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

Subject: Python for Data Science(3150713)

Practical Assignment File

Computer Engineering Department

By: Homit Dalia, Ayush Desai

Enrollment: 200090107009, 200090107018

Dataset: Melbourne Housing Snapshot

Dataset link

CO-1 Assignment:

Practical 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))]

```
import random

operatorsList=["+","-","*","/","%"]
endNumber = 13581
startNumber = endNumber/2
length = len(operatorsList)

while(True):
  randomOperator = random.randint(0,10)
  print("Random Value = ",randomOperator)
  if(randomOperator>length-1):
    print("Index Out of Bound")
    break
  else:
    result = str(startNumber) + " " + str(operatorsList[randomO
    print(result)
    result =eval(result)
    print("Result = ",result)
```

```
Random Value =  5
Index Out of Bound
```

Practical 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 number of lines"))
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 number of lines5
A
AB
ABC
ABCDEF
ABCDEFGHIJKL
```

Practical 3:

Use list comprehension to create the following output for your dataset items. E.g.my dataset is 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
myList=["Albert Park", "3 Herbert Switch", "Altona", "15 Kookab
indices = [i for i, x in enumerate(myList) if re.search("[0-9]+
print(indices)
```

```
[1, 3]
```

Practical 4:

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.

```
#overloading

class HouseInfo:
    def house_information(self,*args):
        print("House last checked on: ")
        for i in args:
            print(i)
        print()

x=HouseInfo()

x.house_information("2014-06-26")
x.house_information("2000-06-01","2017-05-22")
x.house_information("2001-01-12","2001-11-09","2014-09-11")
```

```
House last checked on:
2014-06-26

House last checked on:
```

```
2000-06-01
2017-05-22

House last checked on:
2001-01-12
2001-11-09
2014-09-11
```

```
#overriding

class DistanceFromAirport:
    def rent(self):
        print("Near the Airport")

class areaLocality:
    def rent(self):
        print("Jacksonville")

x=DistanceFromAirport()
y=areaLocality()
x.rent()
y.rent()
```

```
Near the Airport
Jacksonville
```

```
#sub class

class melbourne():
    def __init__(self,houseSize,housePrice):
        self.houseSize=houseSize
        self.housePrice=housePrice
    def print1(self):
        print("Melbourne :\nHouse size : "+str(self.houseSize)+


class albert_road_dr(melbourne):
    def print2(self):
        print("Albert Drive, Melbourne :\nHouse size : "+str(se

x=albert_road_dr("170000 sq.ft","$1.7M")
```

```
x.print1()
x.print2()
```

```
Melbourne :
House size : 170000 sq.ft
Rent per month : $1.7M

Albert Drive, Melbourne :
House size : 170000 sq.ft
Rent per month : $1.7M
```

```
#multilevel inheritance

class mumbai:
    def __init__(self,house_size,house_price):
        self.house_size=house_size
        self.house_price=house_price
    def mumbai_info(self):
        print("House in Mumbai :\nHouse size : "+str(self.house

class bandra(mumbai):
    def bandra_info(self):
        print("House in Bandra, Mumbai :\nMHouse size : "+str(s

class bandra_west(bandra):
    def bandra_west_info(self):
        print("House in Bandra West, Mumbai :\nHouse size : "+s

house=bandra_west("3400","450000")
house.mumbai_info()
house.bandra_info()
house.bandra_west_info()
```

```
#multilevel inheritance

class melbourne:
    def __init__(self,house_size,house_price):
        self.house_size=house_size
        self.house_price=house_price
    def melbourneInformation(self):
        print("Melbourne :\nHouse size : "+str(self.house_size)

class albert_drive(melbourne):
    def localInformation(self):
        print("Albert_drive, melbourne :\nMHouse size : "+str(s

class bandra_west(albert_drive):
    def bandra_west_info(self):
        print("Albert_drive , melbourne :\nHouse size : "+str(s

X=bandra_west("4500","$600k")
X.melbourneInformation()
X.localInformation()
X.bandra_west_info()
```

```
Melbourne :
House size : 4500
Rent per month : $600k

Albert_drive, melbourne :
MHouse size : 4500
Rent per month : $600k

Albert_drive , melbourne :
House size : 4500
Rent per month : $600k
```

