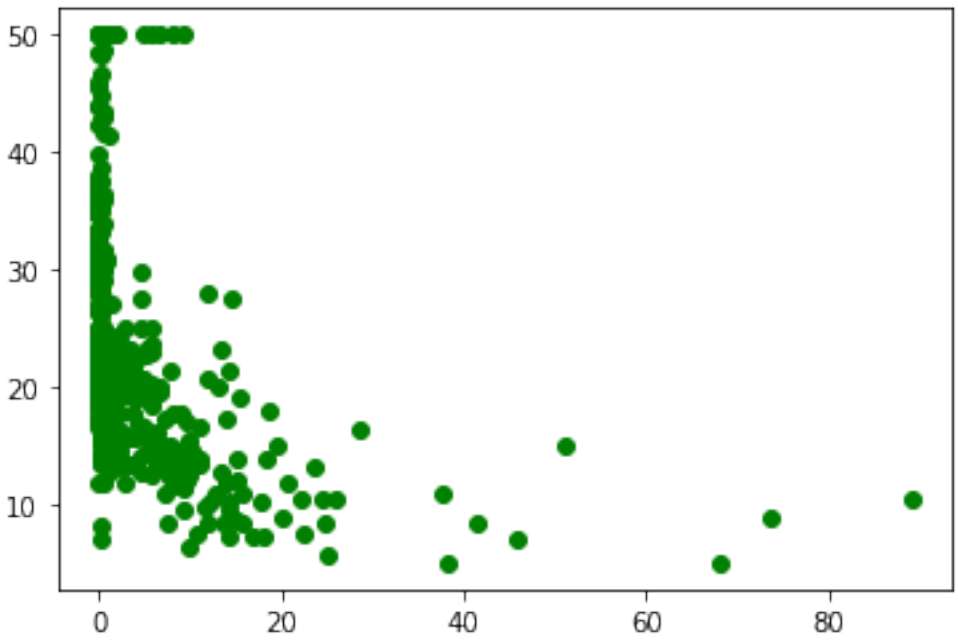

Plot for RAD
120
100
80
60
40
20
0
0
5
10
15
20
25


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

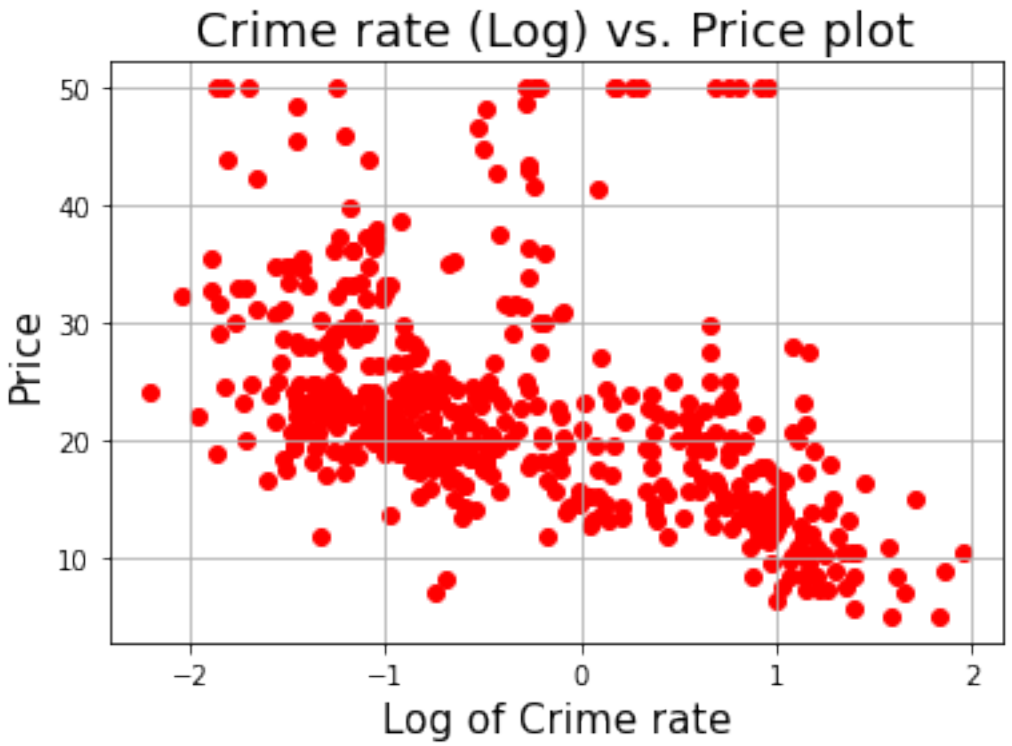

Plot for PTRATIO
140
120
100
80
60
40
20
0
14
16
18
20
22



[6]:
```
# Crime rate could be an indicator of house price (people don't want to live in␣
 ↪high-crime areas). Create a scatter plot of crime rate vs. Price.
plt.scatter(boston_house_df1['CRIM'],boston_house_df1['PRICE'], c ='green')
# The scatter() function plots one dot for each observation. It needs two␣
 ↪arrays of the same length, one for the values of the x-axis, and one for␣
 ↪values on the y-axis:
plt.show()
```



[7]:
```
# plot log10(crime) vs. Price. Create that plot and make it nice. Give proper␣
 ↪title, x-axis, y-axis label, make data points a color of your choice, etc.
# We can understand the relationship better if will use np.log10 function
plt.scatter(np.
 ↪log10(boston_house_df1['CRIM']),boston_house_df1['PRICE'],c='red')
# converting CRIM data value to log10
plt.title("Crime rate (Log) vs. Price plot", fontsize=18)
# displaying title
plt.xlabel("Log of Crime rate",fontsize=15)
# displaying x lable
plt.ylabel("Price",fontsize=15)
# displaying y lable
plt.grid(True)
# enabled grid in graph
plt.show()
```



[8]:
```
# calculate the mean rooms per dwelling
boston_house_df1['RM'].mean()
# calculating average by uisng mean function
```

[8]: 6.284634387351787

[9]:
```
