Arun18

October 17, 2024

[2]:

```
print("Demo of basic data types: Numbers")
x = 3
y = 2.5
print("x = ",x)
print("y = ",y)
print("Datatype of variable x: ",type(x))
print("Datatype of variable y: ",type(y))
print("Addition: ",x+y)
print("Subtraction: ",x-y)
print("Mutiplication: ",x*y)
print("Exponentiation: ",x**2)
```

```
Demo of basic data types: Numbers
x =  3
y =  2.5
Datatype of variable x:  <class 'int'>
Datatype of variable y:  <class 'float'>
Addition:  5.5
Subtraction:  0.5
Mutiplication:  7.5
Exponentiation:  9
```

[5]:

```
print("Demo of basic data types: Boolean")
t = True
f = False
print("t = ",t)
print("f = ",f)
print("Data type of variable t:",type(t))
print("Data type of variable f:",type(f))
print("Logical AND operation:",t and f)
print("Logical OR operation:",t or f)
print("Logical NOT operation:",not t)
print("Logical XOR operation:",t != f)
```

```
Demo of basic data types: Boolean
t =  True
f =  False
Data type of variable t: <class 'bool'>
Data type of variable f: <class 'bool'>
Logical AND operation: False
Logical OR operation: True
Logical NOT operation: False
Logical XOR operation: True
```

[10]:

```
print("Demo of basic data types: String")
s = "Hello"
t = "World"
print("String1 = ",s)
print("String2 = ",t)
d = s+","+t
print("String Concantenation:",d)
print("Capitalize: ",d.capitalize())
print("Converted to Uppercase: ",s.upper())
print("Right justify a string: ",s.rjust(7))
print("String at center: ",s.center(7))
print("After replacing l with ell:  ",s.replace('l','(ell)'))
print("String after striping leading to and trailling white spaces : ",'world '.
  ↪strip())
```

```
Demo of basic data types: String
String1 =  Hello
String2 =  World
String Concantenation: Hello,World
Capitalize:  Hello,world
Converted to Uppercase:  HELLO
Right justify a string:    Hello
String at center:   Hello
After replacing l with ell:   He(ell)(ell)o
String after striping leading to and trailling white spaces :  world
```

[16]:

```
print("Containers:Lists")
nums = list(range(5))
print("List 'nums' contains:",nums)
nums[4] ='abc'
print("List can contain elements of different types. Example: ",nums)
nums.append("xyz")
print("'nums' after inserting a new element a the end: ")
print("Sublists:")
print("A slice from index 2 to 4: ",nums[2:4])
print("A slice from index 2 to the end: ",nums[2:])
print("A slice from  start index to the end: ",nums[:2])
print("SA Slice of the whole list: ",nums[:])
nums[4:] = [8,9]
print("After assigning a new sublist to nums:")
for idx, i in enumerate(nums):
    print('%d:%s' %(idx+1, idx))
even_squares = [x**2 for x in nums if x%2==0]
print("List of squares of even numbers from 'nums'",even_squares)
```

```
Containers:Lists
List 'nums' contains: [0, 1, 2, 3, 4]
List can contain elements of different types. Example:  [0, 1, 2, 3, 'abc']
'nums' after inserting a new element a the end:
Sublists:
A slice from index 2 to 4:  [2, 3]
A slice from index 2 to the end:  [2, 3, 'abc', 'xyz']
A slice from  start index to the end:  [0, 1]
SA Slice of the whole list:  [0, 1, 2, 3, 'abc', 'xyz']
After assigning a new sublist to nums:
1:0
2:1
3:2
4:3
5:4
6:5
List of squares of even numbers from 'nums' [0, 4, 64]
```