Practical 5:

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

```
string1="Melbourne Housing"
string2="Snapshot"
x=string2.replace("Snapshot", "")
print(x)
```

```
#insert
def insertString(string2):
  splitString = string1.split()
  print(splitString) #intermediate result
  splitString.insert(1, string2)
  finalString = " ".join(splitString)
  print(finalString)
```

```
insertString(string2)

#delete
def delete(string):
  del string
  #print(string)

delete(string2)

# Updating
def update(str):
    str=input("Enter a string to update ")
    print(str)

update(string2)

# Retrieving
def retrieve(str):
    print(str[6:])

retrieve(string2)

# Indexing
def index(str):
    print(str.index("Housing"))

index(string1)

#Slicing
def slicing(str):
    slic = str[:4] + str[5:]
    print("Printing Slice  ::::   ",slic)
    return slic

# Concatenation in string
def concatenate(str, function):
    final= str[:4] + function(string2) + str[4:]
    print(final)

print("Printing concatenation with slicing call ", concatenate(
# Joining
def join(str):
    str2= "Market"
    l=[str,str2]
    print("@".join(l))

print("Running Print statement with output  ::::   ",slicing(st
```

```
['Melbourne', 'Housing']
Melbourne Snapshot Housing
Enter a string to updateYellow
Yellow
ot
10
Printing Slice  ::::    Snaphot
MelbSnaphotourne Housing
Printing concatenation with slicing call  None
Printing Slice  ::::    Melburne Housing
Running Print statement with output  ::::    Melburne Housing
```

Practical-6:

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

```
def insert(pos,key):
  list1.insert(pos,key)
  return list1

def update(pos,key,x):
  if x == 0:
    tempVariable = insert(pos,key)
  else:
    tempVariable = delete(key)
  return tempVariable

def retrieve(pos):
  tempVariable = list1[pos]
  return tempVariable

def delete(key):
  list1.remove(key)
  return list1
```

```
def indexing(key):
  tempVariable = list1.index(key)
  return tempVariable

def concating(list1,l2):
  l3 = list1 + l2
  return l3

def slicing(key,tempVariable):
  c = list1[key:tempVariable]
  return c

def join():
  global li
  print(input("enter an element: ").join(li))

list1=["Airport West", 84000, 3067, "Jellis", "Nelson", "Ashwoo
list2=["Airport 97 Runway"]
print("List : " + str(list1) + "\n")


def switch(argument):
    switcher={
        "1":print(insert(1,23)),
        "2":delete("Jellis"),
        "3":update(1,84000,45),
        "4":retrieve(4),
        "5":print(indexing(3067)),
        "6":slicing(3,45),
        "7":concating(list1,list2),
        "8":join,
    }


print("""Select the operation from the following:
        1. Insert
        2. Delete
        3. Update
        4. Retrieve
        5. Index
        6. Slice
        7. Concatenate
        8. Join
""")
arg= input("Enter one of the above : ")
print()

switch(arg)
```

```
List : ['Airport West', 84000, 3067, 'Jellis', 'Nelson', 'Ashwood']

Select the operation from the following:
        1. Insert
        2. Delete
        3. Update
        4. Retrieve
        5. Index
        6. Slice
        7. Concatenate
        8. Join

Enter one of the above : 1

['Airport West', 23, 84000, 3067, 'Jellis', 'Nelson', 'Ashwood']
2
```

Practical-7:

Write a program to insert, delete, update, retrieve, indexing, slicing, concatenation 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")),in
    tuple1=tuple(tuple1)
    print("Tuple before 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 before deletion : " + str(tuple1))
```

```
def tup_update():
    global tuple1
    print("Tuple before updating : " + str(tuple1))
    tup1=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 inde

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))

def switch(argument):
    switcher={
        "1":tup_insert,
        "2":tup_delete,
        "3":tup_update,
        "4":tup_retrieve,
        "5":tup_indexing,
        "6":tup_slicing,
        "7":tup_concatenation,
    }
    r=switcher.get(argument)()


tuple1=("Airport West", "Jellis", "Nelson", "Ashwood")
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()

switch(arg)
```

```
Tuple : ('Airport West', 84000, 3067, 'Jellis', 'Nelson', 'Ashwood')

