Assignment_week_5 & 6_Raghuwanshi_Prashant_DSC540

October 9, 2021

Assignment: Week 5 & Week 6 Exercise, Data Formats/Data Structures/Data Sources

Name: Prashant Raghuwanshi

Date: 10/08/2021

Course: DSC540-T301 Data Preparation (2221-1) Complete the following exercises. You can submit a Jupyter Notebook or a PDF of your code. If you submit a .py file you need to also include a PDF or attachment of your results.

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

[1]:
```
# Import libraries
import pandas as pd
from bs4 import BeautifulSoup
```

[2]:
```
#reading the data from html file by using bs4
fd = open("C:/Users/dell/Documents/docker/List of countries by GDP (nominal) -␣
↪Wikipedia.htm", "r", encoding="utf8")
soup = BeautifulSoup(fd)
fd.close
```

[2]:
```
<function TextIOWrapper.close()>
```

[3]:
```
# Calculate the tables counts in html document:
all_tables_cnt = soup.find_all("table")
print("Total number of tables are {}".format(len(all_tables_cnt)))
```

```
Total number of tables are 9
```

[4]:
```
# list out right table by using class attribute
data_table = soup.find("table", {"class": '"wikitable"|}'})
print(type(data_table))
```

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

[5]:
```
#find out tr_elements
tr_elements = soup.find_all('table')[2].find_all('tr')
```

[6]:
```
# Seperate source and actual data
sources = data_table.tbody.findAll('tr', recursive=False)[0]
sources_list = [td for td in sources.find_all('td')]
print(len(sources_list))
```

```
3
```

[7]:
```
# using findAll functions to find the data from the data_table body tag
data = data_table.tbody.findAll('tr', recursive=False)[1].findAll('td',␣
 ↪recursive=False)
```

[8]:
```
# using findAll function to find out the data from data_table td
data_tables = []
for td in data:
    data_tables.append(td.findAll('table'))
# find the length of data table
len(data_tables)
```

[8]:
```
3
```

[9]:
```
# getting list of source name
sources_names = [sources.findAll('a')[0].getText() for sources in sources_list]
print(sources_names)
```

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

[10]:
```
# seperate the header and  data for the source name
header1 = [th.getText().strip() for th in data_tables[0][0].findAll('thead')[0].
 ↪findAll('th')]
header1
```

[10]:
```
['Rank', 'Country', 'GDP(US$MM)']
```

[11]:
```
#find the rows from data table using findAll
rows1 = data_tables[0][0].findAll('tbody')[0].findAll('tr')[1:]
```

[12]:
```
# find data from rows1 by using strip function for each td tag
data_rows1 = [[td.get_text().strip() for td in tr.findAll('td')] for tr in␣
 ↪rows1]
```

[13]:
```
#create dataframe
df1 = pd.DataFrame(data_rows1, columns=header1)
df1.head()
```

[13]:

	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
```

[14]:
```
# seperate the header and  data for the second source name
header2 = [th.getText().strip() for th in data_tables[1][0].findAll('thead')[0].
 ↪findAll('th')]
header2
```

[14]:
```
['Rank', 'Country', 'GDP(US$MM)']
```

[15]:
```
#find the rows from data table using findAll for source 2
rows2 = data_tables[1][0].findAll('tbody')[0].findAll('tr')[1:]
```

[16]:
```
# find data from rows1 by using strip function for each td tag
data_rows2_a = [[td.get_text().strip() for td in tr.findAll('td')] for tr in␣
 ↪rows2]
data_rows2_a
```

[16]:
```
[['1', 'United States', '7007193906040000000 19,390,604'],
 ['', 'European Union[23]', '7007172776980000000 17,277,698'],
 ['2', 'China[n 4]', '7007122377000000000 12,237,700'],
 ['3', 'Japan', '7006487213700000000 4,872,137'],
 ['4', 'Germany', '7006367743900000000 3,677,439'],
 ['5', 'United Kingdom', '7006262243400000000 2,622,434'],
 ['6', 'India', '7006259749100000000 2,597,491'],
 ['7', 'France', '7006258250100000000 2,582,501'],
 ['8', 'Brazil', '7006205550600000000 2,055,506'],
 ['9', 'Italy', '7006193479800000000 1,934,798'],
 ['10', 'Canada', '7006165304300000000 1,653,043'],
 ['11', 'Russia[n 2]', '7006157752400000000 1,577,524'],
 ['12', 'South Korea', '7006153075100000000 1,530,751'],
 ['13', 'Australia', '7006132342100000000 1,323,421'],
 ['14', 'Spain', '7006131132000000000 1,311,320'],
 ['15', 'Mexico', '7006114991900000000 1,149,919'],
 ['16', 'Indonesia', '7006101553900000000 1,015,539'],
 ['17', 'Turkey', '7005851102000000000 851,102'],
 ['18', 'Netherlands', '7005826200000000000 826,200'],
 ['19', 'Saudi Arabia', '7005683827000000000 683,827'],
 ['20', 'Switzerland', '7005678887000000000 678,887'],
 ['21', 'Argentina', '7005637590000000000 637,590'],
 ['22', 'Sweden', '7005538040000000000 538,040'],
 ['23', 'Poland', '7005524510000000000 524,510'],
 ['24', 'Belgium', '7005492681000000000 492,681'],
 ['25', 'Thailand', '7005455221000000000 455,221'],
 ['26', 'Iran', '7005439514000000000 439,514'],
 ['27', 'Austria', '7005416596000000000 416,596'],
 ['28', 'Norway', '7005398832000000000 398,832'],
 ['29', 'United Arab Emirates', '7005382575000000000 382,575'],
```