[1]:

```
print("Containers:Dictionaries")
d= dict()
d = {'cat':'cute', 'dog':'furry'}
print("Dictionary: ",d)
print("Is the dictionary has the key 'cat'?",'cat' in d)
d['fish'] = 'wet'
print("After adding new entry to 'd': ",d)
print("Get an element 'monkey':", d.get('monkey',"N/A"))
print("Get an element 'fish':", d.get('fish',"N/A"))
del d['fish']
print("After deleting the newly added entry from 'd': ",d)
print("Demo of dictionary comprehension: ")
squares = {x:x*x for x in range(10)}
print("Squares of integers of range 10:")
for k,v in squares.items():
    print(k," ", v)
```

```
Containers:Dictionaries
Dictionary:  {'cat': 'cute', 'dog': 'furry'}
Is the dictionary has the key 'cat'? True
After adding new entry to 'd':  {'cat': 'cute', 'dog': 'furry', 'fish': 'wet'}
Get an element 'monkey': N/A
Get an element 'fish': wet
After deleting the newly added entry from 'd':  {'cat': 'cute', 'dog': 'furry'}
Demo of dictionary comprehension:
Squares of integers of range 10:
```

```
0   0
1   1
2   4
3   9
4   16
5   25
6   36
7   49
8   64
9   81
```

[7]:

```python
print("Containers:Sets")
num1 = {100,110,120}
print("Set'num1': ",num1)
num1.add(90)
print("'num1' after inserting 90: ",num1)
num1.update([50,60,70])
print("'num1' after inserting multiple elements: ",num1)
num1.remove(60)
print("'num1' after removing 60: ",num1)
print("Set comprehension and set options:")
n1 = {x for x in range(10)}
print("n1 = ",n1)
n2 = {x for x in range(10) if x%2!=0}
print("n2 = ",n2)
print("n1 union n2: ",n1|n2)
print("n1 intersection n2: ",n1&n2)
print("n1 difference n2: ",n1-n2)
```

```
Containers:Sets
Set'num1':  {120, 100, 110}
'num1' after inserting 90:  {120, 90, 100, 110}
'num1' after inserting multiple elements:  {100, 70, 110, 50, 120, 90, 60}
'num1' after removing 60:  {100, 70, 110, 50, 120, 90}
Set comprehension and set options:
n1 =  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
n2 =  {1, 3, 5, 7, 9}
n1 union n2:  {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
n1 intersection n2:  {1, 3, 5, 7, 9}
n1 difference n2:  {0, 2, 4, 6, 8}
```

[9]:

```python
print("CONTAINERS : TUPLES")
d = {(x,x+1):x for x in range(10)}
print("Dictionary with tuple keys: ")
for k,v in d.items():
    print(k," : ",v)
t = (5,6)
```

```
print("Tuple t: ",t)
print(d[t])
print(d[1,2])
```

```
CONTAINERS : TUPLES
Dictionary with tuple keys:
(0, 1)  :  0
(1, 2)  :  1
(2, 3)  :  2
(3, 4)  :  3
(4, 5)  :  4
(5, 6)  :  5
(6, 7)  :  6
(7, 8)  :  7
(8, 9)  :  8
(9, 10)  :  9
Tuple t:  (5, 6)
5
1
```

[10]:

```
print("Demo of function: Program to find factorial of a number")
def fact(n):
    if n == 1:
        return 1
    else:
        return(n*fact(n-1))
n = int(input("Enter a number: "))
print("Factorial: ",fact(n))
```

```
Demo of function: Program to find factorial of a number

Enter a number:  5

Factorial:  120
```

[13]:

```
class Greeter:
    def __init__(self,name):
        self.name = name
    def greet(self,loud=False):
        if loud:
            print('HELLO,%s!'%self.name.upper())
        else:
            print('Hello,%s'%self.name)
g = Greeter('Fred')
g.greet()
g.greet(loud=True)
```

```
Hello,Fred
HELLO,FRED!
```

[14]:

```
import numpy as np
a = np.array([1,2,3])
print("One Dimensional Array a: ",a)
b = np.array([[1,2,3],[4,5,6]])
print("Two Dimensional Array n: ",b)
print("Size of the Array: ", a.shape)
print("Elements at indices 0,1,2: ",a[0],a[1],a[2])
a[0]=5
print("Array after changing the element ar index 0: ",a)
a = np.zeros((2,2))
print("An array of all zeros: ",a)
b = np.ones((1,2))
print("An array of all ones: ",b)
c = np.full((2,2),7)
print("A constant array: ",c)
d = np.eye(2)
print("A 2*2 identity matrix: ",d)
e = np.random.random((2,2))
print("An array with random values: ",e)
```

```
One Dimensional Array a:  [1 2 3]
Two Dimensional Array n:  [[1 2 3]
 [4 5 6]]
Size of the Array:  (3,)
Elements at indices 0,1,2:  1 2 3
Array after changing the element ar index 0:  [5 2 3]
An array of all zeros:  [[0. 0.]
 [0. 0.]]
An array of all ones:  [[1. 1.]]
A constant array:  [[7 7]
 [7 7]]
A 2*2 identity matrix:  [[1. 0.]
 [0. 1.]]
An array with random values:  [[0.91225009 0.77410115]
 [0.52442513 0.48230674]]
```