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 : 2

Tuple before deletion : ('Airport West', 84000, 3067, 'Jellis', 'Nelson', 'Ashwood')
Enter the element to be deleted : Jellis
Tuple before deletion : ('Airport West', 84000, 3067, 'Nelson', 'Ashwood')
```

Practical-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

```
s1 = set({"Melbourne"})
```

```
s2 = set({"Housing Snapshot"})

def set_intersection():
    return s1.intersection(s2)

def set_union():
    global s1,s2
    return s1.union(s2)

def set_difference():
    global s1,s2
    return s1.difference(s2)

def set_symmetric_difference():
    global s1,s2
    return s1.symmetric_difference(s2)

def set_subset():
    global s1,s2
    return s1.issubset(s2)

def set_superset():
    global s1,s2
    return s1.issuperset(s2)

def set_disjoint():
    global s1,s2
    return s1.isdisjoint(s2)

def switch(argument):
    switcher={
        "1": print(set_intersection()),
        "2": print(set_union()),
        "3": print(set_difference()),
        "4": print(set_symmetric_difference()),
        "5": print(set_subset()),
        "6": print(set_superset()),
        "7": print(set_disjoint()),
    }
```

```
x= input("Enter a value between 1-7     ::::      ")
switch(x)
```

```
Enter a value between 1-7     ::::      7
set()
{'Melbourne', 'Housing Snapshot'}
{'Melbourne'}
{'Melbourne', 'Housing Snapshot'}
False
False
```

Practical 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 the Index for Insertion : ")), (i
    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
    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))

def dict_update():
    global dict1
    print("Tuple before updating : " + str(dict1))
    key_to_replace=(input("Enter the key to be replaced : "))
    value_to_replace=input("Enter the value : ")
    dict1.update({key_to_replace:value_to_replace})
    print("Tuple after updating : " + str(dict1))
```

```
def dict_retrieve():
    global dict1
    print("Dictionary retrieved : " + str(dict1))

def dict_indexing():
    global dict1
    k=input("Enter the key to get its value : ")
    print(dict1.get(k))

def dict_slicing():
    global dict1
    start=int(input("Enter the start position : "))
    end=int(input("Enter the end position : "))
    res=dict()
    for i in dict1:
        res[i]=dict1[i][start:end]
    print(res)

def dict_concatenation():
    global dict1
    print("Tuple before concatenation : " + str(dict1))
    key_to_concatenate=(input("Enter the key to be concatenated
    value_to_concatenate=input("Enter the value to be concatena
    dict1.update({key_to_concatenate:value_to_concatenate})
    print("Tuple after concatenation : " + str(dict1))


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)()
```

```
dict1={"Airport West":"3143","Jellis":"700,000","Ashwood":"Nels
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 a number to preform a operation : ")
print()

switch(arg)
```

```
Dictionary : {'Airport West': '3143', 'Jellis': '700,000', 'Ashwood': 'Nelson Bay'}

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

Enter a number to preform a operation : 6

Enter the start position : 2
Enter the end position : 5
{'Airport West': '43', 'Jellis': '0,0', 'Ashwood': 'lso'}
```

CO-2 Assignment:

Practical-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
f=files.upload()
```

```
import pandas as pd
import numpy as np

results = pd.read_csv('melbourne_housing_snapshot.csv')

print("Number of Rows = "+str(results.shape[0]))
print("Number of Coloums = "+str(results.shape[1]))
print("Number of missing values for each attribute = \n"+str(re
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Di
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	
...	...	...	...	...	...	...	...	...	
13575	Wheelers Hill	12 Strada Cr	4	h	1245000.0	S	Barry	26/08/2017	
13576	Williamstown	77 Merrett	3	h	1031000.0	SP	Williams	26/08/2017	

Practical 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="https://www.makaan.com/"
r=requests.get(URL)
soup=BeautifulSoup(r.content,'html5lib')
```

```
print(f'{soup.h1.name} : {soup.h1.text}')
print(f'{soup.h3.name} : {soup.h3.text}')
print(f'{soup.li.name} : {soup.li.text}')
```

