C.K.Pithawala College of Engineering and Technology, Surat

Subject: Python for Data Science(3150713)

Practical Assignment File

Computer Engineering Department

By: Jash Kabir Mehta

By: Joy Anilbhai Toor

Div:B

Enrollment: 200090107052

Enrollment: 200090107014

Dataset: Summer Olympics 1896-2012 Athens City

Dataset link

CO-1 ASSIGNMENT:

Q.1 Write a program to implement the following using operators in Python. start no=110, result=550. Store operators(+,-,*,/,1/x,%,// etc.) in a list. Take a random operator from the list, apply the operation on the start no and result. Store answer in result variable. Take input for a random operator till it is valid input. [Use import random print(random.randint(3, 9))]

```
#end row of dataset is result
#end row/2 is start n
startno=31166
result=15583
operatorlist=['+','-','*','/','//']

import random
while(True):
  r=random.randint(0,10)
  print("operand:",r)
  if(r>=len(operatorlist)):
    break
  else:
    s=str(startno)+str(operatorlist[r])+str(result)
    print(s)
    operation=eval(s)
    print(operation)
```

```
operand: 3
31166/15583
2.0
operand: 10
```

Q.2

Write a program to create the following pattern. Take input for n. for n=5. [ord()-for character to ascii, chr()-for ascii to char]

A AB ABC ABCDEF ABCDEFGHIJKL

```
n=int(input("enter no of rows:"))
i=1
a=1
for i in range(n):
  for j in range(a):
    print(chr(65+j),end="")
  print()
```

```
  if(a==1):
    a=2
  else:
    while(i!=0):
      a=a+i
      i=i-1
```

```
enter no of rows:5
A
AB
ABC
ABCDEF
ABCDEFGHIJKL
```

Q3 Use list comprehension to create the following output for your dataset items.

E.g.my dataset if of car car=[“Swift”, ”BMW”, “Skoda5”, “i10”, “Lamborghini”] Create a new list of indices where the car name has a number in the name. Output for the example: [2, 3]

```
import re
shows=["Athens City", "Gold 2", "Athletic Game", "Swimming pool 3", "1987", "6 silver"]
print("List of shows : " + str(shows))
indices=[i for i,x in enumerate(shows) if re.search("[0-9]+",x)]
print("Indices where shows have a number in the name : " + str(indices))
```

```
List of shows : ['Athens City', 'Gold 2', 'Athletic Game', 'Swimming pool 3', '1987', '6 silver']
Indices where shows have a number in the name : [1, 3, 4, 5]
```

Double-click (or enter) to edit

Q4: Write a program to make a module. Implement (a)method overloading (b)method overriding (c)subclass (d)multilevel inheritance (e)multiple inheritance. Use the classes, properties and methods according to your data set. Import it in other program

```
#Method Overloading:
class athens:
    def olympics(self,*args):
        print("olympics : ")
        for i in args:
            print(i)
        print()

n=athens()
n.olympics("Swimming pool")
n.olympics("Silver","Gold")
n.olympics("summer olympics","has so many games","and several different medals")
```

```
olympics :
Swimming pool

olympics :
Silver
Gold

olympics :
summer olympics
has so many games
and several different medals
```

```
#Method Overriding:
class athens:
    def olympics(self):
        print("Many games.")

class city:
```

```
    def olympics(self):
        print("I love summer olympics.")

a=athens()
b=city()
a.olympics()
b.olympics()
```

```
    Many games.
    I love summer olympics.
```

```
#subclasses
class athens():
    def __init__(self,medal,year):
        self.medal=medal
        self.year=year
    def display1(self):
        print("athens:\n medal: "+str(self.medal)+"\n year : "+str(self.year)+"\n")
class athens_city(athens):
    def display2(self):
      print("athens_city:\n medal: "+str(self.medal)+"\n year : "+str(self.year)+"\n")


show=athens_city("Gold","1896")

show.display1()
show.display2()
```

```
    athens:
     medal: Gold
     year : 1896

    athens_city:
     medal: Gold
     year : 1896
```

```
#Multilevel Inheritance:
class athens_city:
    def __init__(self,city,activity):
        self.city=city
        self.activity=activity
    def title_info(self):
        print("Athens_city :\ncity : "+str(self.city)+"\nactivity : "+str(self.activity)+"\n")

class olympics(athens_city):
    def activity_info(self):
      print("olympics :\ncity : "+str(self.city)+"\nactivity : "+str(self.activity)+"\n")


class tv_shows(olympics):
    def show_info(self):
        print("tv_shows :\ncity : "+str(self.city)+"\nactivity : "+str(self.activity)+"\n")

show=tv_shows("Paris","Horse Ridding")
show.title_info()
show.activity_info()
show.show_info()
```

```
    Athens_city :
    city : Paris
    activity : Horse Ridding

    olympics :
    city : Paris
    activity : Horse Ridding

    tv_shows :
    city : Paris
```

```
    activity : Horse Ridding
```

```
#Multiple Inheritance:
class athens_city:
    def __init__(self,city,activity):
        self.city=city
        self.activity=activity
    def first_info(self):
        print("Athens_city :\ncity : "+str(self.city)+"\nactivity : "+str(self.activity)+"\n")


class olympics:
  def __init__(self,city,activity):
        self.city=city
        self.activity=activity
  def second_info(self):
        print("olympics :\ncity : "+str(self.city)+"\nactivity : "+str(self.activity)+"\n")

class summer(athens_city,olympics):
    pass

a=summer("Rome","Swimming")
a.first_info()
a.second_info()

b=summer("Italy","High Jump")
b.first_info()
b.second_info()
```

```
    Athens_city :
    city : Rome
    activity : Swimming

    olympics :
    city : Rome
    activity : Swimming

    Athens_city :
    city : Italy
    activity : High Jump

    olympics :
    city : Italy
    activity : High Jump
```

5 Write a program to insert, delete, update, retrieve, indexing, slicing, concatenation, join etc. for string(s) from data of your dataset

```
def insert():
    global str1
    print("String before insertion : " + str1)

    sub=input("Enter the string for insertion : ")
    pos=int(input("Enter position for insertion: "))

    temp=str1[:pos]+sub+str1[pos:]
    str1=temp

    print("String after insertion : " + str1)

def delete():
    global str1
    print("String before delete : " + str1)
    str1=""
    print("String after delete : " + str1)
```

```
def update():
    global str1
    print("String before updating : " + str1)
    str1=input("Enter the updated string : ")
    print("String after updating : " + str1)

def retrieve():
    global str1
    print("String retrieved : " + str1)

def indexing():
    global str1
    ch=input("Enter the string to get its index : ")
    print("Index of "+ch+" is : "+str(str1.index(ch)))

def slicing():
    global str1
    start=int(input("Enter start point : "))
    end=int(input("Enter end point : "))
    step=int(input("Enter step point : "))
    print(str1[start:end:step])

def concatenation():
    global str1
    print("String before concatenation : " + str1)
    temp=input("Enter a string for concatenation : ")
    str1+=temp
    print("String after concatenation : " + str1)

def join_str():
    global str1
    print("String before join : " + str1)
    str2=input("Enter join character for string : ")
    str1=str2.join(str1)
    print("String after join : " + str1)

def switch(argument):
    switcher={
        "1":insert,
        "2":delete,
        "3":update,
        "4":retrieve,
        "5":indexing,
        "6":slicing,
        "7":concatenation,
        "8":join_str,
    }
    r=switcher.get(argument)()

str1="Athens city summer olympics"
print("String : " + str1)
print()

print("""Select the operation from the following:
1. Insert
2. Delete
3. Update
4. Retrieve
5. Indexing
6. Slicing
7. Concatenation
```

```
8. Join
""")
arg=input("Enter numbers to select operation : ")
print()

switch(arg)
```

```
String : Athens city summer olympics

    Select the operation from the following:
    1. Insert
    2. Delete
    3. Update
    4. Retrieve
    5. Indexing
    6. Slicing
    7. Concatenation
    8. Join

    Enter numbers to select operation : 1

    String before insertion : Athens city summer olympics
    Enter the string for insertion : jkm
    Enter position for insertion: 7
    String after insertion : Athens jkmcity summer olympics
```