[2]:

```
import numpy as np

print("Array indexing: slicing")
a1 = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]])
print("a1=", a1)

# Subarray consisting of first two rows and columns 1 and 2
b = a1[:2, 1:3]
print("Subarray consisting of first two rows and columns 1 and 2:", b)

# Subarray consists of second row
```

```python
b = a1[1:2, :]
print("Subarray consists of second row:", b)

# Accessing columns
print("Accessing columns:")
b = a1[:, 1]
print(b, b.shape)

c = a1[:, 1:2]
print(c, c.shape)

# Array integer indexing
print("Array integer indexing:")
a2 = np.array([[1, 2], [3, 4], [5, 6]])
print("a2=", a2)

# Example of array integer indexing
print("Example of array integer indexing:", a2[[0, 1, 2], [0, 1, 0]])
print(a2[[0, 0], [1, 1]])

# Using consistent variables
print(np.array([a2[0, 1], a2[0, 1]]))

# Assigning the array to a3 (no need to assign it to a)
a3 = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12]])
print("a3=", a3)

b = np.array([0, 2, 0, 1])
print("b=", b)
print("a3=", a3)

# Boolean array indexing
print("Boolean array indexing:")
a = np.array([[1, 2], [3, 4], [5, 6]])
print("a=", a)

bool_idx = (a > 2)
print("Elements greater than 2:", a[bool_idx])
```

```
Array indexing: slicing
a1= [[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
Subarray consisting of first two rows and columns 1 and 2: [[2 3]
 [6 7]]
Subarray consists of second row: [[5 6 7 8]]
Accessing columns:
```

```
[ 2  6 10] (3,)
[[ 2]
 [ 6]
 [10]] (3, 1)
Array integer indexing:
a2= [[1 2]
 [3 4]
 [5 6]]
Example of array integer indexing: [1 4 5]
[2 2]
[2 2]
a3= [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
b= [0 2 0 1]
a3= [[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
Boolean array indexing:
a= [[1 2]
 [3 4]
 [5 6]]
Elements greater than 2: [3 4 5 6]
```

[3]:

```python
import pandas as pd
orders = pd.read_table('http://bit.ly/movieusers')
print("Overview of dataframe :")
print(orders.head())
print("Shape: ", orders.shape)
print()
user_cols = ['use_id', 'age', 'gender', 'occupation', 'zip_code']
users = pd.read_table('https://bit.ly/movieusers',sep ='|',header = None,␣
 ↪names= user_cols)
print("Dataframe after modifying the default parameter values for read_table: ")
print(users.head())
```

```
Overview of dataframe :
   1|24|M|technician|85711
0       2|53|F|other|94043
1      3|23|M|writer|32067
2   4|24|M|technician|43537
3       5|33|F|other|15213
4    6|42|M|executive|98101
Shape:  (942, 1)

Dataframe after modifying the default parameter values for read_table:
   use_id  age gender  occupation zip_code
0       1   24      M  technician    85711
1       2   53      F       other    94043
2       3   23      M      writer    32067
3       4   24      M  technician    43537
4       5   33      F       other    15213
```

[3]:

```python
import pandas as pd
orders = pd.read_table('http://bit.ly/movieusers')
print("Overview of dataframe :")
print(orders.head())
print("Shape: ", orders.shape)
print()
user_cols = ['use_id', 'age', 'gender', 'occupation', 'zip_code']
users = pd.read_table('https://bit.ly/movieusers',sep ='|',header = None,␣
 ↪names= user_cols)
print("Dataframe after modifying the default parameter values for read_table: ")
print(users.head())
```

```
Overview of dataframe :
   1|24|M|technician|85711
0       2|53|F|other|94043
1      3|23|M|writer|32067
2  4|24|M|technician|43537
3       5|33|F|other|15213
4   6|42|M|executive|98101
Shape:  (942, 1)

Dataframe after modifying the default parameter values for read_table:
   use_id  age gender  occupation zip_code
0       1   24      M  technician    85711
1       2   53      F       other    94043
2       3   23      M      writer    32067
3       4   24      M  technician    43537
4       5   33      F       other    15213
```

