Week_3_4

July 4, 2023

1. Data Wrangling with Python: Activity 5, page 116

Load the necessary libraries

[2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Following are the details of the attributes of this dataset for your reference. You may have to refer them while answering question on this activity.

- **CRIM**: per capita crime rate by town
- **ZN**: proportion of residential land zoned for lots over 25,000 sq.ft.
- **INDUS**: proportion of non-retail business acres per town
- **CHAS**: Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- **NOX**: nitric oxides concentration (parts per 10 million)
- **RM**: average number of rooms per dwelling
- **AGE**: proportion of owner-occupied units built prior to 1940
- **DIS**: weighted distances to five Boston employment centres
- **RAD**: index of accessibility to radial highways
- **TAX**: full-value property-tax rate per 10,000 dollars
- **PTRATIO**: pupil-teacher ratio by town
- **B**: 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- **LSTAT**: % of lower status of the population
- **PRICE**: Median value of owner-occupied homes in $1000's

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

[3]:
```
import os
current_path = os.getcwd()
print(current_path)
```

```
/Users/feliperodriguez/Library/CloudStorage/OneDrive-BellevueUniversity/DSC 540
Data Preperation/Week3_4
```

[4]:
```
df = pd.read_csv('/Users/feliperodriguez/Library/CloudStorage/
 ↪OneDrive-BellevueUniversity/DSC 540 Data Preperation/Week3_4/Boston_housing.
 ↪csv')
```

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

[5]:
```
# Shows first 10 columns
df.head(10)
```

[5]:
```
      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
5  0.02985   0.0   2.18     0  0.458  6.430   58.7  6.0622    3  222     18.7
6  0.08829  12.5   7.87     0  0.524  6.012   66.6  5.5605    5  311     15.2
7  0.14455  12.5   7.87     0  0.524  6.172   96.1  5.9505    5  311     15.2
8  0.21124  12.5   7.87     0  0.524  5.631  100.0  6.0821    5  311     15.2
9  0.17004  12.5   7.87     0  0.524  6.004   85.9  6.5921    5  311     15.2

        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
5  394.12   5.21   28.7
6  395.60  12.43   22.9
7  396.90  19.15   27.1
8  386.63  29.93   16.5
9  386.71  17.10   18.9
```

[6]:
```
# Show number of Records and Columns
df.shape
```

[6]:
```
(506, 14)
```

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

[7]:
```
# Creates new dataframe with columns removed
df2 = df.drop(columns=['CHAS','NOX', 'B', 'LSTAT'])
```

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

[8]:
```
# Shows last 7 records of new dataframe
df2.tail(7)
```

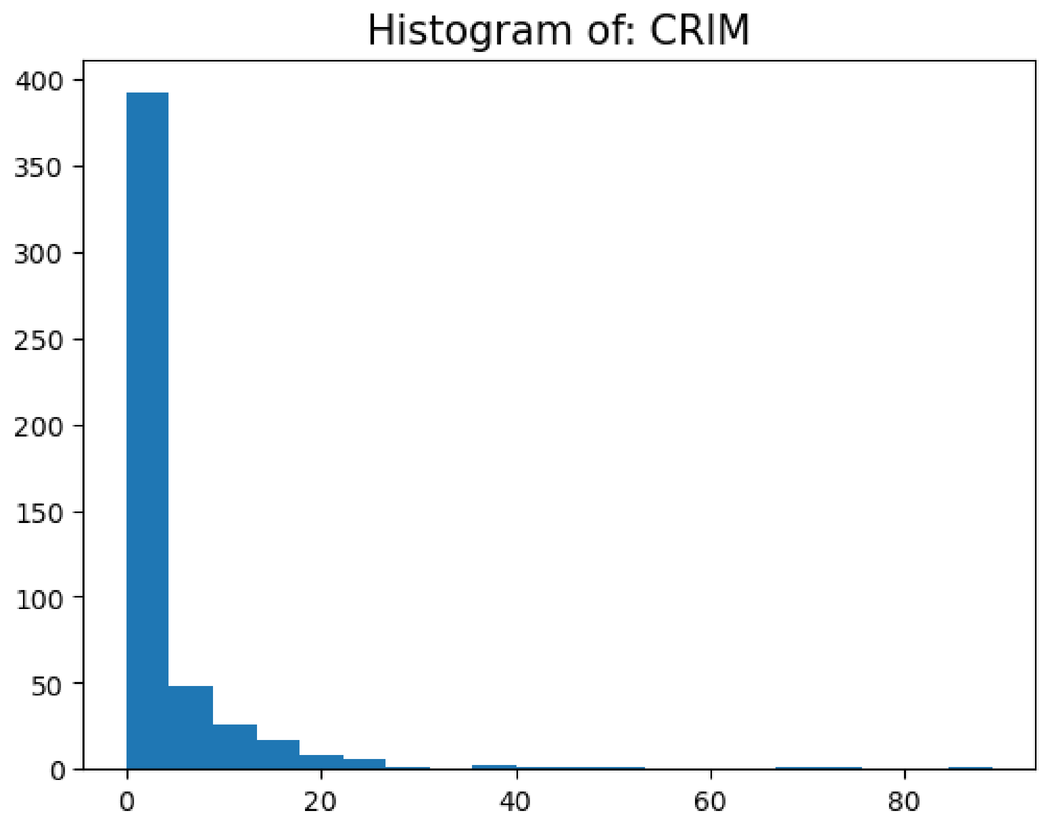
[8]:
```
        CRIM   ZN  INDUS     RM   AGE     DIS  RAD  TAX  PTRATIO  PRICE
499  0.17783  0.0   9.69  5.569  73.5  2.3999    6  391     19.2   17.5
500  0.22438  0.0   9.69  6.027  79.7  2.4982    6  391     19.2   16.8
501  0.06263  0.0  11.93  6.593  69.1  2.4786    1  273     21.0   22.4
502  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
```

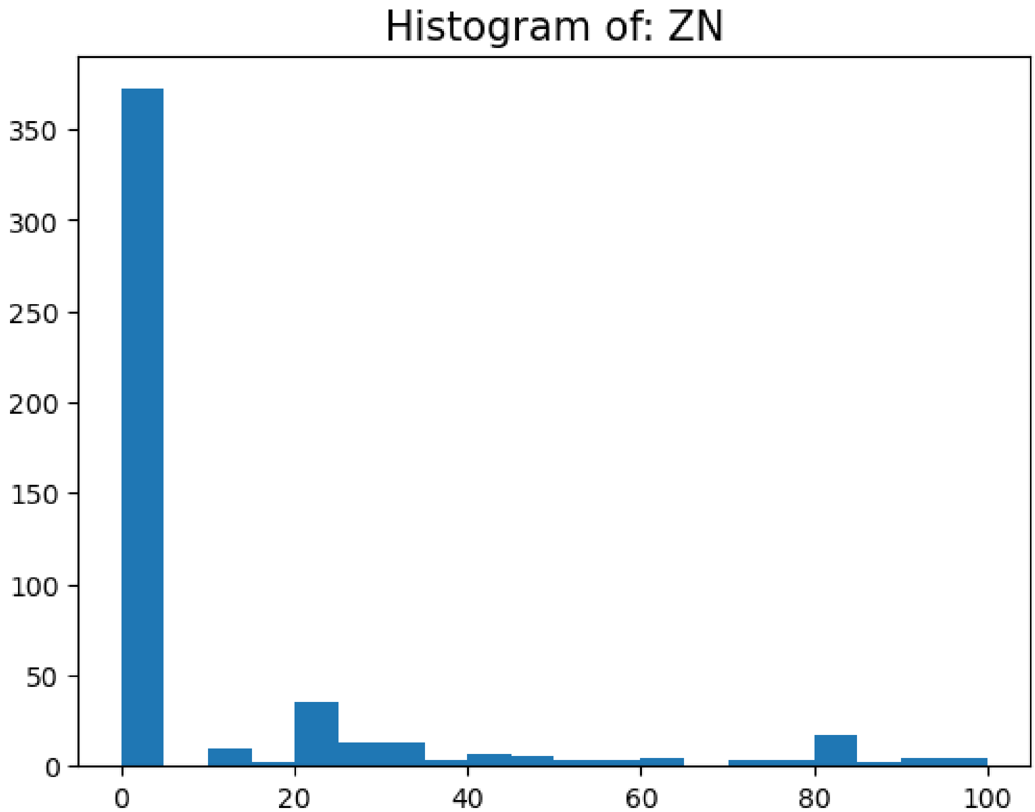
504	0.10959	0.0	11.93	6.794	89.3	2.3889	1	273	21.0	22.0
505	0.04741	0.0	11.93	6.030	80.8	2.5050	1	273	21.0	11.9

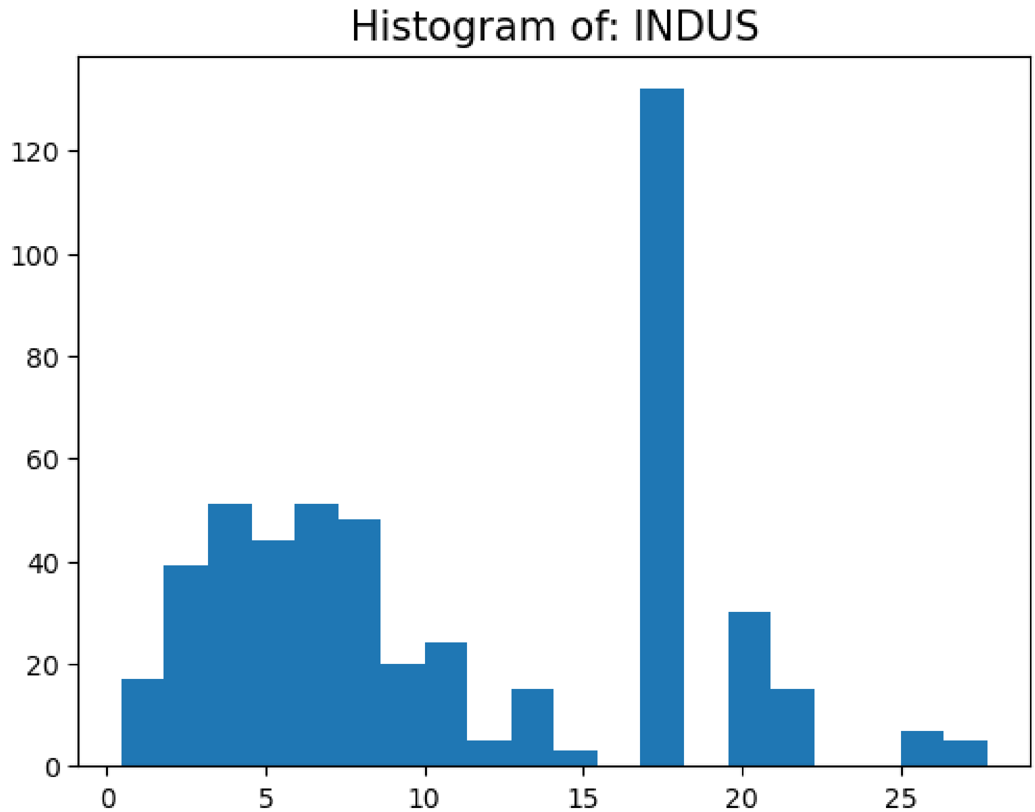
Plot the histogram of all the variables (columns) in the new DataFrame.