Double-click (or enter) to edit

6.Write a program to insert, delete, update, retrieve, indexing, slicing, concatenation, join etc. for list(s) from data of your dataset.

```
def insert():
    global list1
    sub=input("Enter the string for insertion : ")
    pos=int(input("Enter the position for insertion : "))
    list1.insert(pos,sub)

def delete():
    global list1
    list1.remove(input("Enter the element to delete : "))

def update():
    global list1
    index=list1.index(input("Enter the element to be replaced : "))
    list1[index]=input("Enter the new element : ")

def retrieve():
    global list1
    print(list1)

def indexing():
    global list1
    print(list1.index(input("Enter the string to get its index : ")))

def slicing():
    global list1
    start=int(input("Enter the start position : "))
    end=int(input("Enter the end position : "))
    step=int(input("Enter step : "))
    print(list1[start:end:step])

def concatenation():
    global list1
    list1.append(input("Enter element to be concatenated : "))

def join_list():
    global list1
```

```
    print("List after join : ")
    print(input("Enter the element for join : ").join(list1))


list1=["1896","Athens","Athletics","Gold"]
print("List : " + str(list1) + "\n")


print("""Select the operation from the following:
1. Insert
2. Delete
3. Update
4. Retrieve
5. Indexing
6. Slicing
7. Concatenation
8. Join
""")
arg=input("Enter numbers to select operation : ")
print(arg)

if arg=="1":
    print("List before insertion : " + str(list1))
    insert()
    print("List after insertion : " + str(list1))
elif arg=="2":
    print("List before deletion : " + str(list1))
    delete()
    print("List after deletion : " + str(list1))
elif arg=="3":
    print("List before updating : " + str(list1))
    update()
    print("List after updating : " + str(list1))
elif arg=="4":
    print("List retrieved : ")
    retrieve()
elif arg=="5":
    indexing()
elif arg=="6":
    slicing()
elif arg=="7":
    print("List before concatenation : " + str(list1))
    concatenation()
    print("List after concatenation : " + str(list1))
elif arg=="8":
    print("List before join : " + str(list1))
    join_list()
else:
    print("Invalid Operation")

List=["1896","Athens","Athletics","Gold"]
```

```
List : ['1896', 'Athens', 'Athletics', 'Gold']

Select the operation from the following:
1. Insert
2. Delete
3. Update
4. Retrieve
5. Indexing
6. Slicing
7. Concatenation
8. Join

Enter numbers to select operation : 1
1
List before insertion : ['1896', 'Athens', 'Athletics', 'Gold']
```

```
Enter the string for insertion : kk
Enter the position for insertion : 2
List after insertion : ['1896', 'Athens', 'kk', 'Athletics', 'Gold']
```

Double-click (or enter) to edit

7.Write a program to insert, delete, update, retrieve, indexing, slicing, concatenation, join etc. for tuple(s) from data of your dataset.

```
def tup_insert():
    global tuple1
    print("Tuple before insertion : " + str(tuple1))
    tuple1=list(tuple1)
    tuple1.insert(int(input("Enter position for insertion")),input("Enter the string for inserti
    tuple1=tuple(tuple1)
    print("Tuple after insertion : " + str(tuple1))

def tup_delete():
    global tuple1
    print("Tuple before deletion : " + str(tuple1))
    tuple1=list(tuple1)
    tuple1.remove(input("Enter the element to be deleted : "))
    tuple1=tuple(tuple1)
    print("Tuple after deletion : " + str(tuple1))

def tup_update():
    global tuple1
    print("Tuple before updating : " + str(tuple1))
    tuple1=list(tuple1)
    index=tuple1.index(input("Enter the element to be replaced : "))
    tuple1[index]=input("Enter the new element : ")
    print("Tuple after updating : " + str(tuple1))

def tup_retrieve():
    global tuple1
    print("Tuple retrieved : " + str(tuple1))

def tup_indexing():
    global tuple1
    print(tuple1.index(input("Enter the element to get its index : ")))

def tup_slicing():
    global tuple1
    start=int(input("Enter the start position : "))
    end=int(input("Enter the end position : "))
    step=int(input("Enter step : "))
    print(tuple1[start:end:step])

def tup_concatenation():
    global tuple1
    tuple1=list(tuple1)
    print("Tuple before concatenation : " + str(tuple1))
    tuple1.append(input("Enter the element to be concatenated : "))
    tuple1=tuple(tuple1)
    print("Tuple after concatenation : " + str(tuple1))

tuple1=('Athens','city','swimming','1986','gold')
print("Tuple : " + str(tuple1) + "\n")


print("""Select the operation from the following:
1. Insert
2. Delete
```

```
3. Update
4. Retrieve
5. Indexing
6. Slicing
7. Concatenation
""")
arg=input("Enter numbers to select operation : ")
print(arg)
if arg=="1":
    print("Tuple before insertion : " + str(tuple1))
    tup_insert()
    print("Tuple after insertion : " + str(tuple1))
elif arg=="2":
    print("Tuplebefore deletion : " + str(tuple1))
    tup_delete()
    print("tuple after deletion : " + str(tuple1))
elif arg=="3":
    print("tuple before updating : " + str(tuple1))
    tup_update()
    print("tuple after updating : " + str(tuple1))
elif arg=="4":
    print("tuple retrieved : ")
    tup_retrieve()
elif arg=="5":
    tup_indexing()
elif arg=="6":
    tup_slicing()
elif arg=="7":
    print("tuple beforeconcatenation : " + str(tuple1))
    tup_concatenation()
    print("List after concatenation : " + str(tuple1))
else:
    print("Invalid Operation")
```

```
Tuple : ('Athens', 'city', 'swimming', '1986', 'gold')

Select the operation from the following:
1. Insert
2. Delete
3. Update
4. Retrieve
5. Indexing
6. Slicing
7. Concatenation

Enter numbers to select operation : 1
1
Tuple before insertion : ('Athens', 'city', 'swimming', '1986', 'gold')
Tuple before insertion : ('Athens', 'city', 'swimming', '1986', 'gold')
Enter position for insertion1
Enter the string for insertion : jj
Tuple after insertion : ('Athens', 'jj', 'city', 'swimming', '1986', 'gold')
Tuple after insertion : ('Athens', 'jj', 'city', 'swimming', '1986', 'gold')
```