[3]:

```python
import pandas as pd
orders = pd.read_table('http://bit.ly/movieusers')
print("Overview of dataframe :")
print(orders.head())
print("Shape: ", orders.shape)
print()
user_cols = ['use_id', 'age', 'gender', 'occupation', 'zip_code']
users = pd.read_table('https://bit.ly/movieusers',sep ='|',header = None,␣
 ↪names= user_cols)
print("Dataframe after modifying the default parameter values for read_table: ")
print(users.head())
```

```
Overview of dataframe :
   1|24|M|technician|85711
0       2|53|F|other|94043
1      3|23|M|writer|32067
2  4|24|M|technician|43537
3       5|33|F|other|15213
4   6|42|M|executive|98101
Shape:  (942, 1)

Dataframe after modifying the default parameter values for read_table:
   use_id  age gender  occupation zip_code
0       1   24      M  technician    85711
1       2   53      F       other    94043
2       3   23      M      writer    32067
3       4   24      M  technician    43537
4       5   33      F       other    15213
```

[8]:

```python
import pandas as pd #read a csv file
ufo = pdread_csv('https://bitly/uforeports')
print("Overview of UFO  data reports: ")
print(ufo.head())
print()
#series
print("Cityseries(sorted): ")
print(ufo.City.sort_values())
print()
ufo['Location'] = ufo.City+','+ufo.State
print("After creating a new 'Location' Series :")
print(ufo.head())
print()
print("Calculate summary statics: ")
print(ufo.describe())
print()
print("Column names of ufo dataframe :", ufo.columns)
print()
print("Column name of ufo dataframe after renaming two column names:ufo.
  ↪columns")
print()
print("Column name of ufo dataframe after removing two columns(city,state):␣
  ↪",ufo.columns)
print()
ufo.drop([0,1], axis = 0,inplace=True)
print("ufo dataframe after deleting first two rows: ")
print(ufo.head())
```

```
Overview of UFO  data reports:
                   City Colors Reported Shape Reported State             Time
0                Ithaca             NaN       TRIANGLE    NY   6/1/1930 22:00
```

```
1           Willingboro             NaN          OTHER    NJ  6/30/1930 20:00
2              Holyoke             NaN           OVAL    CO  2/15/1931 14:00
3              Abilene             NaN           DISK    KS   6/1/1931 13:00
4  New York Worlds Fair             NaN          LIGHT    NY  4/18/1933 19:00

Cityseries(sorted):
1761     Abbeville
17809     Aberdeen
2297      Aberdeen
9404      Aberdeen
389       Aberdeen
           …
12441          NaN
15767          NaN
15812          NaN
16054          NaN
16608          NaN
Name: City, Length: 18241, dtype: object

After creating a new 'Location' Series :
                   City Colors Reported Shape Reported State             Time  \
0                Ithaca             NaN       TRIANGLE    NY   6/1/1930 22:00
1           Willingboro             NaN          OTHER    NJ  6/30/1930 20:00
2               Holyoke             NaN           OVAL    CO  2/15/1931 14:00
3               Abilene             NaN           DISK    KS   6/1/1931 13:00
4  New York Worlds Fair             NaN          LIGHT    NY  4/18/1933 19:00

                  Location
0                Ithaca,NY
1           Willingboro,NJ
2               Holyoke,CO
3               Abilene,KS
4  New York Worlds Fair,NY

Calculate summary statics:
           City Colors Reported Shape Reported  State             Time  \
count     18215            2882          15597  18241            18241
unique     6475              27             27     52            16145
top     Seattle             RED          LIGHT     CA  11/16/1999 19:00
freq        187             780           2803   2529               27

          Location
count        18215
unique        8028
top     Seattle,WA
freq           187

Column names of ufo dataframe : Index(['City', 'Colors Reported', 'Shape
Reported', 'State', 'Time',
       'Location'],
      dtype='object')

Column name of ufo dataframe after renaming two column names:ufo.columns

Column name of ufo dataframe after removing two columns(city,state):
Index(['City', 'Colors Reported', 'Shape Reported', 'State', 'Time',
       'Location'],
      dtype='object')

ufo dataframe after deleting first two rows:
                   City Colors Reported Shape Reported State             Time  \
2               Holyoke             NaN           OVAL    CO  2/15/1931 14:00
3               Abilene             NaN           DISK    KS   6/1/1931 13:00
4   New York Worlds Fair             NaN          LIGHT    NY  4/18/1933 19:00
5           Valley City             NaN           DISK    ND  9/15/1934 15:30
6           Crater Lake             NaN         CIRCLE    CA   6/15/1935 0:00

                  Location
2               Holyoke,CO
3               Abilene,KS
4  New York Worlds Fair,NY
5           Valley City,ND
6           Crater Lake,CA
```