```
h1 : India's only real estate platform with 10,000+ highly rated sellers
h3 : Popular localities in {"templateFromPromise":true,"hideLabel":true}
li : Buy
```

CO-3 Assignment:

Practical 12:

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

```
from IPython.display import Image
Image(filename='image.png', width=800, height=600)
```

```
from IPython.display import Image
Image(url='https://sf.ezoiccdn.com/ezoimgfmt/i0.wp.com/globalfi
```

```
from IPython.display import HTML
# Youtube
HTML('<iframe width="560" height="315" src="https://www.youtube
```

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 melb_data.csv

- **melb_data.csv**(text/csv) - 2091239 bytes, last modified: 11/20/2022 - 100% done

```
Saving melb_data.csv to melb_data.csv
```

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

1. List item
2. List item

```
import pandas as pd

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

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5
		40							

```
df.index
```

```
RangeIndex(start=0, stop=13580, step=1)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         13580 non-null  object 
 1   Address        13580 non-null  object 
 2   Rooms          13580 non-null  int64  
 3   Type           13580 non-null  object 
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  object 
 6   SellerG        13580 non-null  object 
 7   Date           13580 non-null  object 
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13518 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   7130 non-null   float64
 15  YearBuilt      8205 non-null   float64
 16  CouncilArea    12211 non-null  object 
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object 
 20  Propertycount  13580 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 2.2+ MB
```

```
df.describe()
```

	Rooms	Price	Distance	Postcode	Bedroom2	Bathr
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.0000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.0000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.0000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.0000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.0000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.0000

```
df.tail()
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Dist
13575	Wheelers Hill	12 Strada Cr	4	h	1245000.0	S	Barry	26/08/2017	
13576	Williamstown	77 Merrett Dr	3	h	1031000.0	SP	Williams	26/08/2017	
13577	Williamstown	83 Power St	3	h	1170000.0	S	Raine	26/08/2017	
		96							

```
df.dtypes
```

```
Suburb            object
Address           object
Rooms              int64
Type              object
Price            float64
Method            object
SellerG           object
Date              object
Distance         float64
Postcode         float64
Bedroom2         float64
Bathroom         float64
Car              float64
Landsize         float64
BuildingArea     float64
YearBuilt        float64
CouncilArea       object
Lattitude        float64
Longtitude       float64
Regionname        object
Propertycount    float64
dtype: object
```

```
df.shape
```

```
(13580, 21)
```

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

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

```
Data frame before removing duplicates
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Di
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	
...	...	...	...	...	...	...	...	...	
13575	Wheelers Hill	12 Strada Cr	4	h	1245000.0	S	Barry	26/08/2017	
13576	Williamstown	77 Merrett Dr	3	h	1031000.0	SP	Williams	26/08/2017	
13577	Williamstown	83 Power	3	h	1170000.0	S	Raine	26/08/2017	

```
df.duplicated()
```

```
0        False
1        False
2        False
3        False
4        False
         ...
13575    False
13576    False
13577    False
13578    False
13579    False
Length: 13580, dtype: bool
```

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

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

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Di
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	
3	Abbotsford	40 Federation La	3	h	850000.0	PI	Biggin	4/03/2017	
4	Abbotsford	55a Park St	4	h	1600000.0	VB	Nelson	4/06/2016	
...	...	...	...	...	...	...	...	...	

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

```
df_cat=pd.Series(['SellerG','Distance','Landsize'],dtype= 'cate
print(df_cat)
```

```
0     SellerG
1    Distance
2    Landsize
dtype: category
Categories (3, object): ['Distance', 'Landsize', 'SellerG']
```

```
activities = pd.Series(pd.Categorical(['SellerG','Distance','La
print(activities)
```

```
0     SellerG
1    Distance
2    Landsize
dtype: category
Categories (3, object): ['Distance', 'Landsize', 'SellerG']
```

```
df_cat=pd.Series(['city','discipline','Address'],dtype= 'catego
house=pd.Series(pd.Categorical(['SellerG','Distance','Landsize'
activities.cat.categories=['new_Event','new_city','new_discipli
house.cat.categories=activities.cat.categories
print(house)
```

```
0    NaN
1    NaN
2    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/p
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
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dis
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (fr
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'))
```

```
2022-11-20 10:32:38.725297
Sun, 20 November 2022
10:32:38
```