8 Write a program to do set operations from data of your dataset: (i)intersection (ii)union (iii)difference (iv)symmetric difference (v)check s1 is a subset of s2(vi)check if s1 is a superset of s2(vii)find whether two sets are disjoint or not(viii)find all subsets of a set without using itertools

```
def set_intersection():
    s1,s2
    print("Intersection of s1 and s2 : "+str(s1.intersection(s2)))

def set_union():
    s1,s2
    print("Union of s1 and s2 : " + str(s1.union(s2)))

def set_difference():
    s1,s2
    print("Set Difference : " + str(s1.difference(s2)))
```

```
def set_symmetric_difference():
    s1,s2
    print("Symmetric Difference : " + str(s1.symmetric_difference(s2)))

def set_subset():
    s1,s2
    print("s1 subset of s2 : " + str(s1.issubset(s2)))

def set_superset():
    s1,s2
    print("s1 superset of s2 : "+str(s1.issuperset(s2)))

def set_disjoint():
    s1,s2
    print("s1 and s2 are disjoint : "+str(s1.isdisjoint(s2)))

def set_subsets():

    s=list(s1)
    x=len(s)
    for i in range(1<<x):
        print([s[j] for j in range(x) if (i & (1<<j))])

def switch(argument):
    switcher={
        "1":set_intersection,
        "2":set_union,
        "3":set_difference,
        "4":set_symmetric_difference,
        "5":set_subset,
        "6":set_superset,
        "7":set_disjoint,
        "8":set_subsets,
    }
    r=switcher.get(argument)()

s1={"gold","1896","city","Athens"}
s2={"silver","city"," Athens","Grown Ups"}
print("s1 : " + str(s1) + "\n")
print("s2 : " + str(s2) + "\n")

print("""Select the operation from the following:
1. Intersection
2. Union
3. Difference
4. Symmetric difference
5. Check s1 is a subset of s2
6. Check if s1 is a superset of s2
7. Find whether two sets are disjoint or not
8. Find all subsets of a set without using itertools
""")
arg=input("Enter numbers to select operation : ")
print()

switch(arg)
```

```
s1 : {'gold', 'city', 'Athens', '1896'}

s2 : {'city', 'Grown Ups', ' Athens', 'silver'}

Select the operation from the following:
1. Intersection
2. Union
3. Difference
```

```
4. Symmetric difference
5. Check s1 is a subset of s2
6. Check if s1 is a superset of s2
7. Find whether two sets are disjoint or not
8. Find all subsets of a set without using itertools

Enter numbers to select operation : 8

[]
['gold']
['city']
['gold', 'city']
['Athens']
['gold', 'Athens']
['city', 'Athens']
['gold', 'city', 'Athens']
['1896']
['gold', '1896']
['city', '1896']
['gold', 'city', '1896']
['Athens', '1896']
['gold', 'Athens', '1896']
['city', 'Athens', '1896']
['gold', 'city', 'Athens', '1896']
```

9: Write a program to do operations on a dictionary from data of your dataset.

```
def dict_insert():
    global dict1
    print("Dictionary before insertion : " + str(dict1))
    x=list(dict1.items())
    x.insert(int(input("Enter position for insertion : ")), (input("Enter Key : "),input("Enter
    dict1=dict(x)
    print("Dictionary after insertion : " + str(dict1))


def dict_delete():
    global dict1
    print("Dictionary before deletion : " + str(dict1))

    key_to_delete=(input("Enter key to be deleted : "))
    i=0

    #traverse through the dictionary to find the desired key
    #Once found, it is deleted
    for key in dict1.keys():
        if key_to_delete in dict1:
            del dict1[key_to_delete]
            i=1
            break
    if i==0:
        print("Key not found!")
    print("Dictionary after deletion : " + str(dict1))

#funciton to update dictionary
def dict_update():
    global dict1
    print("dictionary before updating : " + str(dict1))

    key_to_replace=(input("Enter the key to be replaced : "))#input of key for updating the dict
    value_to_replace=input("Enter the value : ")#input of value for updating the dictionary

    #update dictionary with the help of 'update' function
    dict1.update({key_to_replace:value_to_replace})

    print("dictionary after updating : " + str(dict1))

def dict_retrieve():
    global dict1
```

```
    print("Dictionary retrieved : " + str(dict1))

#funciton to get index of key
def dict_indexing():
    global dict1
    k=input("Enter the key to get its value : ")#input of key

    #using list + keys() + index() we get the index of key
    res=list(dict1.keys()).index(k)
    print("Index of key : " + str(res))

#function for dictionary slicing
def dict_slicing():
    global dict1
    start=int(input("Enter the start position : "))
    end=int(input("Enter the end position : "))
    res=dict()

    #loop to traverse through dictionary keys
    for i in dict1:
        res[i]=dict1[i][start:end]
    print(res)

#function to concatenate key-value pair to the existing dictionary
def dict_concatenation():
    global dict1
    print("dictionary before concatenation : " + str(dict1))

    #input of key to be concatenated
    key_to_concatenate=(input("Enter the key to be concatenated : "))

    #input of value to be concatenated
    value_to_concatenate=input("Enter the value to be concatenated : ")

    #concatenation using update()
    dict1.update({key_to_concatenate:value_to_concatenate})
    print("dictionary after concatenation : " + str(dict1))

#switcher to select operation
def switch(argument):
    switcher={
        "1":dict_insert,
        "2":dict_delete,
        "3":dict_update,
        "4":dict_retrieve,
        "5":dict_indexing,
        "6":dict_slicing,
        "7":dict_concatenation,
    }
    r=switcher.get(argument)()

#initial dictionary
dict1={"City":"Athnes","Activity":"Swimming","Medal":"silver"}
print("Dictionary : " + str(dict1) + "\n")


print("""Select the operation from the following:
1. Insert
2. Delete
3. Update
4. Retrieve
5. Indexing
```

```
6. Slicing
7. Concatenation
""")
arg=input("Enter numbers to select operation : ")#input to select operation
print()

switch(arg)
```

```
Dictionary : {'City': 'Athnes', 'Activity': 'Swimming', 'Medal': 'silver'}

Select the operation from the following:
1. Insert
2. Delete
3. Update
4. Retrieve
5. Indexing
6. Slicing
7. Concatenation

Enter numbers to select operation : 1

Dictionary before insertion : {'City': 'Athnes', 'Activity': 'Swimming', 'Medal': 'silver'}
Enter position for insertion : 2
Enter Key : year
Enter value : 2012
Dictionary after insertion : {'City': 'Athnes', 'Activity': 'Swimming', 'year': '2012', 'Medal': 'silver'}
```

CO-2 Assignment:

10 .Read and analyze your data set for the following: a.Number of rows b.Number of attributes c.Number of missing values for each attribute

```
from google.colab import files


uploaded = files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving summer.csv to summer.csv
```

```
import pandas as pd
df = pd.read_csv('summer.csv')
print(df)
```

```
       Year    City      Sport          Discipline  \
0      1896  Athens   Aquatics            Swimming
1      1896  Athens   Aquatics            Swimming
2      1896  Athens   Aquatics            Swimming
3      1896  Athens   Aquatics            Swimming
4      1896  Athens   Aquatics            Swimming
...     ...     ...        ...                 ...
31160  2012  London  Wrestling  Wrestling Freestyle
31161  2012  London  Wrestling  Wrestling Freestyle
31162  2012  London  Wrestling  Wrestling Freestyle
31163  2012  London  Wrestling  Wrestling Freestyle
31164  2012  London  Wrestling  Wrestling Freestyle

                        Athlete Country Gender                       Event  \
0                 HAJOS, Alfred     HUN    Men             100M Freestyle
1               HERSCHMANN, Otto     AUT    Men             100M Freestyle
2              DRIVAS, Dimitrios     GRE    Men  100M Freestyle For Sailors
3             MALOKINIS, Ioannis     GRE    Men  100M Freestyle For Sailors
4             CHASAPIS, Spiridon     GRE    Men  100M Freestyle For Sailors
...                          ...     ...    ...                        ...
31160         JANIKOWSKI, Damian     POL    Men                    Wg 84 KG
31161  REZAEI, Ghasem Gholamreza     IRI    Men                    Wg 96 KG
31162             TOTROV, Rustam     RUS    Men                    Wg 96 KG
31163          ALEKSANYAN, Artur     ARM    Men                    Wg 96 KG
31164             LIDBERG, Jimmy     SWE    Men                    Wg 96 KG

        Medal
0        Gold
1      Silver
2      Bronze
3        Gold
4      Silver
...       ...
31160  Bronze
31161    Gold
31162  Silver
```

```
31163  Bronze
31164  Bronze

[31165 rows x 9 columns]
```

```
print(df.shape[0])#rows
df.shape[1]#column
```

```
31165
9
```

```
df.ndim
```

```
2
```

```
print("Number of rows :",len(df))
print("Number of attributes :",len(df.dtypes))
print("Number of missing values for each attribute :\n"+str(df.isnull().sum()))
print("Total number of missing values :"+str(df.isnull().sum().sum()))
```

```
Number of rows : 31165
Number of attributes : 9
Number of missing values for each attribute :
Year          0
City          0
Sport         0
Discipline    0
Athlete       0
Country       4
Gender        0
Event         0
Medal         0
dtype: int64
Total number of missing values :4
```