[14]:

```
import pandas as pd
movies = pd.read_csv('https://bit.ly//imdbratings')
print("dataframe of top-rated IMDb movies: ")
print(movies.head())
print()
print("Different ways to filter rows of a pandas Dataframe by column value: ")
print("Example: Filter rows to only show movies with a duration of atleast 200␣
  ↪minutes")
print("1.using for loop")
booleans = []
for length in movies.duration:
    if length >= 200:
        booleans.append(True)
    else:
        booleans.append(False)
is_long = pd.Series(booleans)
print(is_long.head())
print()
print("2.broadcasting")
print(movies[movies.duration>=200])
print()
```

```python
print("3.using 'loc' method")
print(movies.loc[movies.duration>=200])
```

```
dataframe of top-rated IMDb movies:
   star_rating                     title content_rating   genre  duration  \
0          9.3  The Shawshank Redemption              R   Crime       142   
1          9.2             The Godfather              R   Crime       175   
2          9.1    The Godfather: Part II              R   Crime       200   
3          9.0           The Dark Knight          PG-13  Action       152   
4          8.9              Pulp Fiction              R   Crime       154   

                                         actors_list  
0  [u'Tim Robbins', u'Morgan Freeman', u'Bob Gunt…  
1    [u'Marlon Brando', u'Al Pacino', u'James Caan']  
2  [u'Al Pacino', u'Robert De Niro', u'Robert Duv…  
3  [u'Christian Bale', u'Heath Ledger', u'Aaron E…  
4  [u'John Travolta', u'Uma Thurman', u'Samuel L…  

Different ways to filter rows of a pandas Dataframe by column value:
Example: Filter rows to only show movies with a duration of atleast 200 minutes
1.using for loop
0    False
1    False
2     True
3    False
4    False
dtype: bool

2.broadcasting
     star_rating                                          title  \
2            9.1                         The Godfather: Part II   
7            8.9  The Lord of the Rings: The Return of the King   
17           8.7                                  Seven Samurai   
78           8.4                    Once Upon a Time in America   
85           8.4                             Lawrence of Arabia   
142          8.3              Lagaan: Once Upon a Time in India   
157          8.2                             Gone with the Wind   
204          8.1                                        Ben-Hur   
445          7.9                           The Ten Commandments   
476          7.8                                         Hamlet   
630          7.7                                      Malcolm X   
767          7.6                It's a Mad, Mad, Mad, Mad World   

    content_rating      genre  duration  \
2                R      Crime       200   
7            PG-13  Adventure       201   
17         UNRATED      Drama       207   
```

```
78               R      Crime       229
85              PG  Adventure       216
142             PG  Adventure       224
157              G      Drama       238
204              G  Adventure       212
445       APPROVED  Adventure       220
476          PG-13      Drama       242
630          PG-13  Biography       202
767       APPROVED     Action       205

                                          actors_list
2    [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
7    [u'Elijah Wood', u'Viggo Mortensen', u'Ian McK…
17   [u'Toshir\xf4 Mifune', u'Takashi Shimura', u'K…
78   [u'Robert De Niro', u'James Woods', u'Elizabet…
85   [u"Peter O'Toole", u'Alec Guinness', u'Anthony…
142  [u'Aamir Khan', u'Gracy Singh', u'Rachel Shell…
157  [u'Clark Gable', u'Vivien Leigh', u'Thomas Mit…
204  [u'Charlton Heston', u'Jack Hawkins', u'Stephe…
445  [u'Charlton Heston', u'Yul Brynner', u'Anne Ba…
476  [u'Kenneth Branagh', u'Julie Christie', u'Dere…
630  [u'Denzel Washington', u'Angela Bassett', u'De…
767  [u'Spencer Tracy', u'Milton Berle', u'Ethel Me…

3.using 'loc' method
     star_rating                                          title  \
2            9.1                         The Godfather: Part II
7            8.9  The Lord of the Rings: The Return of the King
17           8.7                                  Seven Samurai
78           8.4                     Once Upon a Time in America
85           8.4                              Lawrence of Arabia
142          8.3              Lagaan: Once Upon a Time in India
157          8.2                              Gone with the Wind
204          8.1                                        Ben-Hur
445          7.9                          The Ten Commandments
476          7.8                                         Hamlet
630          7.7                                      Malcolm X
767          7.6                It's a Mad, Mad, Mad, Mad World

    content_rating      genre  duration  \
2                R      Crime       200
7            PG-13  Adventure       201
17         UNRATED      Drama       207
78               R      Crime       229
85              PG  Adventure       216
142             PG  Adventure       224
157              G      Drama       238
204              G  Adventure       212
```

```
445       APPROVED  Adventure       220
476          PG-13      Drama       242
630          PG-13  Biography       202
767       APPROVED     Action       205

                                           actors_list
2    [u'Al Pacino', u'Robert De Niro', u'Robert Duv…
7    [u'Elijah Wood', u'Viggo Mortensen', u'Ian McK…
17   [u'Toshir\xf4 Mifune', u'Takashi Shimura', u'K…
78   [u'Robert De Niro', u'James Woods', u'Elizabet…
85   [u"Peter O'Toole", u'Alec Guinness', u'Anthony…
142  [u'Aamir Khan', u'Gracy Singh', u'Rachel Shell…
157  [u'Clark Gable', u'Vivien Leigh', u'Thomas Mit…
204  [u'Charlton Heston', u'Jack Hawkins', u'Stephe…
445  [u'Charlton Heston', u'Yul Brynner', u'Anne Ba…
476  [u'Kenneth Branagh', u'Julie Christie', u'Dere…
630  [u'Denzel Washington', u'Angela Bassett', u'De…
767  [u'Spencer Tracy', u'Milton Berle', u'Ethel Me…
```