(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 :
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcoc
0	False	False	False	False	False	False	False	False	False	Fals
1	False	False	False	False	False	False	False	False	False	Fals
2	False	False	False	False	False	False	False	False	False	Fals
13576	False	False	False	False	False	False	False	False	False	Fals

```
print("\nTotal number of missing values in entire data framme :
```

```
Total number of missing values in entire data framme : 13256
```

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

```
#Selecting Data
# Selecting columns

show_detail=df[["Distance"]]
show_detail.head()
```

	Distance
0	2.5
1	2.5
2	2.5
3	2.5
4	2.5

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

	Distance
0	2.5
1	2.5

```
show_detail=df[(df["Address"]=="Abbotsford")&(df["Bathroom"]==4
show_detail.head()
```

Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance	Postcode	...

0 rows × 21 columns

6 2.5

Slicing dataFrame using loc and iloc:

Syntax: loc[row labels, columns labels]

9 2.5

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

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance
1	Abbotsford	25 Bloomburg St	2	h	1035000.0	S	Biggin	4/02/2016	2.5
2	Abbotsford	5 Charles St	3	h	1465000.0	SP	Biggin	4/03/2017	2.5

```
#Slice columns by label.
df.loc[:, "Distance":"Postcode"]
```

	Distance	Postcode
0	2.5	3067.0

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

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	Distance
0	Abbotsford	85 Turner St	2	h	1480000.0	S	Biggin	3/12/2016	2.5
1	Abbotsford	25 Bloomburg	2	h	1035000.0	S	Biggin	4/02/2016	2.5

13576 6.8 3016.0

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

	Address	Rooms
0	85 Turner St	2
1	25 Bloomburg St	2
2	5 Charles St	3
3	40 Federation La	3
4	55a Park St	4
...	...	...
13575	12 Strada Cr	4
13576	77 Merrett Dr	3
13577	83 Power St	3
13578	96 Verdon St	4
13579	6 Agnes St	4

13580 rows × 2 columns

Dicing dataFrame using loc and iloc:

```
df.loc[0:5, "Distance":"Lattitude"]
```

	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Co
0	2.5	3067.0	2.0	1.0	1.0	202.0	NaN	NaN	
1	2.5	3067.0	2.0	1.0	0.0	156.0	79.0	1900.0	

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

	Address	Rooms
0	85 Turner St	2
1	25 Bloomburg St	2
2	5 Charles St	3
3	40 Federation La	3
4	55a Park St	4
5	129 Charles St	2

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

```
sort_df=df.sort_values(by=['Distance','Propertycount'],ascendin
sort_df
```

	Suburb	Address	Rooms	Type	Price	Method	SellerG	Date	D:
9620	Melbourne	19/300 King St	2	u	740000.0	VB	MICM	17/06/2017	
10393	Melbourne	1814/250 Elizabeth St	2	u	720000.0	S	Harcourts	27/05/2017	

Aggregating Data

Queen St

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

	Rooms				Price			
	min	sum	mean	max	min	sum	mean	max
Bathroom								
0.0	1	84	2.470588	4	350000.0	3.041500e+07	8.945588e+05	1900000.0
1.0	1	18563	2.471113	6	85000.0	6.453481e+09	8.590896e+05	9000000.0
2.0	1	16728	3.363088	8	320000.0	6.128100e+09	1.232027e+06	7650000.0
3.0	2	3842	4.189749	10	450000.0	1.609915e+09	1.755633e+06	6250000.0
4.0	3	498	4.698113	8	485000.0	2.889403e+08	2.725852e+06	5800000.0
5.0	3	136	4.857143	5	630000.0	7.319700e+07	2.614179e+06	8000000.0
6.0	3	22	4.400000	6	751000.0	1.393100e+07	2.786200e+06	6500000.0
7.0	5	13	6.500000	8	2950000.0	6.850000e+06	3.425000e+06	3900000.0
8.0	4	12	6.000000	8	760000.0	2.960000e+06	1.480000e+06	2200000.0

9 rows × 48 columns

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

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

	Suburb	Address	Rooms
0	Abbotsford	85 Turner St	2
1	Abbotsford	25 Bloomburg St	2
2	Abbotsford	5 Charles St	3