# calculate median Age
boston_house_df1['AGE'].median()
```

[9]: 77.5

[10]:
```
# calculate average (mean) distances to five Boston employment centres
boston_house_df1['DIS'].mean()
```

[10]: 3.795042687747034

calculate the percentage of houses with low price ($< \$20{,}000$)

[11]:
```
# filterout the rows with houses price greater than or equal to 20
low_price_house=boston_house_df1.loc[boston_house_df1['PRICE']<20]
# pring out the rows with housing price lower than 20 k
print(low_price_house.head(5))
# calculating the percent of lowprice hours from total house
pcntoflphouse = (len(low_price_house)/len(boston_house_df1))*100
```

```
print("\nPercentage of house with less than $20,000 price is: ",pcntoflphouse)
```

```
      CRIM    ZN  INDUS     RM    AGE     DIS  RAD  TAX  PTRATIO  PRICE
8   0.21124  12.5   7.87  5.631  100.0  6.0821    5  311     15.2   16.5
9   0.17004  12.5   7.87  6.004   85.9  6.5921    5  311     15.2   18.9
10  0.22489  12.5   7.87  6.377   94.3  6.3467    5  311     15.2   15.0
11  0.11747  12.5   7.87  6.009   82.9  6.2267    5  311     15.2   18.9
14  0.63796   0.0   8.14  6.096   84.5  4.4619    4  307     21.0   18.2

Percentage of house with less than $20,000 price is:  41.50197628458498
```

Data Wrangling with Python: Activity 6, page 171 : Working with Adult Income Dataset (UCI)

[12]:
```
# Read in the adult income data set and check first 5 records
df = pd.read_csv("C:/Users/dell/Documents/docker/adult_income_data.csv")
df.head()
```