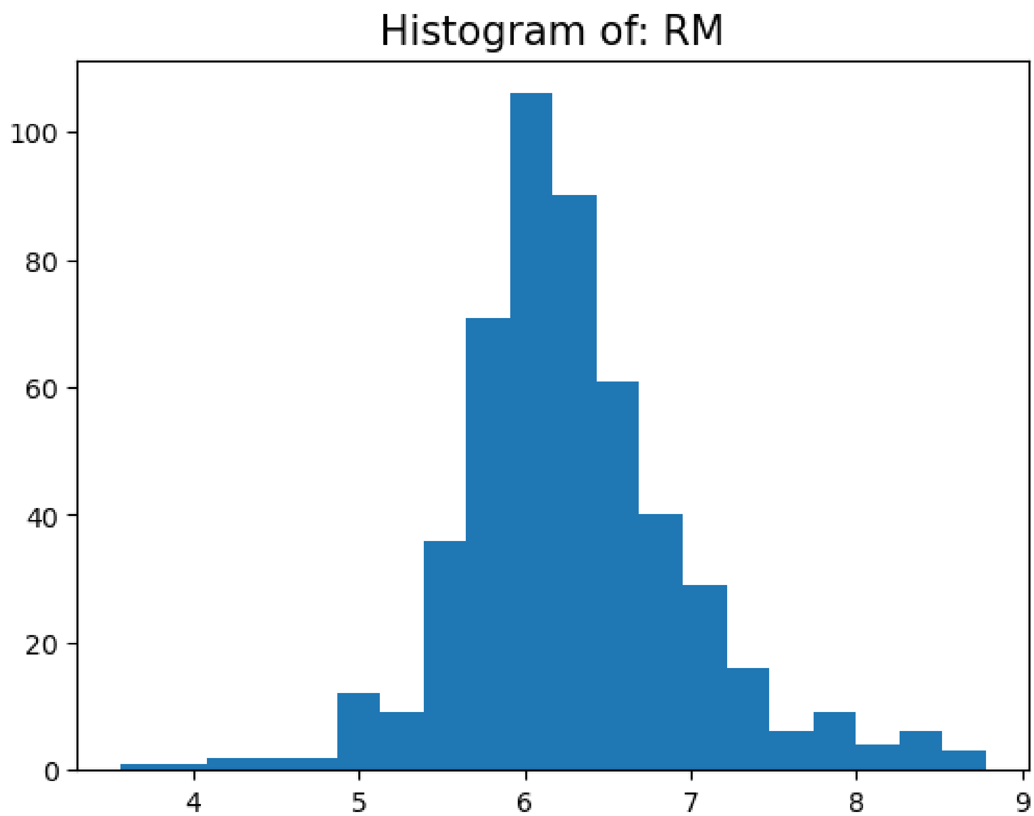
Plot them using a for loop. Try to add a unique title to a plot

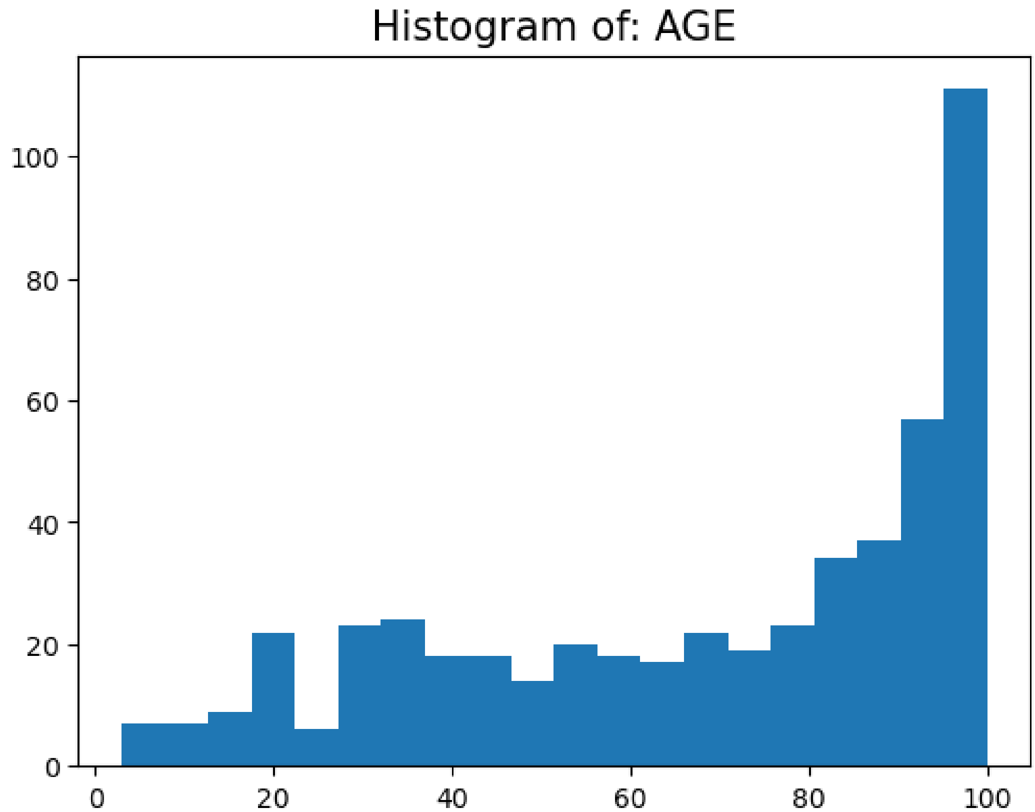
[9]:
```
# Loop that creates histogram of each column
for x in df2.columns:
    # Gets Name of column for title
    plt.title("Histogram of: "+x,fontsize=15)
    # Creates historgram for each column
    plt.hist(df2[x],bins=20)
    # Displays histograms
    plt.show()
```