[15]:

```python
import pandas as pd
orders=pd.read_table('http://bit.ly/chiporders')
print("Dataframe:")
print(orders.head())
print()
print("String methods in pandas:")
print()
print("item_name' series(in uppercase):")
print(orders.item_name.str.upper().head())
print()
print("Checks for a substring 'Chicken' in the given dataframe:")
print(orders[orders.item_name.str.contains('Chicken')].head())
print()
print(orders.choice_description.str.replace('[', '').str.replace(']', '').
  ↪head())
print()
print("Examine the data type of each Series:")
print(orders.dtypes)
print()
print("Dataframe after replacing '$' and converting string to float of␣
  ↪'item_price' series:")
print(orders.item_price.str.replace('$', '').astype(float).head())
```

```
Dataframe:
   order_id  quantity                              item_name  \
0         1         1           Chips and Fresh Tomato Salsa
1         1         1                                   Izze
2         1         1                       Nantucket Nectar
3         1         1  Chips and Tomatillo-Green Chili Salsa
```

```
4         2         2                           Chicken Bowl

                                  choice_description item_price
0                                                NaN     $2.39
1                                       [Clementine]     $3.39
2                                            [Apple]     $3.39
3                                                NaN     $2.39
4  [Tomatillo-Red Chili Salsa (Hot), [Black Beans…    $16.98

String methods in pandas:

item_name' series(in uppercase):
0              CHIPS AND FRESH TOMATO SALSA
1                                      IZZE
2                          NANTUCKET NECTAR
3     CHIPS AND TOMATILLO-GREEN CHILI SALSA
4                              CHICKEN BOWL
Name: item_name, dtype: object

Checks for a substring 'Chicken' in the given dataframe:
    order_id  quantity             item_name  \
4          2         2          Chicken Bowl
5          3         1          Chicken Bowl
11         6         1  Chicken Crispy Tacos
12         6         1    Chicken Soft Tacos
13         7         1          Chicken Bowl

                                   choice_description item_price
4   [Tomatillo-Red Chili Salsa (Hot), [Black Beans…    $16.98
5   [Fresh Tomato Salsa (Mild), [Rice, Cheese, Sou…    $10.98
11  [Roasted Chili Corn Salsa, [Fajita Vegetables,…     $8.75
12  [Roasted Chili Corn Salsa, [Rice, Black Beans,…     $8.75
13  [Fresh Tomato Salsa, [Fajita Vegetables, Rice,…    $11.25

0                                                   NaN
1                                            Clementine
2                                                 Apple
3                                                   NaN
4     Tomatillo-Red Chili Salsa (Hot), Black Beans, …
Name: choice_description, dtype: object

Examine the data type of each Series:
order_id               int64
quantity               int64
item_name             object
choice_description    object
item_price            object
dtype: object
```

```
Dataframe after replacing '$' and converting string to float of 'item_price'
series:
0     2.39
1     3.39
2     3.39
3     2.39
4    16.98
Name: item_price, dtype: float64
```