[12]:
```
   39          State-gov   77516   Bachelors  13        Never-married  \
0  50   Self-emp-not-inc   83311   Bachelors  13   Married-civ-spouse
1  38            Private  215646     HS-grad   9             Divorced
2  53            Private  234721        11th   7   Married-civ-spouse
3  28            Private  338409   Bachelors  13   Married-civ-spouse
4  37            Private  284582     Masters  14   Married-civ-spouse

          Adm-clerical   Not-in-family     Male  2174  0  40   United-States  \
0     Exec-managerial         Husband     Male     0  0  13   United-States
1   Handlers-cleaners   Not-in-family     Male     0  0  40   United-States
2   Handlers-cleaners         Husband     Male     0  0  40   United-States
3      Prof-specialty            Wife   Female     0  0  40            Cuba
4     Exec-managerial            Wife   Female     0  0  40   United-States

    <=50K
0   <=50K
1   <=50K
2   <=50K
3   <=50K
4   <=50K
```

[13]:
```
# Create a script that will read text file line by line, and extracts the first␣
 ↪phrase which is the header name
# create empty list to write extracted columns names from txt file
col_names = []
with open('C:/Users/dell/Documents/docker/adult_income_names.txt','r') as␣
 ↪ref_file:
    for line in ref_file:
        ref_file.readline()
```

```
        # read line by line in a loop & split the records in line by using :␣
 ↪seperator
        var=line.split(":")[0] # selecting first half of sperated word and␣
 ↪store it on variable
        col_names.append(var) # append variables to column name list
# print the list
col_names
```

[13]:
```
['age',
 'workclass',
 'fnlwgt',
 'education',
 'education-num',
 'marital-status',
 'occupation',
 'relationship',
 'sex',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'native-country']
```

[14]:
```
# list dosent contains the last column name,appending Income column name to␣
 ↪col_name list
col_names.append('Income')
```

[15]:
```
df1 = pd.read_csv("C:/Users/dell/Documents/docker/adult_income_data.csv",␣
 ↪names=col_names)
df1.head()
```

[15]:

	age	workclass	fnlwgt	education	education-num	marital-status	occupation	relationship	sex	capital-gain	capital-loss	hours-per-week	native-country	Income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	Male	2174	0	40	United-States	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	Male	0	0	13	United-States	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	Male					
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Male					
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Female					

```
2             0             0             40   United-States   <=50K
3             0             0             40   United-States   <=50K
4             0             0             40            Cuba   <=50K
```

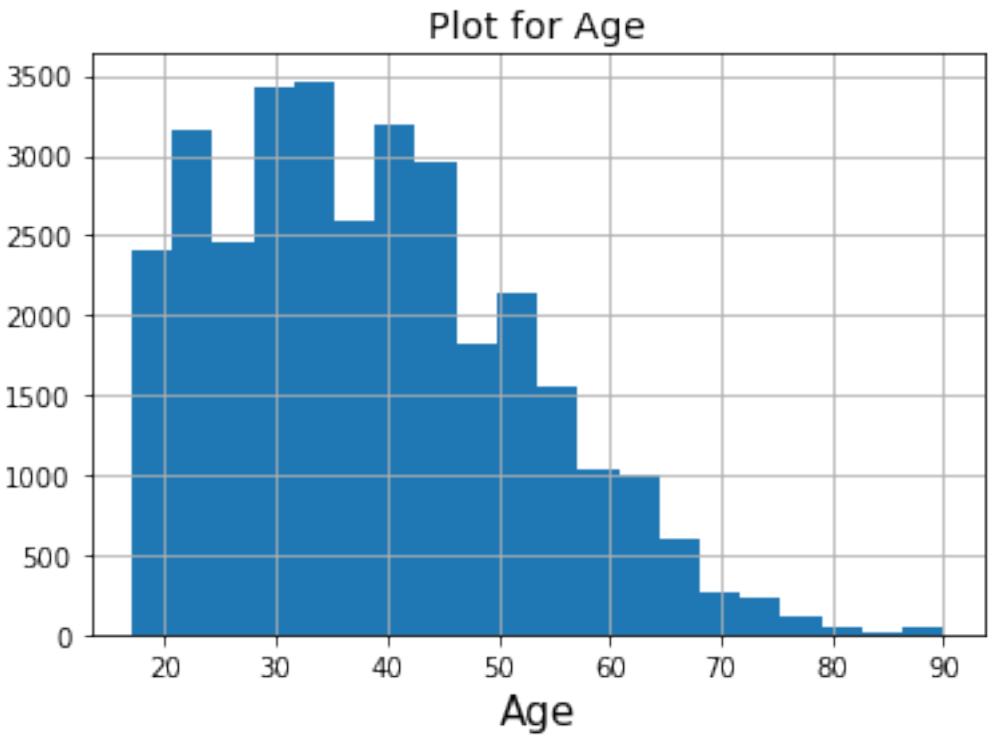
[16]:
```
# find the missing values in dataset
df1.isnull().sum()
```