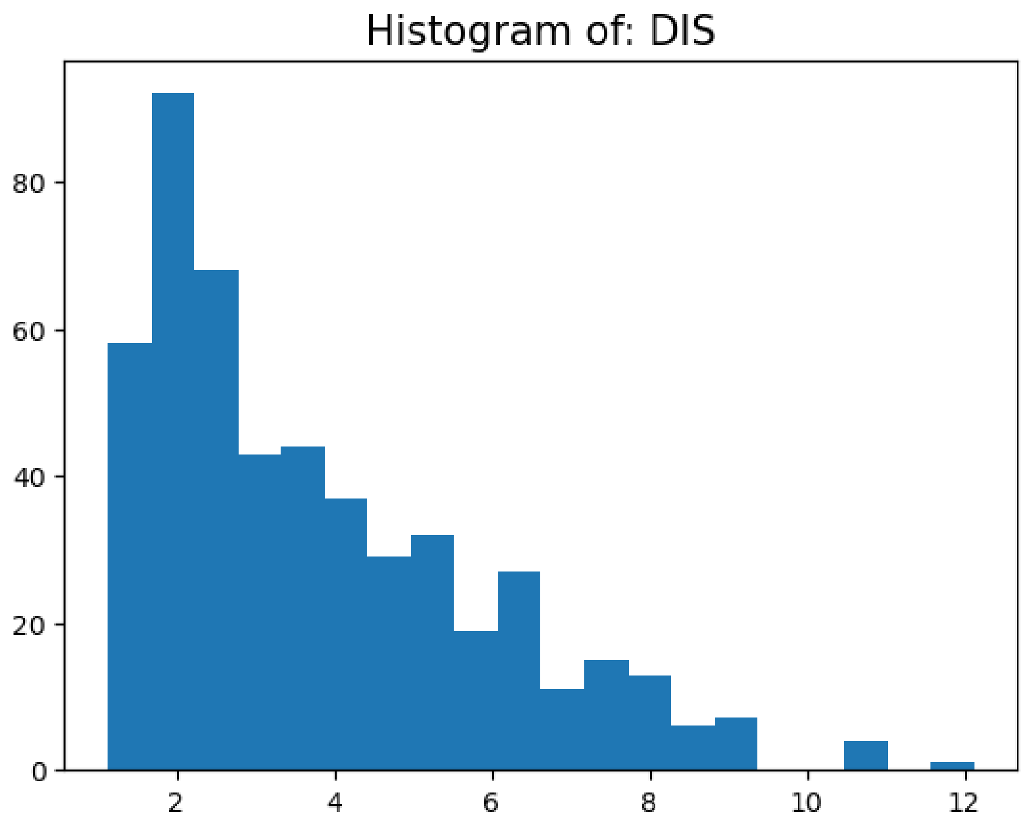


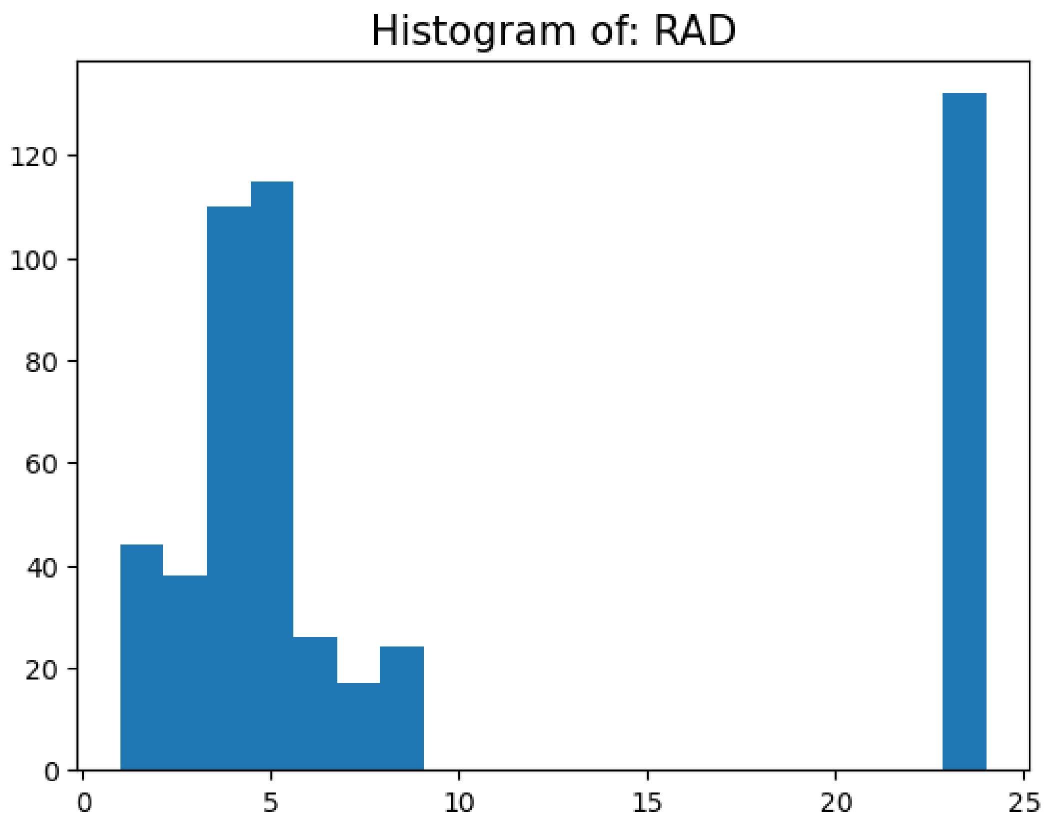

Histogram of: ZN
350
300
250
200
150
100
50
0
0
20
40
60
80
100

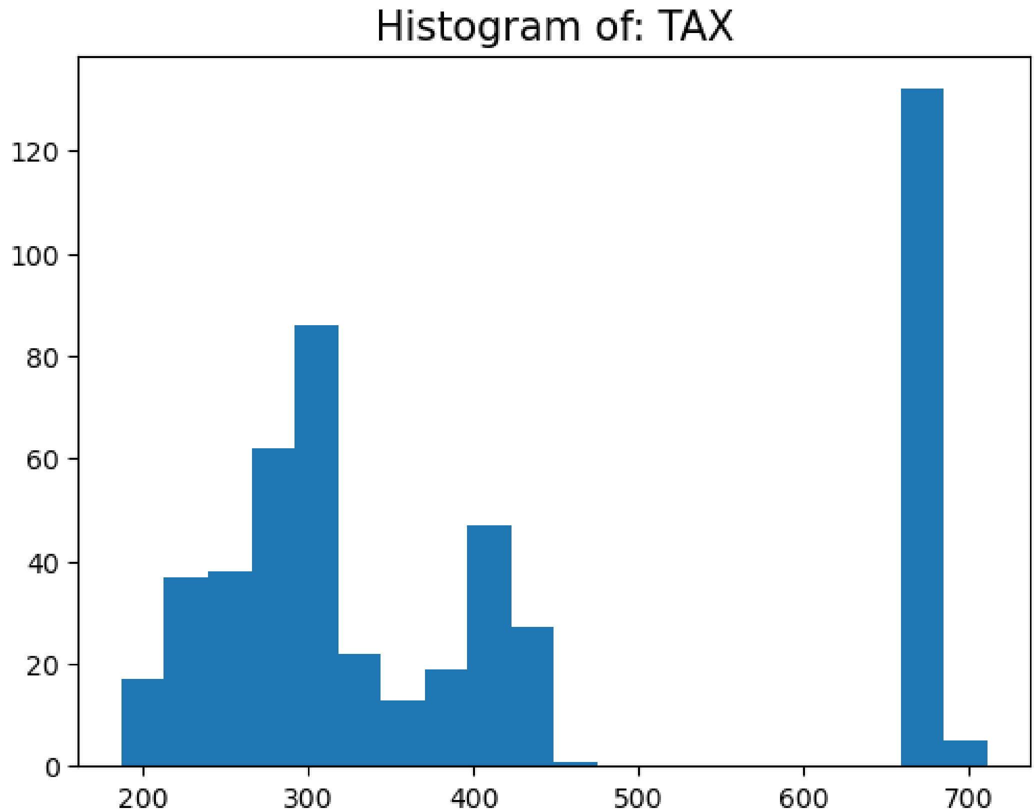

Histogram of: INDUS
0
20
40
60
80
100
120
0
5
10
15
20
25

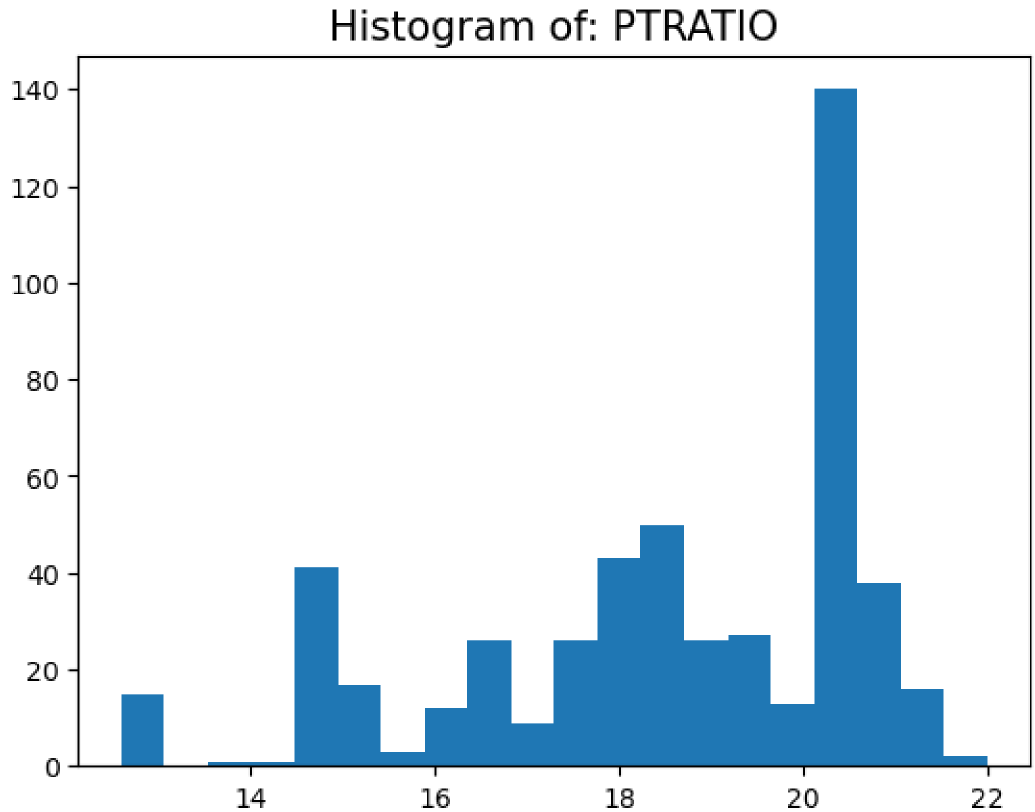

Histogram of: RM
0
20
40
60
80
100
4
5
6
7
8
9

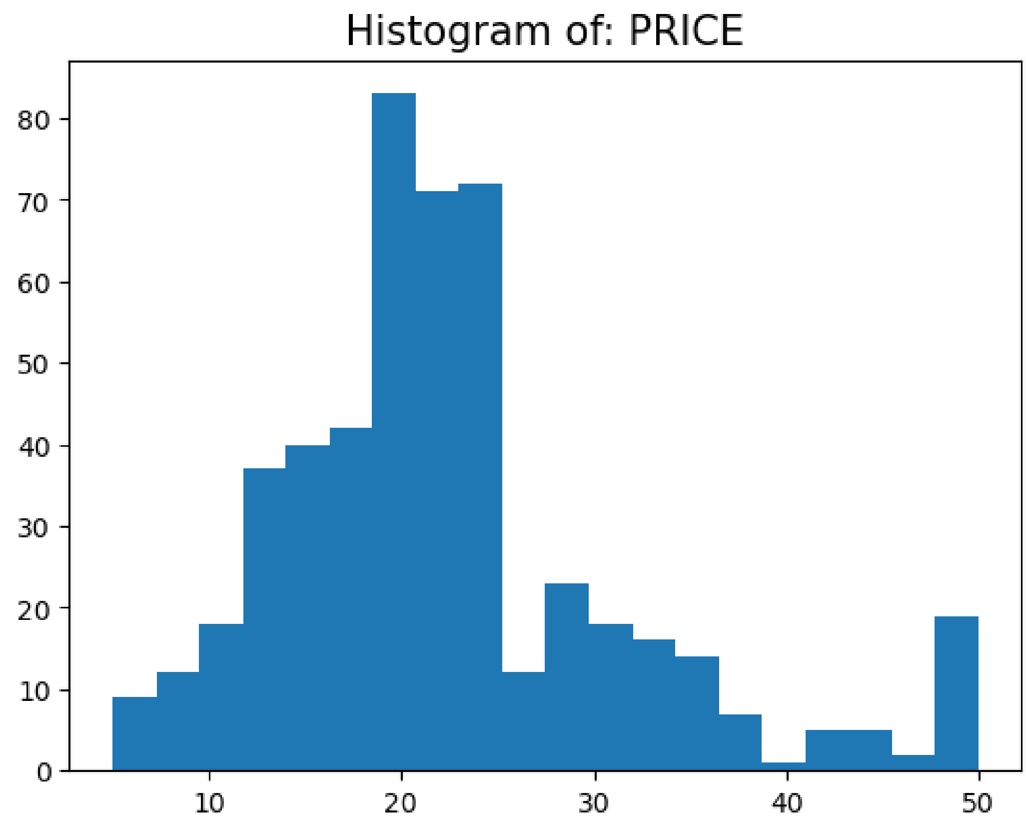

Histogram of: AGE
100
80
60
40
20
0
0
20
40
60
80
100


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


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

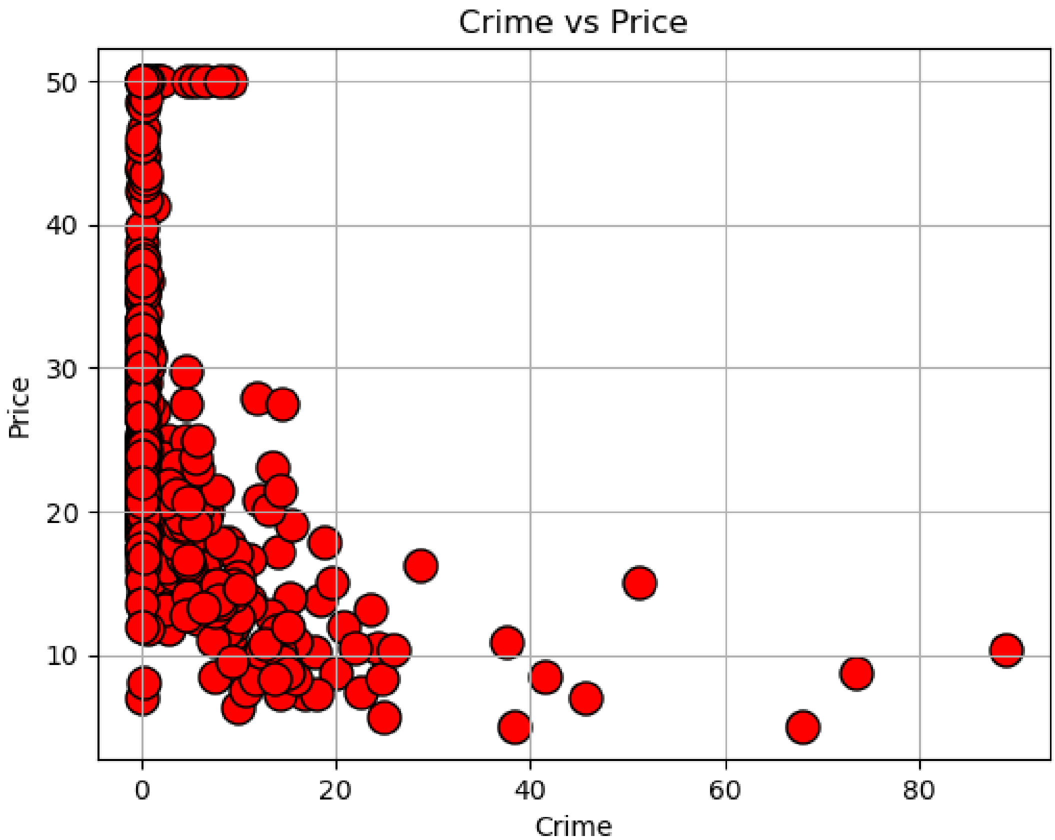

Histogram of: TAX
120
100
80
60
40
20
0
200
300
400
500
600
700


Histogram of: PTRATIO
140
120
100
80
60
40
20
0
14
16
18
20
22



Create a Scatter Plot of crime rate versus price

[10]:
```
# Creates scatter plot using crime and price
df2.plot.scatter('CRIM', 'PRICE', s=150, c='Red', edgecolor='k')
# Add grid to scatter plot
plt.grid(True)
# Creates titles
plt.title("Crime vs Price")
# Labels X
plt.xlabel("Crime")
# Labels Y
plt.ylabel("Price")
# Displays Scatter Plot
plt.show()
```



Plot using **log10(crime)** versus **price**

[11]:
```
# Creates scatter plot us log10 of crime and price
plt.scatter(np.log10(df2['CRIM']), df2['PRICE'], s=150, c='Red', edgecolor='k')
# Add grid to scatter plot
plt.grid(True)
# Creates Title
plt.title("Crime vs Price")
# Labels X
plt.xlabel("Crime")
# Labels Y
plt.ylabel("Price")
# Displays Scatter Plot
plt.show()
```

Calculate some useful statistics, such as mean rooms per dwellings, median age, mean distances to five Boston Employee Centers, and the percentages of houses with a low price (<$20,000)

[12]:
```
# Mean of rooms per dwellings
df2['RM'].mean()
```

[12]: 6.284634387351787

[13]:
```
# Median Age
df2['AGE'].median()
```

[13]: 77.5

[14]:
```
# Mean Distances
df2['DIS'].mean()
```

[14]: 3.795042687747034

[15]:
```
# New dataframe based on prices less than 20k
less_than_20 = df2['PRICE']<20
```

```
# Finds the mean and multiplies by 100 to get percent
percent_less_than_20 =less_than_20.mean()*100
# Pints Percent amount
print("Percent less than $20,000:", percent_less_than_20)
```

```
Percent less than $20,000: 41.50197628458498
```

2. Data Wrangling with Python: Activity 6, page 171

Load the necessary libraries

[16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Following are the details of the type of the attributes of this dataset for your reference. You may have to refer them while answering question on this activity. Note that, many of the attributes are of discrete factor type. These are common type for a classification problem unlike continuous numeric values used for regression problems.

- **age**: continuous.
- **workclass**: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov, State-gov, Without-pay, Never-worked.
- **fnlwgt**: continuous.
- **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.
- **education-num**: continuous.
- **marital-status**: Married-civ-spouse, Divorced, Never-married, Separated, Widowed, Married-spouse-absent, Married-AF-spouse.
- **occupation**: Tech-support, Craft-repair, Other-service, Sales, Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct, Adm-clerical, Farming-fishing, Transport-moving, Priv-house-serv, Protective-serv, Armed-Forces.
- **relationship**: Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
- **race**: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
- **sex**: Female, Male.
- **capital-gain**: continuous.
- **capital-loss**: continuous.
- **hours-per-week**: continuous.
- **native-country**: United-States, Cambodia, England, Puerto-Rico, Canada, Germany, Outlying-US(Guam-USVI-etc), India, Japan, Greece, South, China, Cuba, Iran, Honduras, Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican-Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El-Salvador, Trinadad&Tobago, Peru, Hong, Holand-Netherlands.

Read the adult income dataset

[17]:
```
# Read in File and Display 10 records
```

```
df3 = pd.read_csv('/Users/feliperodriguez/Library/CloudStorage/
↪OneDrive-BellevueUniversity/DSC 540 Data Preperation/Week3_4/
↪adult_income_data.csv')
df3.head(10)
```

[17]:

```