Double-click (or enter) to edit

11.Write a program to parse HTML documents w.r.to your dataset using Beautiful Soup.

```
import requests
import lxml
from bs4 import BeautifulSoup
#url
url1="https://olympics.com/en/olympic-games/athens-2004"
#response from url using requests()
r=requests.get(url1)
#parsing
soup=BeautifulSoup(r.content,'lxml')
print(f'{soup.h1.name} : {soup.h1.text}')#h1
print(f'{soup.h2.name} : {soup.h2.text}')#h2
print(f'{soup.h3.name} : {soup.h3.text}')#h3
print(f'{soup.li.name} : {soup.li.text}')#li
```

```
h1 : Athens 2004
h2 : Olympic Games Athens 2004
h3 : Biggest Games
li :
IOC
```

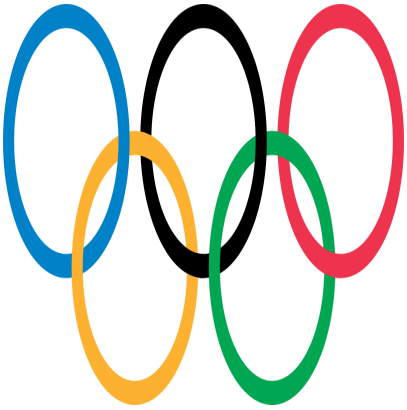
CO-3 Assignment:

12. Display graphics and multimedia video related to your data set in Jupyter notebook.

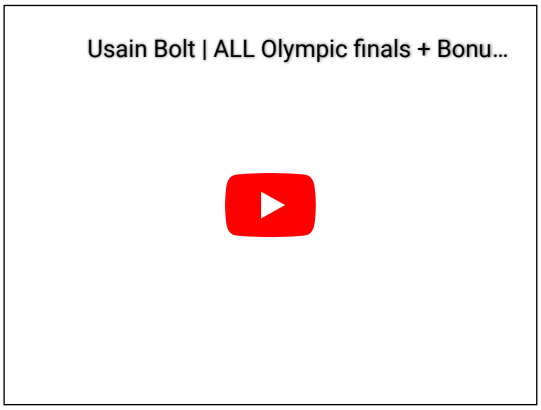
```
from google.colab import files
f=files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving summer.csv to summer.csv
```

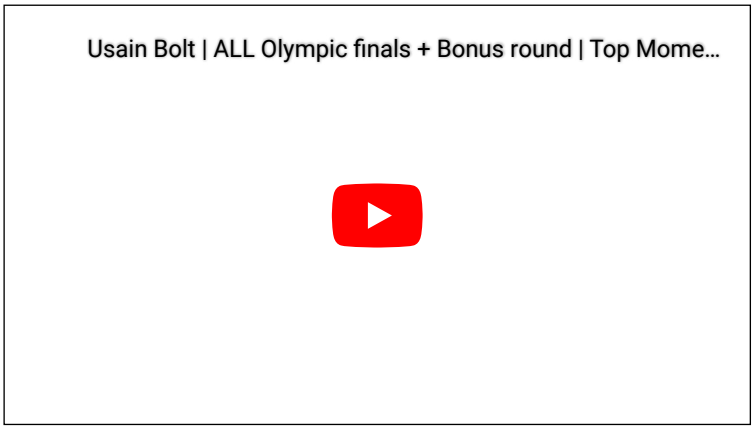
```
from IPython.display import Image
Image(url='https://upload.wikimedia.org/wikipedia/commons/thumb/5/5c/Olympic_rings_without_rims.
```



```
from IPython.lib.display import YouTubeVideo
YouTubeVideo('FuiJHJz4f5Q')
```




```
from IPython.display import HTML
# Youtube
HTML('<iframe width="560" height="315" src="https://www.youtube.com/embed/FuiJHJz4f5Q?rel=0&amp;
```



Double-click (or enter) to edit

Q-13Read your data set and do the following: (a) Validating Your Data, Figuring out what's in your data, (b) Removing duplicates, Creating a data map and data plan, Manipulating (c) Categorical Variables, Creating categorical variables, Renaming levels, (d) Combining levels, Dealing with Dates in Your Data, Formatting date and time values, Using the right time transformation, (e) Dealing with Missing Data, Finding the missing

data, (f) Encoding missingness, Imputing missing data, Slicing and Dicing: i. Filtering and Selecting Data, Slicing rows, Slicing columns, Dicing, ii. Concatenating and Transforming, Adding new cases and variables, Removing data iii.Sorting and shuffling, Aggregating Data at Any Level.

```
from google.colab import files
```

```
uploaded = files.upload()
```

```
Choose Files No file chosen  Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving summer.csv to summer.csv
```

(a) Validating Your Data, Figuring out what's in your data:

```
import pandas as pd
```

```
df=pd.read_csv('summer.csv')
df.head()
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Me
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	G
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Si
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For	Bro

```
df.index
```

```
RangeIndex(start=0, stop=31165, step=1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 31165 entries, 0 to 31164
Data columns (total 9 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Year        31165 non-null  int64 
 1   City        31165 non-null  object
 2   Sport       31165 non-null  object
 3   Discipline  31165 non-null  object
 4   Athlete     31165 non-null  object
 5   Country     31161 non-null  object
 6   Gender      31165 non-null  object
 7   Event       31165 non-null  object
 8   Medal       31165 non-null  object
dtypes: int64(1), object(8)
memory usage: 2.1+ MB
```

```
df.describe()
```

	Year
count	31165.000000
mean	1970.482785
std	33.158416
min	1896.000000
25%	1948.000000
50%	1980.000000
75%	2000.000000
max	2012.000000

```
df.tail()
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	M
31160	2012	London	Wrestling	Wrestling Freestyle	JANIKOWSKI, Damian	POL	Men	Wg 84 KG	B
31161	2012	London	Wrestling	Wrestling Freestyle	REZAEI, Ghasem Gholamreza	IRI	Men	Wg 96 KG	

```
df.dtypes
```

```
Year           int64
City          object
Sport         object
Discipline    object
Athlete       object
Country       object
Gender        object
Event         object
Medal         object
dtype: object
```

```
df.shape
```

```
(31165, 9)
```

```
df.size
```

```
280485
```

(b) Removing duplicates, Creating a data map and data plan, Manipulating:

```
print("Data frame before removing duplicates")
df
```

```
Data frame before removing duplicates
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors
...	...	...	...	...	...	...	...	...