[16]:
```
age               0
workclass         0
fnlwgt            0
education         0
education-num     0
marital-status    0
occupation        0
relationship      0
sex               0
capital-gain      0
capital-loss      0
hours-per-week    0
native-country    0
Income            0
dtype: int64
```

[17]:
```
# Create dataframe with only age,education,occupation by using subsetting
df2 = df1[['age','education','occupation']]
df2.head()
```

[17]:
```
   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
```

[18]:
```
# plot the histogram of age with a bin size 20
plt.hist(df2['age'],bins=20)
plt.title("Plot for Age",fontsize=14) # printing title
plt.xlabel("Age",fontsize=15) # displaying x lable
plt.grid(True)
```



[19]:
```
# create a function to strip white space character
def strip_whitespace(s):
    return s.strip()
```

[20]:
```
# 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
# Education column
df2.loc[:,'education_stripped'] = df1['education'].apply(strip_whitespace)
df2.loc[:,'education'] = df2.loc[:,'education_stripped']
df2.drop(labels=['education_stripped'],axis=1,inplace=True)

# Occupation column
df2.loc[:,'occupation_stripped']=df1['occupation'].apply(strip_whitespace)
df2.loc[:,'occupation']=df2.loc[:,'occupation_stripped']
df2.drop(labels=['occupation_stripped'],axis=1,inplace=True)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1597:
SettingWithCopyWarning:
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
  self.obj[key] = value
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1676:
SettingWithCopyWarning:
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
  self._setitem_single_column(ilocs[0], value, pi)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py:4308:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

[21]:
```
df2.head()
```

[21]:

	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

[22]:
```
# find the number of peoples who are aged between 30 & 50
df_filtered=df2[(df2['age']>=30) & (df2['age']<=50)]
answer_1=df_filtered.shape[0]
```

[23]:
```
print("There are {} peoples age between 30 and 50 in this dataset.".
 ↪format(answer_1))
```

```
There are 16390 peoples age between 30 and 50 in this dataset.
```

[24]:
```
# Group the records based on AGE and education to find how the mean age is␣
 ↪distributed
df2.groupby(['education']).mean()
```

[24]:

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
```

[25]:
```
# Group the records based on occupation and showing the summary statistics of␣
 ↪age
df2.groupby('occupation').describe(include='all')['age']
```