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

	Suburb	Address	Rooms
3	Abbotsford	40 Federation La	3
4	Abbotsford	55a Park St	4
5	Abbotsford	129 Charles St	2

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

	Suburb	Address	Rooms
60	Airport West	174 Parer Rd	2
61	Airport West	138 Victory Rd	3
62	Airport West	75 King St	3
63	Airport West	6 Kittyhawk St	4
64	Airport West	478 Fullarton Rd	3
65	Airport West	144 Marshall Rd	3
66	Airport West	106 Parer Rd	3
67	Airport West	3/7 South Rd	3
68	Airport West	37 North St	3
69	Airport West	10 Hilbert Rd	3
70	Airport West	110 Halsey Rd	3
71	Airport West	2/13 North St	2
72	Airport West	105a Victory Rd	3
73	Airport West	117 Marshall Rd	3
74	Airport West	34 Moorna Dr	4
75	Airport West	33 North St	3
76	Airport West	49 Roberts Rd	4
77	Airport West	85 Roberts Rd	4
78	Albert Park	105 Kerferd Rd	2
79	Albert Park	85 Richardson St	2

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

	Suburb	Address	Rooms
0	Abbotsford	85 Turner St	2
1	Abbotsford	25 Bloomburg St	2
2	Abbotsford	5 Charles St	3
3	Abbotsford	40 Federation La	3
4	Abbotsford	55a Park St	4
5	Abbotsford	129 Charles St	2
60	Airport West	174 Parer Rd	2
61	Airport West	138 Victory Rd	3
62	Airport West	75 King St	3
63	Airport West	6 Kittyhawk St	4
64	Airport West	478 Fullarton Rd	3
65	Airport West	144 Marshall Rd	3
66	Airport West	106 Parer Rd	3
67	Airport West	3/7 South Rd	3
68	Airport West	37 North St	3
69	Airport West	10 Hilbert Rd	3
70	Airport West	110 Halsey Rd	3
71	Airport West	2/13 North St	2
72	Airport West	105a Victory Rd	3
73	Airport West	117 Marshall Rd	3
74	Airport West	34 Moorna Dr	4
75	Airport West	33 North St	3
76	Airport West	49 Roberts Rd	4
77	Airport West	85 Roberts Rd	4

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

	Suburb	Address	Rooms	Type	Price
0	Abbotsford	85 Turner St	2	h	1480000.0
1	Abbotsford	25 Bloomburg St	2	h	1035000.0
2	Abbotsford	5 Charles St	3	h	1465000.0
3	Abbotsford	40 Federation La	3	h	850000.0

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

	Suburb	Address	Rooms	Type	Price
0	Abbotsford	85 Turner St	2	NaN	NaN
1	Abbotsford	25 Bloomburg St	2	NaN	NaN
2	Abbotsford	5 Charles St	3	NaN	NaN
3	Abbotsford	40 Federation La	3	NaN	NaN
4	Abbotsford	55a Park St	4	NaN	NaN
5	Abbotsford	129 Charles St	2	NaN	NaN
0	Abbotsford	85 Turner St	2	h	1480000.0
1	Abbotsford	25 Bloomburg St	2	h	1035000.0
2	Abbotsford	5 Charles St	3	h	1465000.0
3	Abbotsford	40 Federation La	3	h	850000.0

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 melb_data.csv

- **melb_data.csv**(text/csv) - 2091239 bytes, last modified: 11/20/2022 - 100% done

```
Saving melb_data.csv to melb_data (2).csv
```

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

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

```
             Suburb           Address  Rooms Type      Price Method  \
0        Abbotsford      85 Turner St      2    h  1480000.0      S
1        Abbotsford   25 Bloomburg St      2    h  1035000.0      S
2        Abbotsford      5 Charles St      3    h  1465000.0     SP
3        Abbotsford  40 Federation La      3    h   850000.0     PI
4        Abbotsford       55a Park St      4    h  1600000.0     VB
```