```
df.duplicated()
```

```
0        False
1        False
2        False
3        False
4        False
         ...
31160    False
31161    False
31162    False
31163    False
31164    False
Length: 31165, dtype: bool
```

```
df.drop_duplicates()
print("Data frame after removing duplicates : ")
df
```

```
Data frame after removing duplicates :
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis	GRE	Men	100M Freestyle For Sailors
4	1896	Athens	Aquatics	Swimming	CHASAPIS, Spiridon	GRE	Men	100M Freestyle For Sailors
...	...	...	...	...	...	...	...	...

(c)Categorical Variables, Creating categorical variables, Renaming levels:

```
df_cat=pd.Series(['Event','city','discipline'],dtype= 'category')
print(df_cat)
```

```
0         Event
1          city
2    discipline
dtype: category
Categories (3, object): ['Event', 'city', 'discipline']
```

```
activities = pd.Series(pd.Categorical(['Event','discipline','city','sport'],categories=df_cat))
print(activities)
```

```
0         Event
1    discipline
2          city
3           NaN
dtype: category
Categories (3, object): ['Event', 'city', 'discipline']
```

```
#rename
df_cat=pd.Series(['Event','city','discipline'],dtype= 'category')
olympics=pd.Series(pd.Categorical(['Event','discipline','city','sport'],categories=df_cat,ordere
activities.cat.categories=['new_Event','new_city','new_discipline',]
olympics.cat.categories=activities.cat.categories
print(olympics)
```

```
0         new_Event
1    new_discipline
2          new_city
3               NaN
dtype: category
Categories (3, object): ['new_Event', 'new_city', 'new_discipline']
```

(d) Combining levels, Dealing with Dates in Your Data, Formatting date and time values, Using the right time transformation:

```
!pip install pandas-validation
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting pandas-validation
  Downloading pandas_validation-0.5.0-py2.py3-none-any.whl (6.9 kB)
Requirement already satisfied: pandas>=0.22 in /usr/local/lib/python3.7/dist-packages (from pandas-validation) (1.3.5)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.22->pandas-validation) (2022.
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.22->pandas-validation) (1.21
```

```
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.22->pandas-validati
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas>=0.22->panda
Installing collected packages: pandas-validation
Successfully installed pandas-validation-0.5.0
```

```
import datetime as date
time = date.datetime.now()
print(str(time))
print(time.strftime('%a, %d %B %Y'))
print(time.strftime('%H:%M:%S'))
```

```
2022-11-19 15:12:26.167808
Sat, 19 November 2022
15:12:26
```

(e) Dealing with Missing Data, Finding the missing data:

```
print("'True' values the below matrix shows missing data : ")
pd.isna(df)
```

```
'True' values the below matrix shows missing data :
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Medal
0	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...
31160	False	False	False	False	False	False	False	False	False
31161	False	False	False	False	False	False	False	False	False
31162	False	False	False	False	False	False	False	False	False
31163	False	False	False	False	False	False	False	False	False
31164	False	False	False	False	False	False	False	False	False

31165 rows × 9 columns

```
print("\nTotal number of missing values for each attribute' : ")
pd.isna(df).sum()
```

```
Total number of missing values for each attribute' :
Year          0
City          0
Sport         0
Discipline    0
Athlete       0
Country       4
Gender        0
Event         0
Medal         0
dtype: int64
```

```
print("\nTotal number of missing values in entire data framme : "+str(pd.isna(df).sum().sum()))
```

```
Total number of missing values in entire data framme : 4
```

(f) Encoding missingness, Imputing missing data, Slicing and Dicing:

i) Filtering and Selecting Data, Slicing rows, Slicing columns, Dicing:

```
#Selecting Data
# Selecting columns
```

```
show_detail=df[["Gender","Athlete"]]
show_detail.head()
```

	Gender	Athlete
0	Men	HAJOS, Alfred
1	Men	HERSCHMANN, Otto
2	Men	DRIVAS, Dimitrios
3	Men	MALOKINIS, Ioannis
4	Men	CHASAPIS, Spiridon

```
#Selecting rows:
show_detail=show_detail[0:10]
show_detail
```

	Gender	Athlete
0	Men	HAJOS, Alfred
1	Men	HERSCHMANN, Otto
2	Men	DRIVAS, Dimitrios
3	Men	MALOKINIS, Ioannis
4	Men	CHASAPIS, Spiridon
5	Men	CHOROPHAS, Efstathios
6	Men	HAJOS, Alfred
7	Men	ANDREOU, Joannis
8	Men	CHOROPHAS, Efstathios
9	Men	NEUMANN, Paul

```
show_detail=df[(df["Medal"]=="Gold")&(df["Country"]=="POL")]
show_detail.head()
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	
5170	1928	Amsterdam	Athletics	Athletics	KONOPACKA, Halina	POL	Women	Discus
5803	1932	Los Angeles	Athletics	Athletics	KUSOCINSKI, Janusz	POL	Men	1
5809	1932	Los Angeles	Athletics	Athletics	WALASIEWICZ, Stanislawa	POL	Women	

Slicing dataFrame using loc and iloc:

Syntax: loc[row labels, columns labels]

```
#Slice rows by label.
df.loc[1:3, :]
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Me
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Si
					DRIVAS,			100M Freestyle	

```
#Slice columns by label.
df.loc[:, "Year":"Medal"]
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors
					MALOKINIS			100M

```
#To slice rows by index position.
df.iloc[0:2,:]
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Med
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	G

```
#To slice columns by index position.
df.iloc[:,1:3]
```

	City	Sport
0	Athens	Aquatics
1	Athens	Aquatics
2	Athens	Aquatics
3	Athens	Aquatics
4	Athens	Aquatics
...	...	...
31160	London	Wrestling
31161	London	Wrestling
31162	London	Wrestling
31163	London	Wrestling
31164	London	Wrestling

31165 rows × 2 columns

Dicing dataFrame using loc and iloc:

```
df.loc[0:5, "Year":"Medal"]
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	Event	Me
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred	HUN	Men	100M Freestyle	G
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto	AUT	Men	100M Freestyle	Si
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios	GRE	Men	100M Freestyle For Sailors	Bro

```
df.iloc[0:6,1:3]
```

	City	Sport
1	Athens	Aquatics

ii) Sorting and shuffling, Aggregating Data at Any Level:

```
sort_df=df.sort_values(by=['City','Country'],ascending=[True,False])
sort_df
```

	Year	City	Sport	Discipline	Athlete	Country	Gender	
5415	1928	Amsterdam	Gymnastics	Artistic G.	STUKELJ, Leon	YUG	Men	In All
5420	1928	Amsterdam	Gymnastics	Artistic G.	PRIMOZIC, Josip	YUG	Men	
5425	1928	Amsterdam	Gymnastics	Artistic G.	STUKELJ, Leon	YUG	Men	
5427	1928	Amsterdam	Gymnastics	Artistic G.	ANTOSIEWIC, Eduard	YUG	Men	Com
5428	1928	Amsterdam	Gymnastics	Artistic G.	CIOTTI, Dragutin	YUG	Men	Com
...	...	...	...	...	...	...	...	
11398	1964	Tokyo	Judo	Judo	BORONOSKI, Theodore	AUS	Men	C
11495	1964	Tokyo	Sailing	Sailing	NORTHAM, William	AUS	Men	

Aggregating Data

```
df.groupby('Sport').aggregate(['min','sum','mean','max'])
```

	Year			
	min	sum	mean	max
Sport				
Aquatics	1896	8242072	1976.516067	2012

(ii) Concatenating and Transforming, Adding new cases and variables, Removing data:

```
df1=df.iloc[0:3,0:3]
df1
```