```
['30', 'Nigeria', '7005375771000000000 375,771'],
['31', 'Israel', '7005350851000000000 350,851'],
['32', 'South Africa', '7005349419000000000 349,419'],
['33', 'Hong Kong', '7005341449000000000 341,449'],
['34', 'Ireland', '7005333731000000000 333,731'],
['35', 'Denmark', '7005324872000000000 324,872'],
['36', 'Singapore', '7005323907000000000 323,907'],
['37', 'Malaysia', '7005314500000000000 314,500'],
['38', 'Philippines', '7005313595000000000 313,595'],
['39', 'Colombia', '7005309191000000000 309,191'],
['40', 'Pakistan', '7005304952000000000 304,952'],
['41', 'Chile', '7005277076000000000 277,076'],
['42', 'Finland', '7005251885000000000 251,885'],
['43', 'Bangladesh', '7005249724000000000 249,724'],
['44', 'Egypt', '7005235369000000000 235,369'],
['45', 'Vietnam', '7005223864000000000 223,864'],
['46', 'Portugal', '7005217571000000000 217,571'],
['47', 'Czech Republic', '70052157260000000000 215,726'],
['48', 'Romania', '7005211803000000000 211,803'],
['49', 'Peru', '7005211389000000000 211,389'],
['50', 'New Zealand', '7005205853000000000 205,853'],
['51', 'Greece', '7005200288000000000 200,288'],
['52', 'Iraq', '7005197716000000000 197,716'],
['53', 'Algeria', '7005170371000000000 170,371'],
['54', 'Qatar', '7005167605000000000 167,605'],
['55', 'Kazakhstan', '7005159407000000000 159,407'],
['56', 'Hungary', '7005139135000000000 139,135'],
['57', 'Angola', '7005124209000000000 124,209'],
['58', 'Kuwait', '7005120126000000000 120,126'],
['59', 'Sudan', '7005117488000000000 117,488'],
['60', 'Ukraine', '7005112154000000000 112,154'],
['61', 'Morocco[n 5]', '70051093139000000000 109,139'],
['62', 'Ecuador', '7005103057000000000 103,057'],
['63', 'Slovak Republic', '7004957690000000000 95,769'],
['64', 'Sri Lanka', '7004871750000000000 87,175'],
['65', 'Ethiopia', '7004805610000000000 80,561'],
['66', 'Dominican Republic', '70047593200000000000 75,932'],
['67', 'Guatemala', '7004756200000000000 75,620'],
['68', 'Kenya', '7004749380000000000 74,938'],
['69', 'Oman', '7004726430000000000 72,643'],
['70', 'Myanmar', '7004693220000000000 69,322'],
['71', 'Luxembourg', '7004624040000000000 62,404'],
['72', 'Panama', '7004618380000000000 61,838'],
['73', 'Costa Rica', '7004570570000000000 57,057'],
['74', 'Bulgaria', '7004568320000000000 56,832'],
['75', 'Uruguay', '7004561570000000000 56,157'],
['76', 'Croatia', '7004548490000000000 54,849'],
```

['77', 'Belarus', '7004544420000000000 54,442'],
['78', 'Tanzania[n 6]', '7004520900000000000 52,090'],
['79', 'Lebanon', '7004518440000000000 51,844'],
['80', 'Libya', '7004509840000000000 50,984'],
['81', 'Macau', '7004503610000000000 50,361'],
['82', 'Slovenia', '7004487700000000000 48,770'],
['83', 'Uzbekistan', '7004487180000000000 48,718'],
['84', 'Ghana', '7004473300000000000 47,330'],
['85', 'Lithuania', '7004471680000000000 47,168'],
['86', 'Turkmenistan', '7004423550000000000 42,355'],
['87', 'Serbia', '7004414320000000000 41,432'],
['88', 'Azerbaijan', '7004407480000000000 40,748'],
['89', "Cote d'Ivoire", '7004403890000000000 40,389'],
['90', 'Tunisia', '7004402570000000000 40,257'],
['91', 'Jordan', '7004400680000000000 40,068'],
['92', 'Bolivia', '7004375090000000000 37,509'],
['93', 'Democratic Republic of the Congo', '7004372410000000000 37,241'],
['94', 'Bahrain', '7004353070000000000 35,307'],
['95', 'Cameroon', '7004347990000000000 34,799'],
['96', 'Latvia', '7004302640000000000 30,264'],
['97', 'Paraguay', '7004297350000000000 29,735'],
['98', 'Estonia', '7004259210000000000 25,921'],
['99', 'Uganda', '7004258910000000000 25,891'],
['100', 'Zambia', '7004258090000000000 25,809'],