   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
5  49            Private  160187            9th   5  Married-spouse-absent
6  52   Self-emp-not-inc  209642        HS-grad   9     Married-civ-spouse
7  31            Private   45781        Masters  14          Never-married
8  42            Private  159449      Bachelors  13     Married-civ-spouse
9  37            Private  280464   Some-college  10     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
5       Other-service   Not-in-family   Female      0  0  16         Jamaica
6     Exec-managerial         Husband     Male      0  0  45   United-States
7      Prof-specialty   Not-in-family   Female  14084  0  50   United-States
8     Exec-managerial         Husband     Male   5178  0  40   United-States
9     Exec-managerial         Husband     Male      0  0  80   United-States

    <=50K
0   <=50K
1   <=50K
2   <=50K
3   <=50K
4   <=50K
5   <=50K
6    >50K
7    >50K
8    >50K
9    >50K
```

Create a script that will read the text file line by line

[18]:

```
# Creates header values when reading the txt file
# Creates head list
headers = []
# Opens the file
```

```
with open('adult_income_names.txt','r') as f:
    # Iterates through each line in file
    for line in f:
        # Read the line
        f.readline()
        # Splits header from the rest
        var=line.split(":")[0]
        # Adds header name to list
        headers.append(var)
```

[19]:
```
# Displays Values
headers
```

[19]:
```
['age',
 'workclass',
 'fnlwgt',
 'education',
 'education-num',
 'marital-status',
 'occupation',
 'relationship',
 'sex',
 'capital-gain',
 'capital-loss',
 'hours-per-week',
 'native-country']
```

Add a name of **Income** for the response variable to the dataset

[20]:
```
# Adds income into headers
headers.append('Income')
```

[21]:
```
# Adds headers to data and displays first 10 records
df3 = pd.read_csv('adult_income_data.csv', names=headers)
df3.head(10)
```

[21]:
```
   age         workclass  fnlwgt   education  education-num  \
0   39         State-gov   77516   Bachelors             13
1   50  Self-emp-not-inc   83311   Bachelors             13
2   38           Private  215646     HS-grad              9
3   53           Private  234721        11th              7
4   28           Private  338409   Bachelors             13
5   37           Private  284582     Masters             14
6   49           Private  160187         9th              5
7   52  Self-emp-not-inc  209642     HS-grad              9
8   31           Private   45781     Masters             14
9   42           Private  159449   Bachelors             13
```

```
          marital-status          occupation    relationship      sex  \
0          Never-married        Adm-clerical   Not-in-family     Male
1     Married-civ-spouse     Exec-managerial         Husband     Male
2               Divorced   Handlers-cleaners   Not-in-family     Male
3     Married-civ-spouse   Handlers-cleaners         Husband     Male
4     Married-civ-spouse      Prof-specialty            Wife   Female
5     Married-civ-spouse     Exec-managerial            Wife   Female
6  Married-spouse-absent       Other-service   Not-in-family   Female
7     Married-civ-spouse     Exec-managerial         Husband     Male
8          Never-married      Prof-specialty   Not-in-family   Female
9     Married-civ-spouse     Exec-managerial         Husband     Male

   capital-gain  capital-loss  hours-per-week  native-country  Income
0          2174             0              40   United-States   <=50K
1             0             0              13   United-States   <=50K
2             0             0              40   United-States   <=50K
3             0             0              40   United-States   <=50K
4             0             0              40            Cuba   <=50K
5             0             0              40   United-States   <=50K
6             0             0              16         Jamaica   <=50K
7             0             0              45   United-States    >50K
8         14084             0              50   United-States    >50K
9          5178             0              40   United-States    >50K
```