	Year	City	Sport
0	1896	Athens	Aquatics
1	1896	Athens	Aquatics
2	1896	Athens	Aquatics

```
df2=df.iloc[3:6,0:3]
df2
```

	Year	City	Sport
3	1896	Athens	Aquatics
4	1896	Athens	Aquatics
5	1896	Athens	Aquatics
Gymnastics	1896	4515332	1957.231036	2012

```
df3=df.iloc[60:80,0:3]
df3
```

	Year	City	Sport
60	1896	Athens	Cycling
61	1896	Athens	Cycling
62	1896	Athens	Cycling
63	1896	Athens	Cycling
64	1896	Athens	Fencing
65	1896	Athens	Fencing
66	1896	Athens	Fencing
67	1896	Athens	Fencing
68	1896	Athens	Fencing
69	1896	Athens	Fencing
70	1896	Athens	Fencing
71	1896	Athens	Fencing
72	1896	Athens	Gymnastics
73	1896	Athens	Gymnastics
74	1896	Athens	Gymnastics
75	1896	Athens	Gymnastics
76	1896	Athens	Gymnastics
77	1896	Athens	Gymnastics
78	1896	Athens	Gymnastics
79	1896	Athens	Gymnastics

```
concat_df=pd.concat([df1,df2,df3])
concat_df
```

	Year	City	Sport
0	1896	Athens	Aquatics
1	1896	Athens	Aquatics
2	1896	Athens	Aquatics
3	1896	Athens	Aquatics
4	1896	Athens	Aquatics
5	1896	Athens	Aquatics
60	1896	Athens	Cycling
61	1896	Athens	Cycling
62	1896	Athens	Cycling
63	1896	Athens	Cycling
64	1896	Athens	Fencing
65	1896	Athens	Fencing
66	1896	Athens	Fencing
67	1896	Athens	Fencing
68	1896	Athens	Fencing
69	1896	Athens	Fencing
70	1896	Athens	Fencing
71	1896	Athens	Fencing
72	1896	Athens	Gymnastics
73	1896	Athens	Gymnastics
74	1896	Athens	Gymnastics
75	1896	Athens	Gymnastics
76	1896	Athens	Gymnastics
77	1896	Athens	Gymnastics

```
df4=df.iloc[0:4,0:5]
df4
```

	Year	City	Sport	Discipline	Athlete
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis

```
pd.concat([df1,df2,df4])
```

	Year	City	Sport	Discipline	Athlete
0	1896	Athens	Aquatics	NaN	NaN
1	1896	Athens	Aquatics	NaN	NaN
2	1896	Athens	Aquatics	NaN	NaN
3	1896	Athens	Aquatics	NaN	NaN
4	1896	Athens	Aquatics	NaN	NaN
5	1896	Athens	Aquatics	NaN	NaN
0	1896	Athens	Aquatics	Swimming	HAJOS, Alfred
1	1896	Athens	Aquatics	Swimming	HERSCHMANN, Otto
2	1896	Athens	Aquatics	Swimming	DRIVAS, Dimitrios
3	1896	Athens	Aquatics	Swimming	MALOKINIS, Ioannis

CO-4 ASSIGNMENT:

14 Draw each type of graph for your dataset. Bar graph Line graph Scatter plot Pie chart Histogram Box plot

```
from google.colab import files


uploaded = files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving summer.csv to summer.csv
```

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

df=pd.read_csv('summer.csv')
print(df)
```

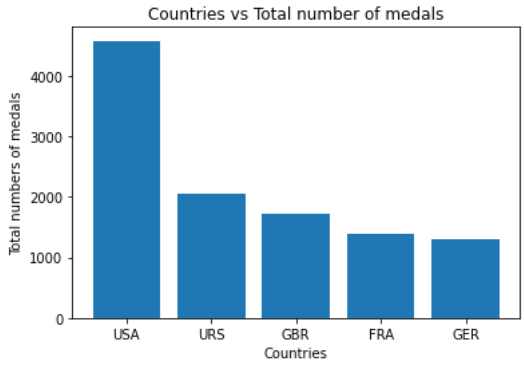
```
          Year    City      Sport          Discipline  \
    0      1896  Athens   Aquatics            Swimming
    1      1896  Athens   Aquatics            Swimming
    2      1896  Athens   Aquatics            Swimming
    3      1896  Athens   Aquatics            Swimming
    4      1896  Athens   Aquatics            Swimming
    ...     ...     ...        ...                 ...
    31160  2012  London  Wrestling  Wrestling Freestyle
    31161  2012  London  Wrestling  Wrestling Freestyle
    31162  2012  London  Wrestling  Wrestling Freestyle
    31163  2012  London  Wrestling  Wrestling Freestyle
    31164  2012  London  Wrestling  Wrestling Freestyle

                           Athlete Country Gender                       Event  \
    0                  HAJOS, Alfred     HUN    Men              100M Freestyle
    1                HERSCHMANN, Otto     AUT    Men              100M Freestyle
    2               DRIVAS, Dimitrios     GRE    Men  100M Freestyle For Sailors
    3              MALOKINIS, Ioannis     GRE    Men  100M Freestyle For Sailors
    4              CHASAPIS, Spiridon     GRE    Men  100M Freestyle For Sailors
    ...                           ...     ...    ...                         ...
    31160         JANIKOWSKI, Damian     POL    Men                    Wg 84 KG
    31161  REZAEI, Ghasem Gholamreza     IRI    Men                    Wg 96 KG
    31162             TOTROV, Rustam     RUS    Men                    Wg 96 KG
    31163          ALEKSANYAN, Artur     ARM    Men                    Wg 96 KG
    31164             LIDBERG, Jimmy     SWE    Men                    Wg 96 KG

           Medal
    0       Gold
    1     Silver
    2     Bronze
    3       Gold
    4     Silver
    ...      ...
    31160 Bronze
    31161   Gold
    31162 Silver
    31163 Bronze
    31164 Bronze

    [31165 rows x 9 columns]
```

```
a = df.Country.value_counts().sort_values(ascending=False).head(5)
a
```

```
    USA    4585
    URS    2049
    GBR    1720
    FRA    1396
    GER    1305
    Name: Country, dtype: int64
```

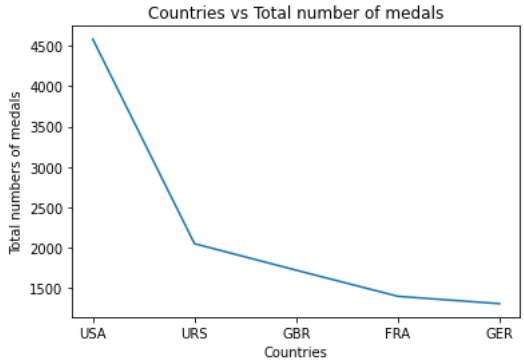
Bar Graph

```
plt.bar(a.index, a)
plt.title("Countries vs Total number of medals")
plt.xlabel("Countries")
plt.ylabel("Total numbers of medals")
plt.show()
```



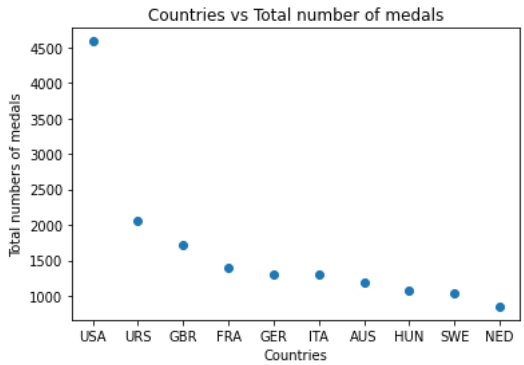
Line Graph