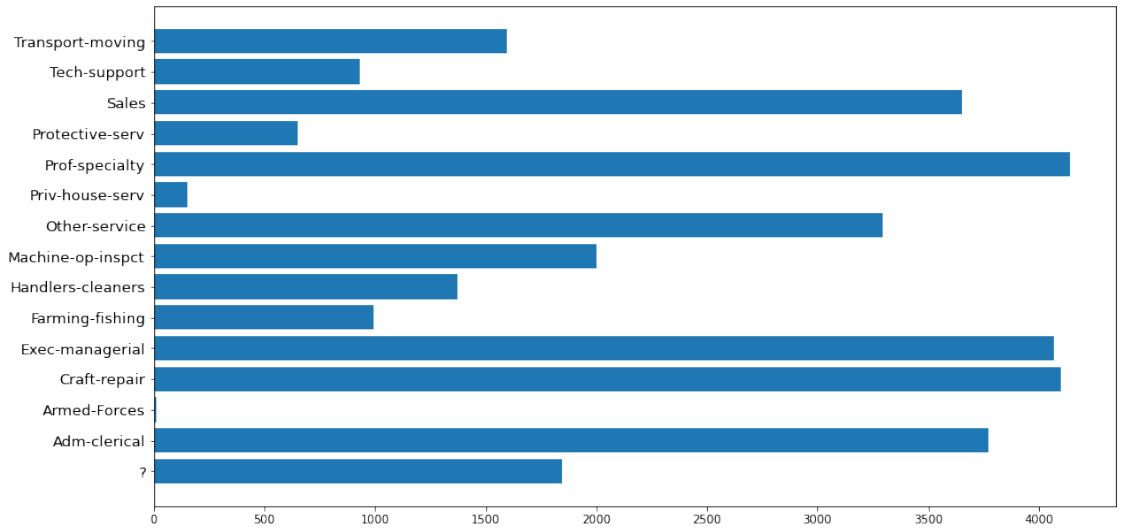
[25]:

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

```
Sales               25.0  35.0  47.0  90.0
Tech-support        28.0  36.0  44.0  73.0
Transport-moving    30.0  39.0  49.0  90.0
```

[26]:
```
# unknown occupation is having oldest worker - 40.882800
# unknown occupation is having largest share of workforce above 75th percentile
# Detecting outlier: Is there a particular occupation group which has very low␣
 ↪representation?
# Actually, just by looking at the table above,we found that the 'Armed-Forces'␣
 ↪group has only 9 count i.e. 9 data points. But how to detect it
```

[27]:
```
# use subset and groupby to find outliers
occupation_stats=df2.groupby('occupation').describe(include='all')['age']
# plot the values on a bar chart
plt.figure(figsize=(15,8))
plt.barh(y=occupation_stats.index,width=occupation_stats['count'])
plt.yticks(fontsize=13)
plt.show()
```



Create a series and practice basic arithmetic steps

a. Series 1 = 7.3, -2.5, 3.4, 1.5
b. Index = ‘a’, ‘c’, ‘d’, ‘e’
c. Series 2 = -2.1, 3.6, -1.5, 4, 3.1
d. Index = ‘a’, ‘c’, ‘e’, ‘f’, ‘g’
e. Add Series 1 and Series 2 together and print the results
f. Subtract Series 1 from Series 2 and print the results

[28]:
```
# initialize labels, list
data1 = [7.3, -2.5, 3.4, 1.5]
```

```
index1 = ['a', 'c', 'd', 'e']
data2 = [-2.1, 3.6, -1.5, 4, 3.1]
index2 = ['a', 'c', 'e', 'f', 'g']
```

[29]:
```
# create series from numpy array
series_1 = pd.Series(data=data1, index = index1)
series_2 = pd.Series(data=data2, index = index2)
print("series_1", series_1)
print("series_2", series_2)
```

```
series_1 a    7.3
c   -2.5
d    3.4
e    1.5
dtype: float64
series_2 a   -2.1
c    3.6
e   -1.5
f    4.0
g    3.1
dtype: float64
```

[30]:
```
# Apply binary addition between two pandas.Series instances
series_add = series_1 + series_2
print(series_add)
```

```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
dtype: float64
```

[31]:
```
# fixing the NaN due to index mismatch , using fill misisng value= 0
series_1.add(series_2, fill_value=0)
```

[31]:
```
a    5.2
c    1.1
d    3.4
e    0.0
f    4.0
g    3.1
dtype: float64
```

[32]:
```
# Apply binary substration between two pandas.Series instances
series_sub = series_2 - series_1
print(series_sub)
```

```
a   -9.4
c    6.1
d    NaN
e   -3.0
f    NaN
g    NaN
dtype: float64
```

[33]:
```
# fixing the NaN issue due to index mismatch , using fill misisng value= 0
series_2.subtract(series_1, fill_value=0)
```

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

[ ]:
```

```