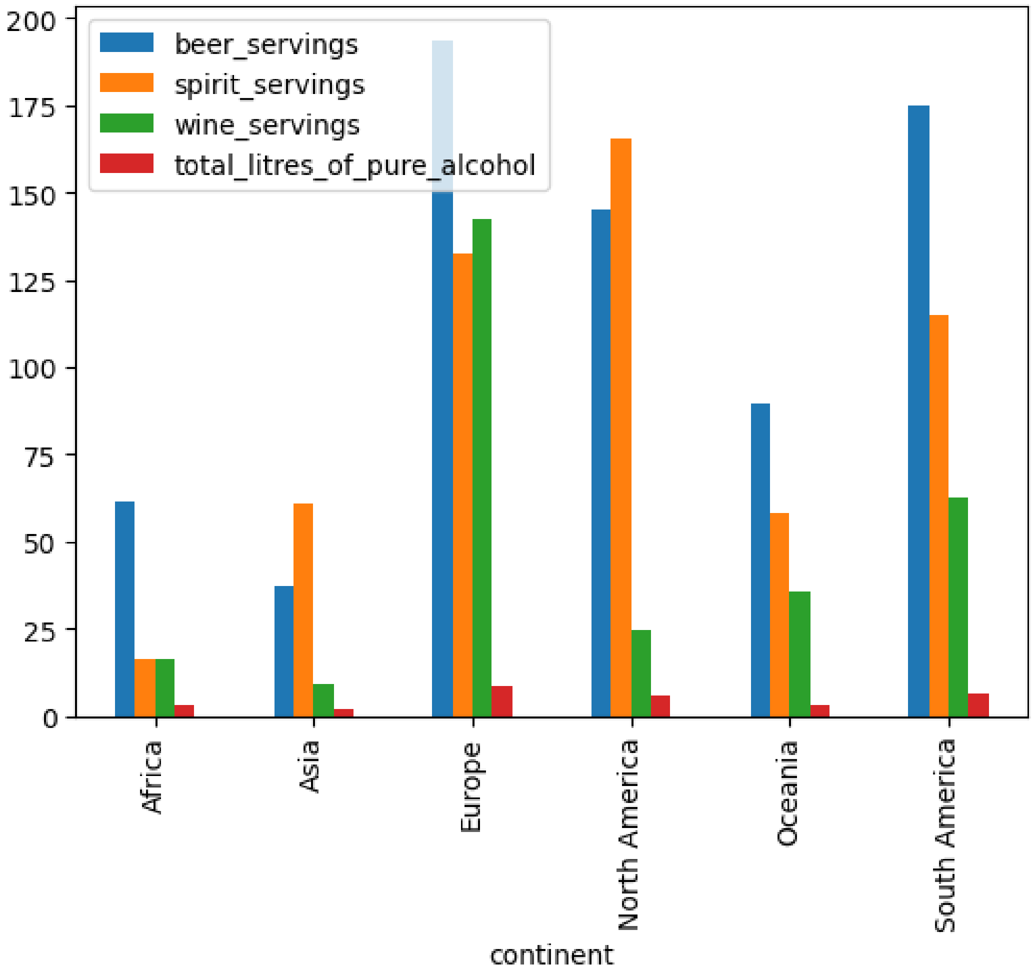
[2]:

```
import pandas as pd
import matplotlib.pyplot as plt

# Read a dataset of alcohol consumption into a DataFrame
drinks = pd.read_csv('http://bit.ly/drinksbycountry')
print("Dataframe:")
print(drinks.head())

# Calculate the mean beer servings across the entire dataset
print("\nMean beer servings across the entire dataset:",␣
  ↪drinks['beer_servings'].mean())

# Calculate the mean beer servings just for countries in Africa
print("\nMean beer servings just for countries in Africa:",␣
  ↪drinks[drinks['continent'] == 'Africa']['beer_servings'].mean())

# Aggregate functions used with groupby
print("\nMean beer servings for each continent:", drinks.
  ↪groupby('continent')['beer_servings'].mean())
print("\nMaximum beer servings for each continent:", drinks.
  ↪groupby('continent')['beer_servings'].max())

# Multiple aggregation functions applied simultaneously
print("\nMultiple aggregation functions can be applied simultaneously:")
print(drinks.groupby('continent')['beer_servings'].agg(['count', 'mean', 'min',␣
  ↪'max']))

# Group by 'continent' and calculate mean for all numerical columns
print("\nMean for all numerical columns grouped by continent (excluding␣
  ↪non-numeric columns):")
print(drinks.groupby('continent').mean(numeric_only=True))

# Allow plots to appear in the notebook
%matplotlib inline

# Side-by-side bar plot of the DataFrame directly above
drinks.groupby('continent').mean(numeric_only=True).plot(kind='bar')
```

```
plt.show()
```

```
Dataframe:
       country  beer_servings  spirit_servings  wine_servings  \
0  Afghanistan              0                0              0
1      Albania             89              132             54
2      Algeria             25                0             14
3      Andorra            245              138            312
4       Angola            217               57             45

   total_litres_of_pure_alcohol continent
0                           0.0      Asia
1                           4.9    Europe
2                           0.7    Africa