Find the missing Values

[22]:
```python
# Creates Loop that counts missing values for each column
for c in df3.columns:
    # Counts the amount null in each column
    miss = df3[c].isnull().sum()
    # If missing values is greater than 0 prints statement
    if miss > 0:
        print( " {} has {} missing value(s)".format(c,miss))
    # Handle if there are no missing values
    else:
        print( " {} has NO missing value(s)".format(c))
```

```
age has NO missing value(s)
workclass has NO missing value(s)
fnlwgt has NO missing value(s)
education has NO missing value(s)
education-num has NO missing value(s)
marital-status has NO missing value(s)
occupation has NO missing value(s)
relationship has NO missing value(s)
sex has NO missing value(s)
capital-gain has NO missing value(s)
```

```
capital-loss has NO missing value(s)
hours-per-week has NO missing value(s)
native-country has NO missing value(s)
Income has NO missing value(s)
```

Create a dataframe with only age, education, and occupation by subsetting

[23]:
```
# Shows column names of df3
df3.columns
```

[23]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education-num',
       'marital-status', 'occupation', 'relationship', 'sex', 'capital-gain',
       'capital-loss', 'hours-per-week', 'native-country', 'Income'],
      dtype='object')
```

[24]:
```
# Creates subset using only three columns and displays first 10 records
df3_subset = df3[['age','education','occupation']]
df3_subset.head(10)
```

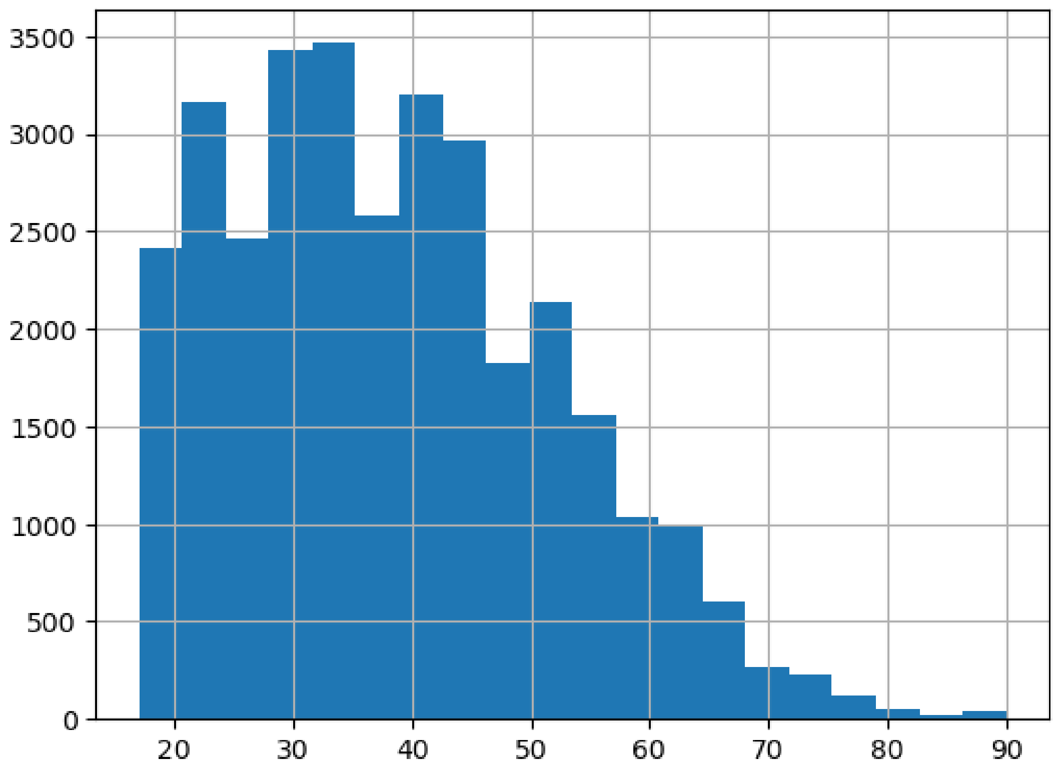
[24]:

	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
5	37	Masters	Exec-managerial
6	49	9th	Other-service
7	52	HS-grad	Exec-managerial
8	31	Masters	Prof-specialty
9	42	Bachelors	Exec-managerial

Plot a histogram of age with a bin size of 20

[25]:
```
# Creates histogram of Age
df3['age'].hist(bins=20)
```

[25]:
```
<AxesSubplot:>
```



Create a function to strip whitepspace characters

[26]:
```
# Function that takes string as a parameter that removes white space
def strip_whitespace(s):
    return s.strip()
```

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.

[27]:
```
# shows data type of each column
df3_subset.dtypes
```

[27]:
```
age             int64
education      object
occupation     object
dtype: object
```

[28]:
```
# Strips whitespace from eduction
df3_subset['education_stripped']= df3['education'].apply(strip_whitespace)
# Append stripped whitespace column to old column
df3_subset['education']= df3_subset['education_stripped']
# Removes whitespace removed column from df
df3_subset.drop(labels=['education_stripped'],axis=1,inplace=True)
```

```python
# Occupation column
df3_subset['occupation_stripped']=df3['occupation'].apply(strip_whitespace)
# Append stripped whitespace column to old column
df3_subset['occupation']=df3_subset['occupation_stripped']
# Removes whitespace removed column from df
df3_subset.drop(labels=['occupation_stripped'],axis=1,inplace=True)
```

```
/var/folders/sr/xvmzsbj91c91yq0f0qnq71xh0000gn/T/ipykernel_53030/3166918394.py:2
: 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
  df3_subset['education_stripped']= df3['education'].apply(strip_whitespace)
/var/folders/sr/xvmzsbj91c91yq0f0qnq71xh0000gn/T/ipykernel_53030/3166918394.py:4
: 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
  df3_subset['education']= df3_subset['education_stripped']
/var/folders/sr/xvmzsbj91c91yq0f0qnq71xh0000gn/T/ipykernel_53030/3166918394.py:6
: 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
  df3_subset.drop(labels=['education_stripped'],axis=1,inplace=True)
/var/folders/sr/xvmzsbj91c91yq0f0qnq71xh0000gn/T/ipykernel_53030/3166918394.py:9
: 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
  df3_subset['occupation_stripped']=df3['occupation'].apply(strip_whitespace)
/var/folders/sr/xvmzsbj91c91yq0f0qnq71xh0000gn/T/ipykernel_53030/3166918394.py:1
1: 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
  df3_subset['occupation']=df3_subset['occupation_stripped']
```

```
/var/folders/sr/xvmzsbj91c91yq0f0qnq71xh0000gn/T/ipykernel_53030/3166918394.py:1
3: 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
  df3_subset.drop(labels=['occupation_stripped'],axis=1,inplace=True)
```

[29]:
```python
# Displays first five records with whitespace now removed
df3_subset.head()
```

[29]:

	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

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

[30]:
```python
# Function to count people within the age of 30 and 50
# Starts list at 0
count = 0
for age in df3_subset['age']:
    # Count increases by 1 if age is between 30 and 50
    if age >= 30 and age <= 50:
        count += 1
    # Any other ages get skipped
    else:
        pass