['101', 'El Salvador', '7004248050000000000 24,805'],
['102', 'Nepal', '7004244720000000000 24,472'],
['103', 'Iceland', '7004239090000000000 23,909'],
['104', 'Honduras', '7004229790000000000 22,979'],
['105', 'Cambodia', '7004221580000000000 22,158'],
['106', 'Trinidad and Tobago', '7004221050000000000 22,105'],
['107', 'Cyprus[n 7]', '7004216520000000000 21,652'],
['108', 'Papua New Guinea', '7004210890000000000 21,089'],
['109', 'Afghanistan', '7004208150000000000 20,815'],
['110', 'Bosnia and Herzegovina', '7004181690000000000 18,169'],
['111', 'Zimbabwe', '7004178460000000000 17,846'],
['112', 'Botswana', '7004174070000000000 17,407'],
['113', 'Laos', '7004168530000000000 16,853'],
['114', 'Senegal', '7004163750000000000 16,375'],
['115', 'Mali', '7004152880000000000 15,288'],
['116', 'Georgia[n 8]', '7004151590000000000 15,159'],
['117', 'Jamaica', '7004147680000000000 14,768'],
['118', 'Gabon', '7004146230000000000 14,623'],
['119', 'West Bank and Gaza', '7004144980000000000 14,498'],
['120', 'Nicaragua', '7004138140000000000 13,814'],
['121', 'Mauritius', '7004133380000000000 13,338'],
['122', 'Namibia', '7004132450000000000 13,245'],
['123', 'Albania', '7004130390000000000 13,039'],

['124', 'Burkina Faso', '7004128730000000000 12,873'],
['125', 'Malta', '7004125380000000000 12,538'],
['126', 'Equatorial Guinea', '7004124870000000000 12,487'],
['127', 'Mozambique', '7004123340000000000 12,334'],
['128', 'The Bahamas', '7004121620000000000 12,162'],
['129', 'Brunei', '7004121280000000000 12,128'],
['130', 'Armenia', '7004115370000000000 11,537'],
['131', 'Madagascar', '7004115000000000000 11,500'],
['132', 'Mongolia', '7004114880000000000 11,488'],
['133', 'Macedonia', '7004113380000000000 11,338'],
['134', 'Guinea', '7004104910000000000 10,491'],
['135', 'Chad', '7003998100000000000 9,981'],
['136', 'Benin', '7003927400000000000 9,274'],
['137', 'Rwanda', '7003913700000000000 9,137'],
['138', 'Republic of the Congo', '70038723000000000000 8,723'],
['139', 'Haiti', '7003840800000000000 8,408'],
['140', 'Moldova[n 9]', '7003812800000000000 8,128'],
['141', 'Niger', '7003812000000000000 8,120'],
['142', 'Kyrgyzstan', '7003756500000000000 7,565'],
['143', 'Somalia', '7003736900000000000 7,369'],
['144', 'Tajikistan', '7003714600000000000 7,146'],
['145', 'Kosovo', '7003712900000000000 7,129'],
['146', 'Malawi', '7003630300000000000 6,303'],
['147', 'Fiji', '7003506100000000000 5,061'],
['148', 'Mauritania', '7003502500000000000 5,025'],
['149', 'Togo', '7003481300000000000 4,813'],
['150', 'Barbados', '7003479700000000000 4,797'],
['151', 'Montenegro', '7003477400000000000 4,774'],
['152', 'Maldives', '7003459700000000000 4,597'],
['153', 'Swaziland', '7003440900000000000 4,409'],
['154', 'Sierra Leone', '7003377400000000000 3,774'],
['155', 'Guyana', '7003367600000000000 3,676'],
['156', 'Burundi', '7003347800000000000 3,478'],
['157', 'Suriname', '7003332400000000000 3,324'],
['158', 'Andorra', '7003301300000000000 3,013'],
['159', 'Timor-Leste', '7003295500000000000 2,955'],
['160', 'Lesotho', '7003263900000000000 2,639'],
['161', 'Bhutan', '7003251200000000000 2,512'],
['162', 'Liberia', '7003215800000000000 2,158'],
['163', 'Central African Republic', '70031949000000000000 1,949'],
['164', 'Djibouti', '7003184500000000000 1,845'],
['165', 'Belize', '7003183800000000000 1,838'],
['166', 'Cabo Verde', '7003175400000000000 1,754'],
['167', 'Saint Lucia', '7003171200000000000 1,712'],
['168', 'San Marino', '7003165900000000000 1,659'],
['169', 'Antigua and Barbuda', '70031532000000000000 1,532'],
['170', 'Seychelles', '7003148600000000000 1,486'],