```
plt.plot(a)
plt.title("Countries vs Total number of medals")
plt.xlabel("Countries")
plt.ylabel("Total numbers of medals")
plt.show()
```



Scatter Plot

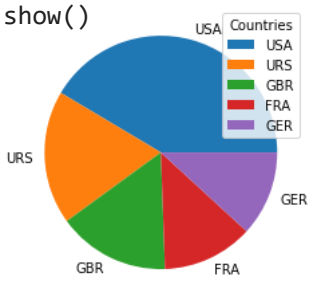
```
b = df.Country.value_counts().sort_values(ascending=False).head(10)
plt.scatter(b.index, b)
plt.title("Countries vs Total number of medals")
plt.xlabel("Countries")
plt.ylabel("Total numbers of medals")
plt.show()
```



Pie Chart

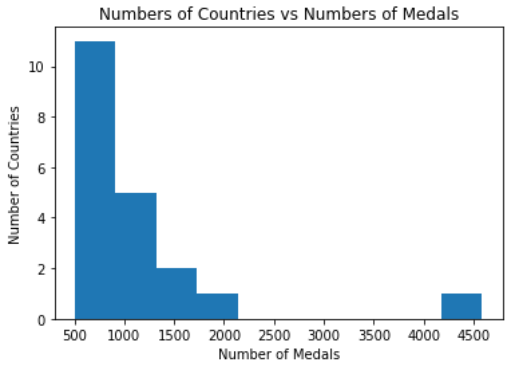
```
from turtle import title


mylabels = ["USA", "URS", "GBR", "FRA", "GER"]
plt.pie(a, labels=mylabels)
```

```
plt.legend(title = "Countries")
plt.show()
```



Histogram

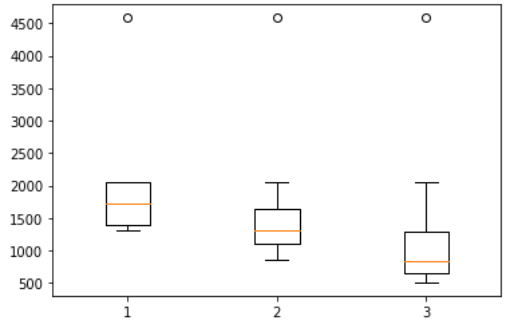
```
c = df.Country.value_counts().sort_values(ascending=False).head(20)
plt.hist(c)
plt.ylabel("Number of Countries")
plt.xlabel("Number of Medals")
plt.title("Numbers of Countries vs Numbers of Medals")
plt.show()
```



Box Plot

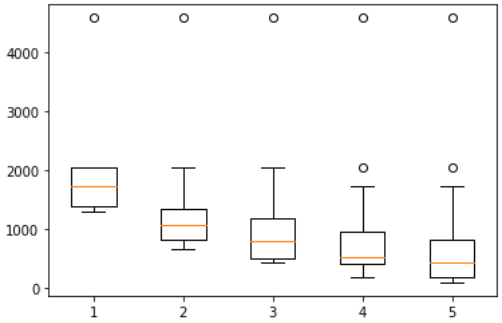
```
data = [a,b,c]
plt.boxplot(data)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleD
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```




```
data1 = df["Country"].value_counts().head(5)
data2 = df["Country"].value_counts().head(15)
data3 = df["Country"].value_counts().head(25)
data4 = df["Country"].value_counts().head(35)
data5 = df["Country"].value_counts().head(45)

boxplot_data = [data1, data2, data3, data4, data5]
plt.boxplot(boxplot_data)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleD
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```



CO-5 ASSIGNMENT

Practical 15

write a program to find the relationship between two attributes using correlation.

```
# Import those libraries
import pandas as pd
from scipy.stats import pearsonr

# Import your data into Python
df = pd.read_csv("summer.csv")
# Convert dataframe into series
list2 = [22,23,24,25,26]
list1 = [1,2,3,4,5]


# Apply the pearsonr()
corr, _ = pearsonr(list1, list2)
print('Pearsons correlation: %.3f' % corr)
```

```
Pearsons correlation: 1.000
```

Practical-16

Write a program to test data using chi-square.

```
from scipy.stats import chi2_contingency

# defining the table
data = [[207, 282, 241], [234, 242, 232]]
stat, p, dof, expected = chi2_contingency(data)

# interpret p-value
alpha = 0.05
print("p value is " + str(p))
if p <= alpha:
    print('Dependent (reject H0)')
else:
    print('Independent (H0 holds true)')
```

```
p value is 0.1031971404730939
Independent (H0 holds true)
```

Practical-17

Write a program to transform data using Z-score.

```
# Calculate the z-score from with scipy
import scipy.stats as stats
values = df["Year"]

zscores = stats.zscore(values)
print(zscores)
```

```
0       -2.246307
1       -2.246307
2       -2.246307
3       -2.246307
4       -2.246307
           ...
31160    1.252107
31161    1.252107
31162    1.252107
31163    1.252107
31164    1.252107
Name: Year, Length: 31165, dtype: float64
```

Practical 18

Write a program to calculate the TF-IDF score of the given documents using NLTK.

```
corpus1 = 'Athletics is a group of sporting events that involves competitive running, jumping, t
corpus1
```

```
'Athletics is a group of sporting events that involves competitive running, jumpi
ng, throwing, and walking. The most common types of athletics competitions are tr
ack and field, road running, cross country running, and racewalking'
```

```
corpus2 = 'this will be great i really want an e-reader but love my physical books so much that
'covers but if i can enjoy some of the benefits of an e-reader on the go then come home and pick
corpus2
```