3                          12.4    Europe
4                           5.9    Africa

Mean beer servings across the entire dataset: 106.16062176165804

Mean beer servings just for countries in Africa: 61.471698113207545

Mean beer servings for each continent: continent
Africa            61.471698
Asia              37.045455
Europe           193.777778
North America    145.434783
Oceania           89.687500
South America    175.083333
Name: beer_servings, dtype: float64

Maximum beer servings for each continent: continent
Africa           376
Asia             247
Europe           361
North America    285
Oceania          306
South America    333
Name: beer_servings, dtype: int64

Multiple aggregation functions can be applied simultaneously:
               count        mean  min  max
continent
Africa            53   61.471698    0  376
Asia              44   37.045455    0  247
Europe            45  193.777778    0  361
North America     23  145.434783    1  285
Oceania           16   89.687500    0  306
```

```
South America     12  175.083333   93  333

Mean for all numerical columns grouped by continent (excluding non-numeric
columns):
               beer_servings  spirit_servings  wine_servings  \
continent
Africa             61.471698        16.339623      16.264151
Asia               37.045455        60.840909       9.068182
Europe            193.777778       132.555556     142.222222
North America     145.434783       165.739130      24.521739
Oceania            89.687500        58.437500      35.625000
South America     175.083333       114.750000      62.416667

               total_litres_of_pure_alcohol
continent
Africa                             3.007547
Asia                               2.170455
Europe                             8.617778
North America                      5.995652
Oceania                            3.381250
South America                      6.308333
```


200
175
150
125
100
75
50
25
0
beer_servings
spirit_servings
wine_servings
total_litres_of_pure_alcohol
Africa
Asia
Europe
North America
Oceania
South America
continent

```
[ ]:
```