```
['171', 'Guinea-Bissau', '7003134700000000000 1,347'],
['172', 'Solomon Islands', '7003130300000000000 1,303'],
['173', 'Grenada', '7003111900000000000 1,119'],
['174', 'The Gambia', '7003101500000000000 1,015'],
['175', 'Saint Kitts and Nevis', '7002946000000000000 946'],
['176', 'Vanuatu', '7002863000000000000 863'],
['177', 'Samoa', '7002857000000000000 857'],
['178', 'Saint Vincent and the Grenadines', '7002790000000000000 790'],
['179', 'Comoros', '7002649000000000000 649'],
['180', 'Dominica', '7002563000000000000 563'],
['181', 'Tonga', '7002426000000000000 426'],
['182', 'Sao Tome and Principe', '7002391000000000000 391'],
['183', 'Federated States of Micronesia', '7002336000000000000 336'],
['184', 'Palau', '7002292000000000000 292'],
['185', 'Marshall Islands', '7002199000000000000 199'],
['186', 'Kiribati', '7002196000000000000 196'],
['187', 'Nauru', '7002114000000000000 114'],
['188', 'Tuvalu', '7001400000000000000 40']]
```

[17]:
```
# Removing unwanted data values by using strip function
def find_right_text(i, td):
    if i == 0:
        return td.getText().strip()
    elif i == 1:
        return td.getText().strip()
    else:
        index = td.text.find(" ")
        return td.text[index+1:].strip()
```

[18]:
```
# find data from rows2 by using strip function for each td tag
data_rows2 = [[find_right_text(i, td) for i, td in enumerate(tr.findAll('td'))]␣
 ↪for tr in rows2]
```

[19]:
```
#create dataframe 2
df2 = pd.DataFrame(data_rows2, columns=header2)
df2.head()
```

[19]:

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

[20]:
```
# seperate the header and  data for the third source name
header3 = [th.getText().strip() for th in data_tables[2][0].findAll('thead')[0].
 ↪findAll('th')]
```