print(count)
```

```
16390
```

Group the records based on age and education to find how the mean age is distributed.

[31]:
```python
# Subset grouped by education and using describe on age to get statistical data
df3_subset.groupby('education').describe()['age']
```

[31]:

	count	mean	std	min	25%	50%	75%	max
education								
10th	933.0	37.429796	16.720713	17.0	22.00	34.0	52.0	90.0
11th	1175.0	32.355745	15.545485	17.0	18.00	28.0	43.0	90.0
12th	433.0	32.000000	14.334625	17.0	19.00	28.0	41.0	79.0
1st-4th	168.0	46.142857	15.615625	19.0	33.00	46.0	57.0	90.0
5th-6th	333.0	42.885886	15.557285	17.0	29.00	42.0	54.0	84.0
7th-8th	646.0	48.445820	16.092350	17.0	34.25	50.0	61.0	90.0

9th	514.0	41.060311	15.946862	17.0	28.00	39.0	54.0	90.0
Assoc-acdm	1067.0	37.381443	11.095177	19.0	29.00	36.0	44.0	90.0
Assoc-voc	1382.0	38.553546	11.631300	19.0	30.00	37.0	46.0	84.0
Bachelors	5355.0	38.904949	11.912210	19.0	29.00	37.0	46.0	90.0
Doctorate	413.0	47.702179	11.784716	24.0	39.00	47.0	55.0	80.0
HS-grad	10501.0	38.974479	13.541524	17.0	28.00	37.0	48.0	90.0
Masters	1723.0	44.049913	11.068935	18.0	36.00	43.0	51.0	90.0
Preschool	51.0	42.764706	15.126914	19.0	31.00	41.0	53.5	75.0
Prof-school	576.0	44.746528	11.962477	25.0	36.00	43.0	51.0	90.0
Some-college	7291.0	35.756275	13.474051	17.0	24.00	34.0	45.0	90.0

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

[32]:
```
# Creates and displays summary of occupation by age
occupation_describe = df3_subset.groupby('occupation').describe()['age']
occupation_describe
```

[32]:

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

[33]:
```
# Sorts dataframe by highest mean of age to find the occupation where oldest␣
↪people work
occupation_describe.sort_values(by=['mean'], ascending=False)
```

[33]:

occupation	count	mean	std	min	25%	50%	75%	max
Exec-managerial	4066.0	42.169208	11.974548	17.0	33.0	41.0	50.0	90.0
Priv-house-serv	149.0	41.724832	18.633688	17.0	24.0	40.0	57.0	81.0
Farming-fishing	994.0	41.211268	15.070283	17.0	29.0	39.0	52.0	90.0
?	1843.0	40.882800	20.336350	17.0	21.0	35.0	61.0	90.0

Prof-specialty	4140.0	40.517633	12.016676	17.0	31.0	40.0	48.0	90.0
Transport-moving	1597.0	40.197871	12.450792	17.0	30.0	39.0	49.0	90.0
Craft-repair	4099.0	39.031471	11.606436	17.0	30.0	38.0	47.0	90.0
Protective-serv	649.0	38.953775	12.822062	17.0	29.0	36.0	47.0	90.0
Machine-op-inspct	2002.0	37.715285	12.068266	17.0	28.0	36.0	46.0	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
Adm-clerical	3770.0	36.964456	13.362998	17.0	26.0	35.0	46.0	90.0
Other-service	3295.0	34.949621	14.521508	17.0	22.0	32.0	45.0	90.0
Handlers-cleaners	1370.0	32.165693	12.372635	17.0	23.0	29.0	39.0	90.0
Armed-Forces	9.0	30.222222	8.089774	23.0	24.0	29.0	34.0	46.0

The profession Exec Managerial has the highest workers on average.

[34]:
```python
# # Sorts dataframe by occupations where they have the highest 75th percentile
occupation_describe.sort_values(by=['75%'], ascending=False)
```

[34]:

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
Priv-house-serv	149.0	41.724832	18.633688	17.0	24.0	40.0	57.0	81.0
Farming-fishing	994.0	41.211268	15.070283	17.0	29.0	39.0	52.0	90.0
Exec-managerial	4066.0	42.169208	11.974548	17.0	33.0	41.0	50.0	90.0
Transport-moving	1597.0	40.197871	12.450792	17.0	30.0	39.0	49.0	90.0
Prof-specialty	4140.0	40.517633	12.016676	17.0	31.0	40.0	48.0	90.0
Craft-repair	4099.0	39.031471	11.606436	17.0	30.0	38.0	47.0	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
Adm-clerical	3770.0	36.964456	13.362998	17.0	26.0	35.0	46.0	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
Tech-support	928.0	37.022629	11.316594	17.0	28.0	36.0	44.0	73.0
Handlers-cleaners	1370.0	32.165693	12.372635	17.0	23.0	29.0	39.0	90.0
Armed-Forces	9.0	30.222222	8.089774	23.0	24.0	29.0	34.0	46.0

Priv House Serv has the largest share of workers above the 75th Precentile. However, the count for this is very low which gives us an uneven proportion.

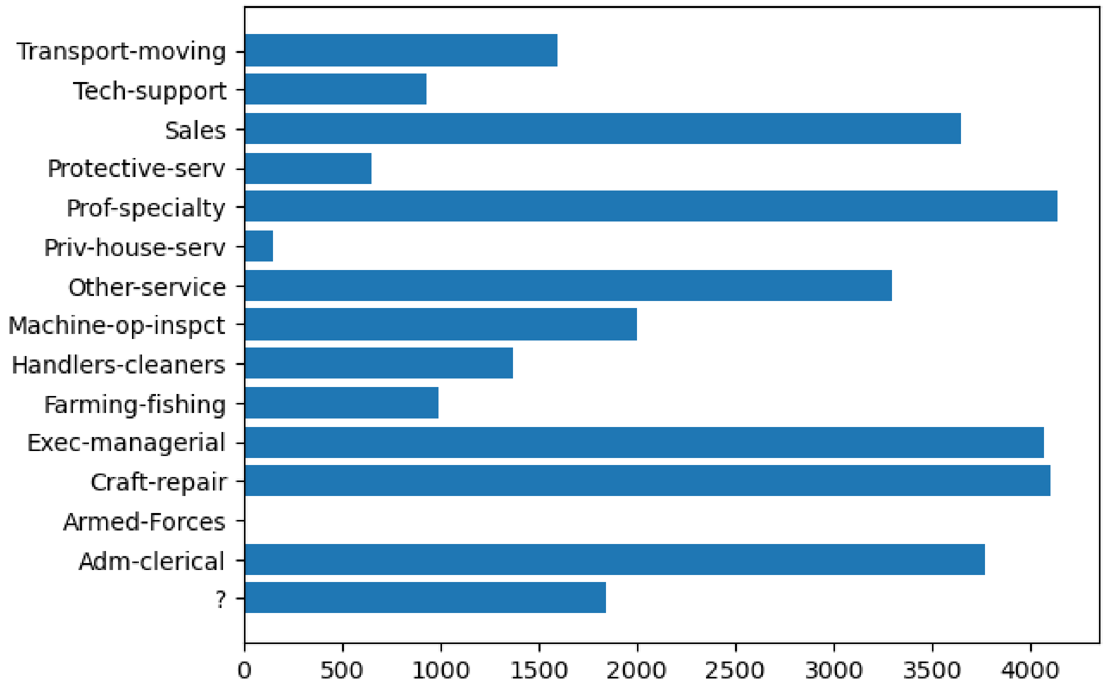
Use subset and groupby to find outliers

[35]:
```python
# Creates summary of occupation by age
occupation_describe = df3_subset.groupby('occupation').describe()['age']
```

Plot the values on a bar chart

[36]:
```python
# Creates Figure
plt.figure()
# Sets width and heights of the bars
plt.barh(y = occupation_describe.index, width = occupation_describe['count'])
```

```python
# Displays Plot
plt.show()
```



Outliers can exist where the are counts that don't meet the trend of the other data. For example, Armed Forces and Priv House Serv have very low counts in comparison to the other fields

Merge the data using common keys

[37]:
```python
df3_subset.head()
```

[37]:

	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

[38]:
```python
# Creates New Subset with only three columns
df5_subset = df3[['occupation', 'sex', 'Income']]
df5_subset.head()
```

[38]:

	occupation	sex	Income
0	Adm-clerical	Male	<=50K
1	Exec-managerial	Male	<=50K
2	Handlers-cleaners	Male	<=50K

```
3   Handlers-cleaners     Male   <=50K
4      Prof-specialty   Female   <=50K
```

[39]:
```python
# Inner join on two subset on occupation # HELP
joined_subset = pd.merge(df3_subset, df5_subset, on='occupation', how='left')
joined_subset
```

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

[32561 rows x 5 columns]
```

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

[40]:
```python
# Creates series 1 and 2 with the values and index
Series_1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
Series_2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1], index=['a', 'c', 'e', 'f', 'g'])
```

[41]:
```python
# Calculates sum of series 1 and series 2
series_sum = Series_1 + Series_2
series_sum
```

[41]:
```
a    5.2
c    1.1
d    NaN
e    0.0
f    NaN
g    NaN
```

```
dtype: float64
```

[42]:
```
# Subtracts series 1 from series 2
series_subtract = Series_1 - Series_2
series_subtract
```

[42]:
```
a    9.4
c   -6.1
d    NaN
e    3.0
f    NaN
g    NaN
dtype: float64
```

4. Data Wrangling with Python: Activity 7, page 207

Open the page in a separate tab and use something like an **Inspect Element** tool to view the source HTLM and understan its scructure.

Read the page using bs4

[43]:
```
# Import libraries to pull data from online source
import requests
from bs4 import BeautifulSoup

# Creates fd field with wiki data
fd = open("List of countries by GDP (nominal) - Wikipedia.htm", "r")

# Creates soup from the content
soup = BeautifulSoup(fd)

# Prints type of variable for soup
print(type(soup))
```

```
<class 'bs4.BeautifulSoup'>
```

Find the table structure you will need to deal with (how many tables are there?)

[44]:
```
# Finds all the tables
tables = soup.find_all('table')

# Print the number of tables
print("Number of tables: ", len(tables))
```

```
Number of tables:  9
```

Find the right page using bs4

[45]:
```
# Finds class of table
data_table = soup.find("table", {"class": '"wikitable"|}'})
```

```
print(type(data_table))
```

```
<class 'bs4.element.Tag'>
```

Separate the source names and their corresponding data

[46]:
```
# Prints the sources
sources = data_table.tbody.findAll('tr', recursive=False)[0]
sources_list = [td for td in sources.findAll('td')]
print(len(sources_list))
```

```
3
```

[47]:
```
# Displays sources
sources_list
```

[47]:
```
[<td style="width:33%; text-align:center;"><b>Per the <a
href="https://en.wikipedia.org/wiki/International_Monetary_Fund"
title="International Monetary Fund">International Monetary Fund</a>
(2017)</b><sup class="reference" id="cite_ref-GDP_IMF_1-2"><a href="https://en.w
ikipedia.org/wiki/List_of_countries_by_GDP_(nominal)#cite_note-
GDP_IMF-1">[1]</a></sup>
 </td>,
 <td style="width:33%; text-align:center;"><b>Per the <a
href="https://en.wikipedia.org/wiki/World_Bank" title="World Bank">World
Bank</a> (2017)</b><sup class="reference" id="cite_ref-worldbank_20-0"><a href="
https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)#cite_note-
worldbank-20">[20]</a></sup>
 </td>,
 <td style="width:33%; text-align:center;"><b>Per the <a
href="https://en.wikipedia.org/wiki/United_Nations" title="United
Nations">United Nations</a> (2016)</b><sup class="reference" id="cite_ref-21"><a
href="https://en.wikipedia.org/wiki/List_of_countries_by_GDP_(nominal)#cite_note
-21">[21]</a></sup><sup class="reference" id="cite_ref-22"><a href="https://en.w
ikipedia.org/wiki/List_of_countries_by_GDP_(nominal)#cite_note-22">[22]</a></sup
>
 </td>]
```

[48]:
```
# Gets data from data table
soup_data = data_table.tbody.findAll('tr', recursive=False)[1].findAll('td',␣
↪recursive=False)
```

[49]:
```
# Gets tables
data_tables = []
for td in soup_data:
    data_tables.append(td.findAll('table'))
```

[50]:
```
len(data_tables)
```

[50]: 3

Get the source names from the list of sources you have created

[51]:
```python
# Gets table names from data tables
source_names = [source.findAll('a')[0].getText() for source in sources_list]
print(source_names)
```

```
['International Monetary Fund', 'World Bank', 'United Nations']
```

Separate the header and the data that you separated for the first source only , and then create a DataFrame using that