```
'this will be great i really want an e-reader but love my physical books so much
that i do not see the point of buy one because they have beautiful'
```

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer # This is a very good package for Ma

list_of_words_1 = corpus1.split(' ')
list_of_words_1
```

```
['Athletics',
 'is',
 'a',
 'group',
 'of',
 'sporting',
 'events',
 'that',
 'involves',
 'competitive',
 'running,',
 'jumping,',
 'throwing,',
 'and',
 'walking.',
 'The',
 'most',
 'common',
 'types',
 'of',
 'athletics',
 'competitions',
 'are',
 'track',
 'and',
 'field,',
 'road',
 'running,',
 'cross',
 'country',
 'running,',
 'and',
 'racewalking']
```

```
list_of_words_2 = corpus2.split(' ')
list_of_words_2
```

```
['this',
 'will',
 'be',
 'great',
 'i',
 'really',
 'want',
 'an',
 'e-reader',
 'but',
 'love',
 'my',
 'physical',
 'books',
 'so',
 'much',
 'that',
 'i',
 'do',
 'not',
 'see',
 'the',
 'point',
 'of',
 'buy',
 'one',
 'because',
 'they',
 'have',
 'beautiful']
```

```
unique_words = set(list_of_words_1).union(set(list_of_words_2 ))
unique_words
```

```
{'Athletics',
 'The',
 'a',
 'an',
 'and',
 'are',
 'athletics',
 'be',
 'beautiful',
 'because',
 'books',
 'but',
 'buy',
 'common',
 'competitions',
 'competitive',
 'country',
 'cross',
 'do',
 'e-reader',
 'events',
 'field,',
 'great',
 'group',
 'have',
 'i',
 'involves',
 'is',
 'jumping,',
 'love',
 'most',
 'much',
 'my',
 'not',
 'of',
 'one',
 'physical',
 'point',
 'racewalking',
 'really',
 'road',
 'running,',
 'see',
 'so',
 'sporting',
 'that',
 'the',
 'they',
 'this',
 'throwing,',
```

```
     'track',
     'types',
     'walking.',
     'want',
     'will'}
```

```
# Create a dictionaryto count the occurence of the unique words in both corpus

num_words_1 = dict.fromkeys(unique_words,0)
for word in list_of_words_1:
  num_words_1[word] += 1
num_words_2 = dict.fromkeys(unique_words,0)
for word in list_of_words_2:
  num_words_2[word] += 1

# Function to calculate TF

def computeTF(word_dict, list_of_words):
  tf_dict = {}
  words_count = len(list_of_words)
  for word, count in word_dict.items():
    tf_dict[word] = count / float(words_count)
  return tf_dict

tf1 = computeTF(num_words_1, list_of_words_1)
tf1
```

```
    {'i': 0.0,
     'throwing,': 0.030303030303030304,
     'and': 0.09090909090909091,
     'a': 0.030303030303030304,
     'competitive': 0.030303030303030304,
     'my': 0.0,
     'love': 0.0,
     'see': 0.0,
     'much': 0.0,
     'they': 0.0,
     'that': 0.030303030303030304,
     'most': 0.030303030303030304,
     'track': 0.030303030303030304,
     'point': 0.0,
     'group': 0.030303030303030304,
     'cross': 0.030303030303030304,
     'Athletics': 0.030303030303030304,
     'an': 0.0,
     'sporting': 0.030303030303030304,
     'of': 0.06060606060606061,
     'one': 0.0,
     'walking.': 0.030303030303030304,
     'is': 0.030303030303030304,
     'not': 0.0,
     'have': 0.0,
     'running,': 0.09090909090909091,
     'this': 0.0,
     'buy': 0.0,
     'the': 0.0,
     'great': 0.0,
     'will': 0.0,
     'involves': 0.030303030303030304,
     'but': 0.0,
     'physical': 0.0,
     'so': 0.0,
     'competitions': 0.030303030303030304,
     'events': 0.030303030303030304,
     'be': 0.0,
     'racewalking': 0.030303030303030304,
     'are': 0.030303030303030304,
     'country': 0.030303030303030304,
     'do': 0.0,
     'common': 0.030303030303030304,
     'road': 0.030303030303030304,
     'types': 0.030303030303030304,
     'athletics': 0.030303030303030304,
     'want': 0.0,
     'The': 0.030303030303030304,
     'books': 0.0,
     'jumping,': 0.030303030303030304,
     'beautiful': 0.0,
     'because': 0.0,
```

```
 'e-reader': 0.0,
 'really': 0.0,
 'field,': 0.030303030303030304}
```

```
tf2 = computeTF(num_words_2, list_of_words_2)
tf2
```

```
{'i': 0.06666666666666667,
 'throwing,': 0.0,
 'and': 0.0,
 'a': 0.0,
 'competitive': 0.0,
 'my': 0.03333333333333333,
 'love': 0.03333333333333333,
 'see': 0.03333333333333333,
 'much': 0.03333333333333333,
 'they': 0.03333333333333333,
 'that': 0.03333333333333333,
 'most': 0.0,
 'track': 0.0,
 'point': 0.03333333333333333,
 'group': 0.0,
 'cross': 0.0,
 'Athletics': 0.0,
 'an': 0.03333333333333333,
 'sporting': 0.0,
 'of': 0.03333333333333333,
 'one': 0.03333333333333333,
 'walking.': 0.0,
 'is': 0.0,
 'not': 0.03333333333333333,
 'have': 0.03333333333333333,
 'running,': 0.0,
 'this': 0.03333333333333333,
 'buy': 0.03333333333333333,
 'the': 0.03333333333333333,
 'great': 0.03333333333333333,
 'will': 0.03333333333333333,
 'involves': 0.0,
 'but': 0.03333333333333333,
 'physical': 0.03333333333333333,
 'so': 0.03333333333333333,
 'competitions': 0.0,
 'events': 0.0,
 'be': 0.03333333333333333,
 'racewalking': 0.0,
 'are': 0.0,
 'country': 0.0,
 'do': 0.03333333333333333,
 'common': 0.0,
 'road': 0.0,
 'types': 0.0,
 'athletics': 0.0,
 'want': 0.03333333333333333,
 'The': 0.0,
 'books': 0.03333333333333333,
 'jumping,': 0.0,
 'beautiful': 0.03333333333333333,
 'because': 0.03333333333333333,
 'e-reader': 0.03333333333333333,
 'really': 0.03333333333333333,
 'field,': 0.0}
```

```
# Function to calculate IDF

def computeIDF (documents):
  import math
  N = len(documents)

  idf_dict = dict.fromkeys(documents[0].keys(), 0)
  for document in documents:
    for word, val in document.items():
      if val > 0:
        idf_dict[word] += 1

  for word, val in idf_dict.items():
    idf_dict[word] = math.log(N / float(val))
  return idf_dict
```

```
idfs = computeIDF([num_words_1, num_words_2])
idfs
```

```
{'i': 0.6931471805599453,
 'throwing,': 0.6931471805599453,
 'and': 0.6931471805599453,
 'a': 0.6931471805599453,
 'competitive': 0.6931471805599453,
 'my': 0.6931471805599453,
 'love': 0.6931471805599453,
 'see': 0.6931471805599453,
 'much': 0.6931471805599453,
 'they': 0.6931471805599453,
 'that': 0.0,
 'most': 0.6931471805599453,
 'track': 0.6931471805599453,
 'point': 0.6931471805599453,
 'group': 0.6931471805599453,
 'cross': 0.6931471805599453,
 'Athletics': 0.6931471805599453,
 'an': 0.6931471805599453,
 'sporting': 0.6931471805599453,
 'of': 0.0,
 'one': 0.6931471805599453,
 'walking.': 0.6931471805599453,
 'is': 0.6931471805599453,
 'not': 0.6931471805599453,
 'have': 0.6931471805599453,
 'running,': 0.6931471805599453,
 'this': 0.6931471805599453,
 'buy': 0.6931471805599453,
 'the': 0.6931471805599453,
 'great': 0.6931471805599453,
 'will': 0.6931471805599453,
 'involves': 0.6931471805599453,
 'but': 0.6931471805599453,
 'physical': 0.6931471805599453,
 'so': 0.6931471805599453,
 'competitions': 0.6931471805599453,
 'events': 0.6931471805599453,
 'be': 0.6931471805599453,
 'racewalking': 0.6931471805599453,
 'are': 0.6931471805599453,
 'country': 0.6931471805599453,
 'do': 0.6931471805599453,
 'common': 0.6931471805599453,
 'road': 0.6931471805599453,
 'types': 0.6931471805599453,
 'athletics': 0.6931471805599453,
 'want': 0.6931471805599453,
 'The': 0.6931471805599453,
 'books': 0.6931471805599453,
 'jumping,': 0.6931471805599453,
 'beautiful': 0.6931471805599453,
 'because': 0.6931471805599453,
 'e-reader': 0.6931471805599453,
 'really': 0.6931471805599453,
 'field,': 0.6931471805599453}
```

```
# Function to calculate TF-IDF

def computeTFIDF(tf, idfs):
  tfidf = {}
  for word, val in tf.items():
    tfidf[word] = val * idfs[word]
  return tfidf

tfidf1 = computeTFIDF(tf1, idfs)
tfidf2 = computeTFIDF(tf2, idfs)
df = pd.DataFrame([tfidf1, tfidf2])

df
```

	i	throwing,	and	a	competitive	my	love	see
0	0.00000	0.021004	0.063013	0.021004	0.021004	0.000000	0.000000	0.000000
1	0.04621	0.000000	0.000000	0.000000	0.000000	0.023105	0.023105	0.023105

2 rows × 55 columns