```
header3
```

[20]:
```
['Rank', 'Country', 'GDP(US$MM)']
```

[21]:
```
#find the rows from data table using findAll for source 3
rows3 = data_tables[2][0].findAll('tbody')[0].findAll('tr')[1:]
```

[22]:
```
# find data from rows3 by using strip function for each td tag
data_rows3 = [[td.get_text().strip() for td in tr.findAll('td')] for tr in␣
↪rows3]
```

[23]:
```
#create dataframe 3
df3 = pd.DataFrame(data_rows3, columns=header3)
df2.head()
```

[23]:

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

[ ]:
```

```

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

[24]:
```
# load libraries
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

[25]:
```
# loading data to dataframe
df_visit = pd.read_csv("C:/Users/dell/Documents/Packt-Data_Wrangling/Lesson 6/
↪visit_data.csv")
df_visit.head()
```

[25]:

	id	first_name	last_name	email	gender	ip_address	visit
0	1	Sonny	Dahl	sdahl0@mysql.com	Male	135.36.96.183	1225.0
1	2	NaN	NaN	dhoovart1@hud.gov	NaN	237.165.194.143	919.0
2	3	Gar	Armal	garmal2@technorati.com	NaN	166.43.137.224	271.0
3	4	Chiarra	Nulty	cnulty3@newyorker.com	NaN	139.98.137.108	1002.0
4	5	NaN	NaN	sleaver4@elegantthemes.com	NaN		

```
4    46.117.117.27  2434.0
```

[26]:
```
# find out the duplicates present in required columns
print("Is Duplicate data is present in first_name columns - {}".
 ↪format(any(df_visit.first_name.duplicated())))
print("Is Duplicate data is present in last_name columns - {}".
 ↪format(any(df_visit.last_name.duplicated())))
print("Is Duplicate data is present in email_name columns - {}".
 ↪format(any(df_visit.email.duplicated())))
```

```
Is Duplicate data is present in first_name columns - True
Is Duplicate data is present in last_name columns - True
Is Duplicate data is present in email_name columns - False
```

[27]:
```
# find out the null present in required columns
print(f"is null is present in email -- {df_visit.email.isnull().values.any()}")
print(f"is null is present in ipaddress -- {df_visit.ip_address.isnull().values.
 ↪any()}")
print(f"is null is present in visit -- {df_visit.visit.isnull().values.any()}")
```

```
is null is present in email -- False
is null is present in ipaddress -- False
is null is present in visit -- True
```

[28]:
```
save_old_shape = df_visit.shape
```

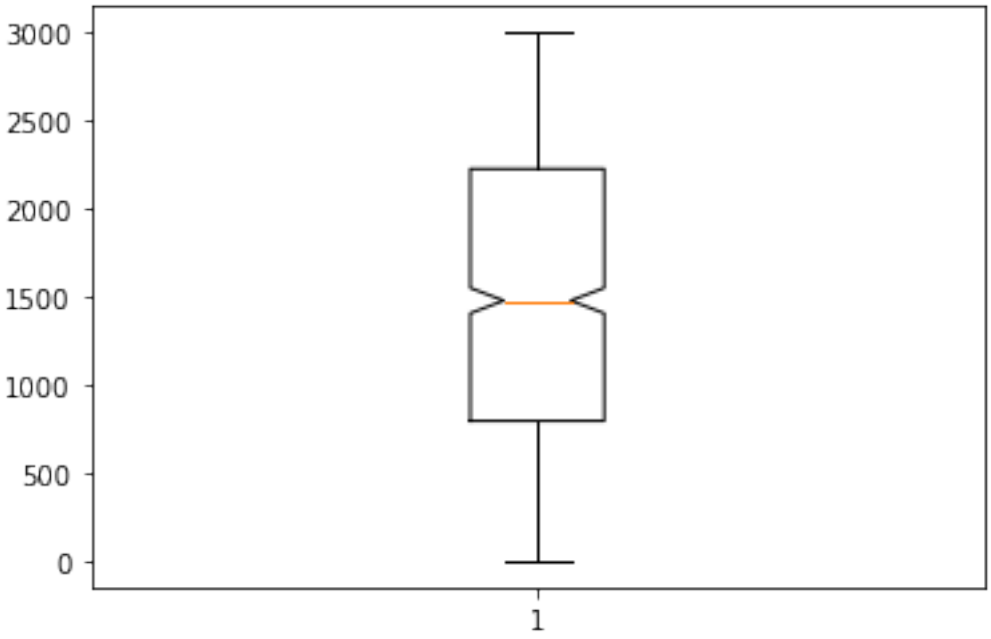
[29]:
```
# use numpy.isfinite() , fix if an element is finite or not.
df_visit = df_visit[np.isfinite(df_visit['visit'])]
```