[52]:
```python
# Gets column names from first table source
header1 = [th.getText().strip() for th in data_tables[0][0].findAll('thead')[0].
↪findAll('th')]
header1
```

[52]:
```
['Rank', 'Country', 'GDP(US$MM)']
```

[53]:
```python
# Collects rows from first table source
rows1 = data_tables[0][0].findAll('tbody')[0].findAll('tr')[1:]
```

[54]:
```python
data_rows1 = [[td.get_text().strip() for td in tr.findAll('td')] for tr in␣
↪rows1]
```

[55]:
```python
# Creates df from headers and rows
df1 = pd.DataFrame(data_rows1, columns=header1)
```

[56]:
```python
df1.head()
```

[56]:

	Rank	Country	GDP(US$MM)
0	1	United States	19,390,600
1	2	China[n 1]	12,014,610
2	3	Japan	4,872,135
3	4	Germany	3,684,816
4	5	United Kingdom	2,624,529

Repeat the task for the other two data sources

[57]:
```python
# Gets column names from the second table source
header2 = [th.getText().strip() for th in data_tables[1][0].findAll('thead')[0].
↪findAll('th')]
header2
```

[57]:
```
['Rank', 'Country', 'GDP(US$MM)']
```

[58]:
```
# Collects rows from second table source
rows2 = data_tables[1][0].findAll('tbody')[0].findAll('tr')[1:]
```

[59]:
```
data_rows2 = [[td.get_text().strip() for td in tr.findAll('td')] for tr in␣
↪rows2]
```

[60]:
```
# Creates df from headers and rows
df2 = pd.DataFrame(data_rows2, columns=header2)
```

[61]:
```
df2.head()
```

[61]:

	Rank	Country	GDP(US$MM)
0	1	United States	7007193906040000000 19,390,604
1		European Union[23]	7007172776980000000 17,277,698
2	2	China[n 4]	7007122377000000000 12,237,700
3	3	Japan	7006487213700000000 4,872,137
4	4	Germany	7006367743900000000 3,677,439

[62]:
```
# Gets column names from the second table source
header3 = [th.getText().strip() for th in data_tables[2][0].findAll('thead')[0].
↪findAll('th')]
header3
```

[62]: ['Rank', 'Country', 'GDP(US$MM)']

[63]:
```
# Collects rows from third data source
rows3 = data_tables[2][0].findAll('tbody')[0].findAll('tr')[1:]
```

[64]:
```
data_rows3 = [[td.get_text().strip() for td in tr.findAll('td')] for tr in␣
↪rows3]
```

[65]:
```
# Creates df from headers and rows
df3 = pd.DataFrame(data_rows3, columns=header3)
```

[66]:
```
df3.head()
```

[66]:

	Rank	Country	GDP(US$MM)
0	1	United States	7007186244750000000 18,624,475
1	2	China[n 4]	7007112182810000000 11,218,281
2	3	Japan	7006493621100000000 4,936,211
3	4	Germany	7006347779600000000 3,477,796
4	5	United Kingdom	7006264789800000000 2,647,898

5. Data Wrangling with Python: Activity 8, page 233

Read the **visit_data.csv** file

[67]:
```python
# Reads in the data
visit_data = pd.read_csv('visit_data.csv')
```

[68]:
```python
# Displays the first ten records
visit_data.head(10)
```

[68]:
```
   id  first_name   last_name                              email  gender  \
0   1       Sonny        Dahl                 sdahl0@mysql.com    Male
1   2         NaN         NaN                dhoovart1@hud.gov     NaN
2   3         Gar       Armal          garmal2@technorati.com     NaN
3   4     Chiarra      Nulty           cnulty3@newyorker.com     NaN
4   5         NaN         NaN      sleaver4@elegantthemes.com     NaN
5   6     Raymund  Ingerfield      ringerfield5@microsoft.com     NaN
6   7  Wilhelmina      Dagnan            wdagnan6@nytimes.com  Female
7   8         NaN         NaN  mdewilde7@creativecommons.org  Female
8   9      Gunter    Lisamore           glisamore8@disqus.com     NaN
9  10      Luelle    Scinelli             lscinelli9@issuu.com  Female

        ip_address   visit
0    135.36.96.183  1225.0
1  237.165.194.143   919.0
2   166.43.137.224   271.0
3   139.98.137.108  1002.0
4    46.117.117.27  2434.0
5   90.100.118.215   451.0
6    88.133.77.243  1540.0
7  229.215.244.227   537.0
8    134.185.44.82   743.0
9    160.130.58.61  1507.0
```

Check for duplicates

[69]:
```python
# identifies duplicates
dups = visit_data.duplicated()
```

[70]:
```python
# Adds duplicate identifier to dataset
visit_data['duplicate'] = dups.values
```

[71]:
```python
# Groups by duplicates to see how many true vs false
visit_data.groupby('duplicate').count()
```

[71]:
```
             id  first_name  last_name  email  gender  ip_address  visit
duplicate
False      1000         704        704   1000     495        1000    974
```

The results above indicate that there are no duplicate records, but this does not mean that there are not duplicate values within the columns themselves.

[72]:
```python
# Creates a functions that iterates through each column and checks if there is␣
↪duplicates
def check_duplicates(data):
    # Loop to run through columns
    for col in data.columns:
        # Statement if duplicates are true
        if data[col].duplicated().any():
            print(f"Column: {col} has duplicates")
        # Statement if duplicates are false
        else:
            print(f"Column: {col} does not have duplicates")

check_duplicates(visit_data)
```

```
Column: id does not have duplicates
Column: first_name has duplicates
Column: last_name has duplicates
Column: email does not have duplicates
Column: gender has duplicates
Column: ip_address does not have duplicates
Column: visit has duplicates
Column: duplicate has duplicates
```

Check if any essential column contains NaN

[73]:
```python
# Checks
print("First Name NaN: ", visit_data['first_name'].isna().sum())
print("Last Name NaN: ", visit_data['last_name'].isna().sum())
print("IP Address NaN: ", visit_data['ip_address'].isna().sum())
print("Visit Address NaN: ", visit_data['visit'].isna().sum())
```

```
First Name NaN:  296
Last Name NaN:  296
IP Address NaN:  0
Visit Address NaN:  26
```

[74]:
```python
# Creates a functions that iterates through each column and checks if there is␣
↪NaN
def check_na(data):
    # Loop to run through columns
    for col in data.columns:
        # Statement if NaN are true
        if data[col].isna().any():
            print(f"Column: {col} has NaN")
        # Statement if NaN are false
        else:
            print(f"Column: {col} does not have NaN")
```

```
check_na(visit_data)
```

```
Column: id does not have NaN
Column: first_name has NaN
Column: last_name has NaN
Column: email does not have NaN
Column: gender has NaN
Column: ip_address does not have NaN
Column: visit has NaN
Column: duplicate does not have NaN
```

Get rid of the outliers

[75]:
```
# Drops the null values
visit_data_no_na = visit_data.dropna()
visit_data_no_na
```

[75]:
```
      id  first_name   last_name                    email  gender  \
0      1       Sonny        Dahl         sdahl0@mysql.com    Male
6      7  Wilhelmina      Dagnan     wdagnan6@nytimes.com  Female
9     10      Luelle    Scinelli     lscinelli9@issuu.com  Female
12    13       Katya  Rewcassell  krewcassellc@dyndns.org  Female
17    18   Forrester   Randleson     frandlesonh@cnet.com    Male
..   …         …          …                       …      …
985  986      Aylmar     Bridson       abridsonrd@loc.gov    Male
987  988      Dinnie      Bendik    dbendikrf@samsung.com  Female
989  990   Cristabel     Vedyaev  cvedyaevrh@omniture.com  Female
992  993      Nancey     Goldsby       ngoldsbyrk@163.com  Female
995  996      Averil    Pickover       apickoverrn@vk.com    Male

          ip_address   visit  duplicate
0      135.36.96.183  1225.0      False
6      88.133.77.243  1540.0      False
9      160.130.58.61  1507.0      False
12     68.203.78.150   661.0      False
17   133.200.143.251   303.0      False
..               …       …         …
985   222.88.182.120   772.0      False
987     49.41.36.231  1616.0      False
989     48.203.12.64  2279.0      False
992    86.142.91.166  1455.0      False
995     10.45.16.167  1305.0      False

[337 rows x 8 columns]
```

Report the Size Difference

[76]:
```
# Shape before change
visit_data.shape
```

[76]: (1000, 8)

[77]:
```
# Shape after change
visit_data_no_na.shape
```

[77]: (337, 8)

The first table contained 1000 records and when we dropped all the NaN values we end up with 337 records

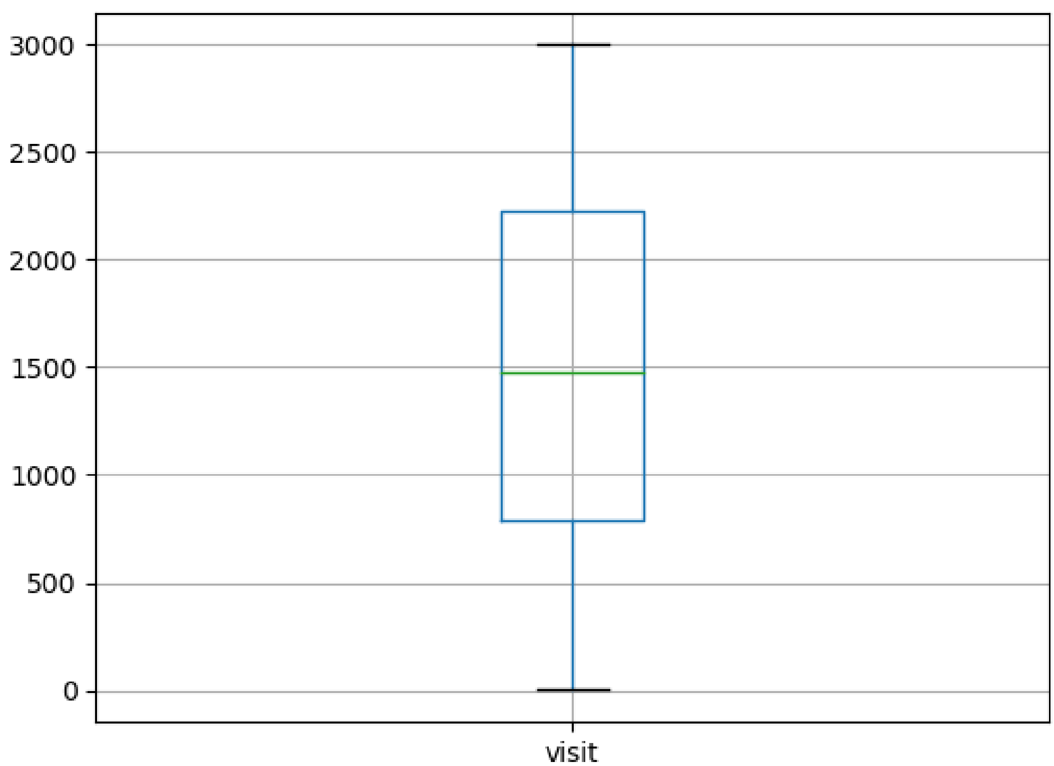
Create a box plot to check for outliers

[78]:
```
# Remove duplicate identifier since it is no longer needed
visit_data = visit_data.drop(columns='duplicate')
```