[30]:
```
size_after = df_visit.shape
print(f"old size {save_old_shape} new size{size_after}")
```

```
old size (1000, 7) new size(974, 7)
```

[31]:
```
#plot boxplot to find outliers data
plt.boxplot(df_visit.visit, notch=True)
```

[31]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x18cb7320790>,
  <matplotlib.lines.Line2D at 0x18cb7320af0>],
 'caps': [<matplotlib.lines.Line2D at 0x18cb7320e50>,
  <matplotlib.lines.Line2D at 0x18cb73371f0>],
 'boxes': [<matplotlib.lines.Line2D at 0x18cb7320430>],
 'medians': [<matplotlib.lines.Line2D at 0x18cb7337550>],
 'fliers': [<matplotlib.lines.Line2D at 0x18cb73378b0>],
 'means': []}
```



herei found the lots of data points exist in between 700 to 2300, so removing outliers below 100 and above 2900

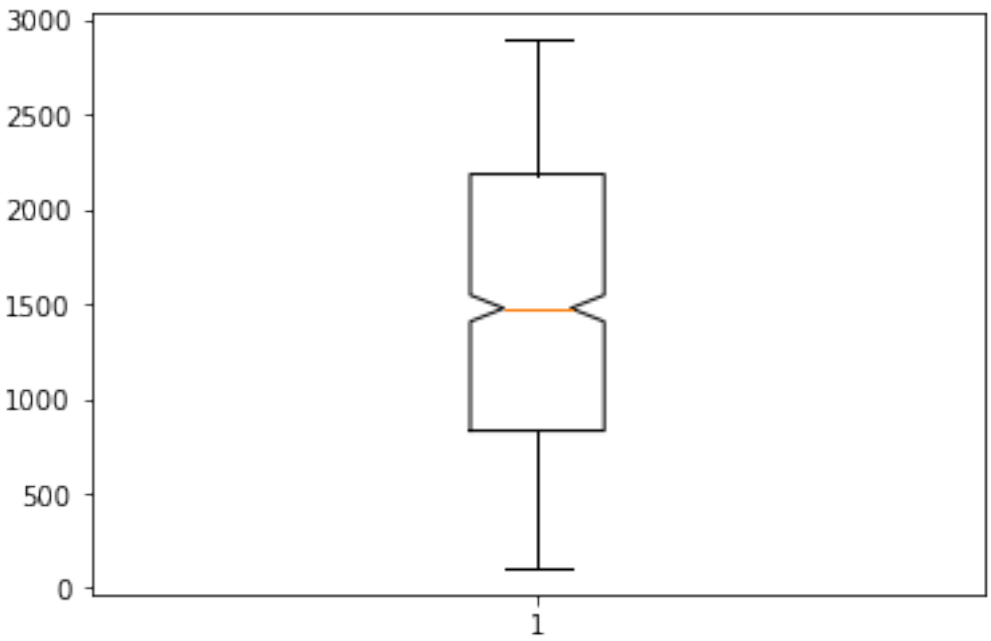
[32]:
```
df_visit_fix = df_visit[(df_visit['visit'] <= 2900) & (df_visit['visit'] >=␣
↪100)]
```

[33]:
```
# new shape of dh after fixing outliers
df_visit_fix.shape
```

[33]:
```
(923, 7)
```

[34]:
```
#plot boxplot to after fixing outliers data
plt.boxplot(df_visit_fix.visit, notch=True)
```

[34]:
```
{'whiskers': [<matplotlib.lines.Line2D at 0x18cb7459400>,
  <matplotlib.lines.Line2D at 0x18cb7459760>],
 'caps': [<matplotlib.lines.Line2D at 0x18cb7459ac0>,
  <matplotlib.lines.Line2D at 0x18cb7459e20>],
 'boxes': [<matplotlib.lines.Line2D at 0x18cb74590a0>],
 'medians': [<matplotlib.lines.Line2D at 0x18cb74641c0>],
 'fliers': [<matplotlib.lines.Line2D at 0x18cb7464520>],
 'means': []}
```



3. Insert data into a SQL Lite database – create a table with the following data (Hint: Python for Data Analysis page 191):

   a. Name, Address, City, State, Zip, Phone Number

[35]:
```
# import sqllite library
import sqlite3
```

[36]:
```
# create ddl statement variable
create_table_sql = """CREATE TABLE IF NOT EXISTS projects (
        Name VARCHAR[20],
        Address text NOT NULL,
        City VARCHAR[10],
        State  VARCHAR[10],
        Phone_Number VARCHAR[10]
);"""
```

[37]:
```
# establish connection with default sqllite connector
con = sqlite3.connect('mydata.sqlite')
```

[38]:
```
# execute ddl statement
con.execute(create_table_sql)
```

[38]:
```
<sqlite3.Cursor at 0x18cb74915e0>
```

[39]:
```
#Commit the transaction
con.commit()
```

b. Add at least 10 rows of data and submit your code with a query generating your results.

[40]:
```
# create data list contains sample data for projects table
data = [('Prashant', 'test add1', 'moline', 'IL', '123456789'), ('Prashant1',␣
 ↪'test add2', 'moline', 'IL', '123456789'), ('Prashant3', 'test add1',␣
 ↪'moline', 'IL', '123456789'), ('Prashant4', 'test add1', 'moline', 'IL',␣
 ↪'123456789'), ('Prashant5', 'test add1', 'moline', 'IL', '123456789'),␣
 ↪('Prashant6', 'test add1', 'moline', 'IL', '123456789'), ('Prashant7', 'test␣
 ↪add1', 'moline', 'IL', '123456789'), ('Prashant8', 'test add1', 'moline',␣
 ↪'IL', '123456789'), ('Prashant9', 'test add1', 'moline', 'IL', '123456789'),␣
 ↪('Prashant10', 'test add1', 'moline', 'IL', '123456789')]
```

[41]:
```
# create sql select statemet variable
sql_statement = "INSERT INTO projects VALUES(?, ?, ?, ?, ?)"
```

[42]:
```
# execute ddm statement
con.executemany(sql_statement, data)
```

[42]:
```
<sqlite3.Cursor at 0x18cb7491f80>
```

[43]:
```
#Commit the transaction
con.commit()
```

[44]:
```
# pull out records from table
cursor = con.execute('select * from projects')
#fetch all rows by using cursor
rows = cursor.fetchall()
# display the fetched data
rows
```

[44]:
```
[('Prashant', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant1', 'test add2', 'moline', 'IL', '123456789'),
 ('Prashant3', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant4', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant5', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant6', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant7', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant8', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant9', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant10', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant1', 'test add2', 'moline', 'IL', '123456789'),
 ('Prashant3', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant4', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant5', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant6', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant7', 'test add1', 'moline', 'IL', '123456789'),
 ('Prashant8', 'test add1', 'moline', 'IL', '123456789'),
```

```
('Prashant9', 'test add1', 'moline', 'IL', '123456789'),
('Prashant10', 'test add1', 'moline', 'IL', '123456789'),
('Prashant', 'test add1', 'moline', 'IL', '123456789'),
('Prashant1', 'test add2', 'moline', 'IL', '123456789'),
('Prashant3', 'test add1', 'moline', 'IL', '123456789'),
('Prashant4', 'test add1', 'moline', 'IL', '123456789'),
('Prashant5', 'test add1', 'moline', 'IL', '123456789'),
('Prashant6', 'test add1', 'moline', 'IL', '123456789'),
('Prashant7', 'test add1', 'moline', 'IL', '123456789'),
('Prashant8', 'test add1', 'moline', 'IL', '123456789'),
('Prashant9', 'test add1', 'moline', 'IL', '123456789'),
('Prashant10', 'test add1', 'moline', 'IL', '123456789')]
```