[79]:
```
from scipy import stats
```

[80]:
```
# Creates box plot of visit column
visit_data.boxplot(column='visit')
```

[80]: <AxesSubplot:>



When looking at this box plot, the top and bottom line represent the min and max that "Visit" has. The majority of visits lie within 650 and 2200 so some of the values of either close to either extremes can be considered outliers.

Get rid of any outliers

[81]:
```
from scipy import stats
```

[82]:
```
# Calculates Z score of visit
z_score = stats.zscore(visit_data['visit'], nan_policy='omit')

# Adds z_score to the dataframe
visit_data['z_score'] = z_score
visit_data
```

[82]:
```
       id first_name  last_name                          email  gender  \
0       1      Sonny       Dahl              sdahl0@mysql.com    Male
1       2        NaN        NaN            dhoovart1@hud.gov     NaN
2       3        Gar      Armal        garmal2@technorati.com     NaN
3       4    Chiarra      Nulty          cnulty3@newyorker.com     NaN
4       5        NaN        NaN      sleaver4@elegantthemes.com     NaN
..    …          …          …                             …     …
995   996     Averil   Pickover            apickoverrn@vk.com    Male
996   997     Walton  Hallewell          whallewellro@nasa.gov     NaN
997   998        NaN        NaN        ggallamorerp@meetup.com  Female
998   999   Sapphira     Terron        sterronrq@wordpress.org     NaN
999  1000        NaN        NaN  jandreuzzirr@paginegialle.it    Male

          ip_address   visit   z_score
0      135.36.96.183  1225.0 -0.325542
1    237.165.194.143   919.0 -0.690467
2     166.43.137.224   271.0 -1.463249
3     139.98.137.108  1002.0 -0.591484
4      46.117.117.27  2434.0  1.116269
..               …       …        …
995     10.45.16.167  1305.0 -0.230137
996  231.224.238.232  2531.0  1.231948
997     118.65.94.40     NaN       NaN
998    24.77.234.208   250.0 -1.488293
999   211.136.66.144  2389.0  1.062604

[1000 rows x 8 columns]
```

[83]:
```
visit_data.shape
```

[83]: (1000, 8)

[84]:
```
# Removes values that +-3 z_score
visit_data_no_outliers = visit_data[(visit_data['z_score'] > -3) &
 ↪(visit_data['z_score'] < 3)]
```

[85]:
```
# Shape of dataframe with no outliers
visit_data_no_outliers.shape
```

[85]: (974, 8)

Using z_scores we can get the data that is between -3 or 3 standard deviations from visit mean. This will remove any outliers from the data leaving us from 1000 records to 974.

6. Insert data into a SQL Lite database – create a table with the following data below that you will create yourself (Hint on how to create the SQL: Python for Data Analysis 2nd edition page 191, Python for Data Analysis 3rd Edition: Page 199):
   a. Name, Address, City, State, Zip, Phone Number
   b. Add at least 10 rows of data and submit your code with a query generating your results.

[86]:
```
# Import Required libraries
import sqlite3
```

[87]:
```
# Creates table and column names
query = """
    Create TABLE test
    (Name VARCHAR,
    Address VARCHAR,
    City VARCHAR,
    State VARCHAR,
    Zip INTEGER,
    PhoneNumber INTEGER
    );"""
```

[88]:
```
# Creates connection
connection = sqlite3.connect("mydata.sqlite")
```

[92]:
```
# Creating the data for the 6 columns
sql_data = [("Felipe Rodriguez", "111 M St", "Omaha", "NE", "68144",
 ↪"4025555555"),
       ("Maya Rodriguez", "111 M St", "Omaha", "NE", "68144", "4025555556"),
       ("Silvia Martinez", "112 M St", "Omaha", "NE", "68144", "4025555557"),
       ("Jessica Gomora", "113 M St", "Omaha", "NE", "68144", "4025555558"),
       ("Heraclio Alfonso", "114 M St", "Omaha", "NE", "68144", "4025555559"),
       ("Victor Martinez", "115 M St", "Omaha", "NE", "68144", "4025555560"),
       ("Panchita Rodriguez", "116 M St", "Omaha", "NE", "68144", "4025555561"),
       ("Rosa Castillo", "117 M St", "Omaha", "NE", "68144", "4025555562"),
```

```
        ("Tomiya Michiko", "118 M St", "Omaha", "NE", "68144", "4025555563"),
        ("Noah Heuertz", "119 M St", "Omaha", "NE", "68144", "4025555564")]
```

[93]:
```
# Creating command to insert data into table
stmt = "INSERT INTO test VALUES (?, ?, ?, ?, ?, ?)"
```

[94]:
```
# Exectutes insert of all rows
connection.executemany(stmt, sql_data)
```

[94]: <sqlite3.Cursor at 0x7fd0311c7dc0>

[95]:
```
# Commits changes made to SQL Database
connection.commit()
```

[96]:
```
# Selects all from test table
cursor = connection.execute("SELECT * FROM test")
```

[97]:
```
# Collects all rows
rows = cursor.fetchall()
```

[98]:
```
# Displays Rows
rows
```

[98]:
```
[('Felipe Rodriguez', '111 M St', 'Omaha', 'NE', 68144, 4025555555),
 ('Maya Rodriguez', '111 M St', 'Omaha', 'NE', 68144, 4025555556),
 ('Silvia Martinez', '112 M St', 'Omaha', 'NE', 68144, 4025555557),
 ('Jessica Gomora', '113 M St', 'Omaha', 'NE', 68144, 4025555558),
 ('Heraclio Alfonso', '114 M St', 'Omaha', 'NE', 68144, 4025555559),
 ('Victor Martinez', '115 M St', 'Omaha', 'NE', 68144, 4025555560),
 ('Panchita Rodriguez', '116 M St', 'Omaha', 'NE', 68144, 4025555561),
 ('Rosa Castillo', '117 M St', 'Omaha', 'NE', 68144, 4025555562),
 ('Tomiya Michiko', '118 M St', 'Omaha', 'NE', 68144, 4025555563),
 ('Noah Heuertz', '119 M St', 'Omaha', 'NE', 68144, 4025555564),
 ('Felipe Rodriguez', '111 M St', 'Omaha', 'NE', 68144, 4025555555),
 ('Maya Rodriguez', '111 M St', 'Omaha', 'NE', 68144, 4025555556),
 ('Silvia Martinez', '112 M St', 'Omaha', 'NE', 68144, 4025555557),
 ('Jessica Gomora', '113 M St', 'Omaha', 'NE', 68144, 4025555558),
 ('Heraclio Alfonso', '114 M St', 'Omaha', 'NE', 68144, 4025555559),
 ('Victor Martinez', '115 M St', 'Omaha', 'NE', 68144, 4025555560),
 ('Panchita Rodriguez', '116 M St', 'Omaha', 'NE', 68144, 4025555561),
 ('Rosa Castillo', '117 M St', 'Omaha', 'NE', 68144, 4025555562),
 ('Tomiya Michiko', '118 M St', 'Omaha', 'NE', 68144, 4025555563),
 ('Noah Heuertz', '119 M St', 'Omaha', 'NE', 68144, 4025555564)]
```

[99]:
```
# Creates Dataframe of data insert and adds column names based on table␣
 ↪description
pd.DataFrame(rows, columns=[x[0]for x in cursor.description])
```

[99]:

	Name	Address	City	State	Zip	PhoneNumber
0	Felipe Rodriguez	111 M St	Omaha	NE	68144	4025555555
1	Maya Rodriguez	111 M St	Omaha	NE	68144	4025555556
2	Silvia Martinez	112 M St	Omaha	NE	68144	4025555557
3	Jessica Gomora	113 M St	Omaha	NE	68144	4025555558
4	Heraclio Alfonso	114 M St	Omaha	NE	68144	4025555559
5	Victor Martinez	115 M St	Omaha	NE	68144	4025555560
6	Panchita Rodriguez	116 M St	Omaha	NE	68144	4025555561
7	Rosa Castillo	117 M St	Omaha	NE	68144	4025555562
8	Tomiya Michiko	118 M St	Omaha	NE	68144	4025555563
9	Noah Heuertz	119 M St	Omaha	NE	68144	4025555564
10	Felipe Rodriguez	111 M St	Omaha	NE	68144	4025555555
11	Maya Rodriguez	111 M St	Omaha	NE	68144	4025555556
12	Silvia Martinez	112 M St	Omaha	NE	68144	4025555557
13	Jessica Gomora	113 M St	Omaha	NE	68144	4025555558
14	Heraclio Alfonso	114 M St	Omaha	NE	68144	4025555559
15	Victor Martinez	115 M St	Omaha	NE	68144	4025555560
16	Panchita Rodriguez	116 M St	Omaha	NE	68144	4025555561
17	Rosa Castillo	117 M St	Omaha	NE	68144	4025555562
18	Tomiya Michiko	118 M St	Omaha	NE	68144	4025555563
19	Noah Heuertz	119 M St	Omaha	NE	68144	4025555564