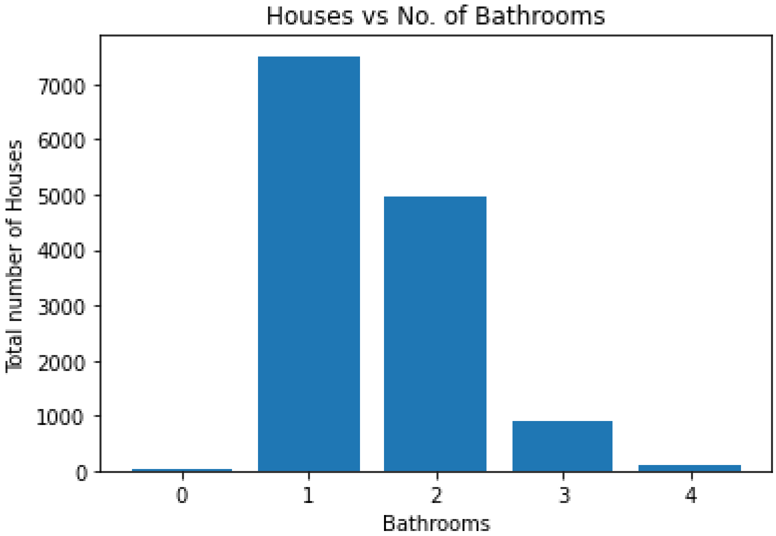
```
...              ...             ...    ...  ...        ...    ...
13575  Wheelers Hill      12 Strada Cr      4    h  1245000.0      S
13576   Williamstown     77 Merrett Dr      3    h  1031000.0     SP
13577   Williamstown       83 Power St      3    h  1170000.0      S
13578   Williamstown      96 Verdon St      4    h  2500000.0     PI
13579     Yarraville         6 Agnes St      4    h  1285000.0     SP

        SellerG        Date  Distance  Postcode  ...  Bathroom  Car  Landsize  \
0        Biggin   3/12/2016       2.5    3067.0  ...       1.0  1.0     202.0
1        Biggin   4/02/2016       2.5    3067.0  ...       1.0  0.0     156.0
2        Biggin   4/03/2017       2.5    3067.0  ...       2.0  0.0     134.0
3        Biggin   4/03/2017       2.5    3067.0  ...       2.0  1.0      94.0
4        Nelson   4/06/2016       2.5    3067.0  ...       1.0  2.0     120.0
...         ...         ...       ...       ...  ...       ...  ...       ...
13575     Barry  26/08/2017      16.7    3150.0  ...       2.0  2.0     652.0
13576  Williams  26/08/2017       6.8    3016.0  ...       2.0  2.0     333.0
13577     Raine  26/08/2017       6.8    3016.0  ...       2.0  4.0     436.0
13578   Sweeney  26/08/2017       6.8    3016.0  ...       1.0  5.0     866.0
13579   Village  26/08/2017       6.3    3013.0  ...       1.0  1.0     362.0

       BuildingArea  YearBuilt  CouncilArea Lattitude  Longtitude  \
0               NaN        NaN        Yarra -37.79960   144.99840
1              79.0     1900.0        Yarra -37.80790   144.99340
2             150.0     1900.0        Yarra -37.80930   144.99440
3               NaN        NaN        Yarra -37.79690   144.99690
4             142.0     2014.0        Yarra -37.80720   144.99410
...             ...        ...          ...       ...         ...
13575           NaN     1981.0          NaN -37.90562   145.16761
13576         133.0     1995.0          NaN -37.85927   144.87904
13577           NaN     1997.0          NaN -37.85274   144.88738
13578         157.0     1920.0          NaN -37.85908   144.89299
13579         112.0     1920.0          NaN -37.81188   144.88449

                       Regionname Propertycount
0           Northern Metropolitan        4019.0
1           Northern Metropolitan        4019.0
2           Northern Metropolitan        4019.0
3           Northern Metropolitan        4019.0
4           Northern Metropolitan        4019.0
...                           ...           ...
13575  South-Eastern Metropolitan        7392.0
13576         Western Metropolitan        6380.0
13577         Western Metropolitan        6380.0
13578         Western Metropolitan        6380.0
13579         Western Metropolitan        6543.0

[13580 rows x 21 columns]
```

```
a = df.Bathroom.value_counts().sort_values(ascending=False).hea
a
```

```
1.0    7512
2.0    4974
3.0     917
4.0     106
0.0      34
Name: Bathroom, dtype: int64
```

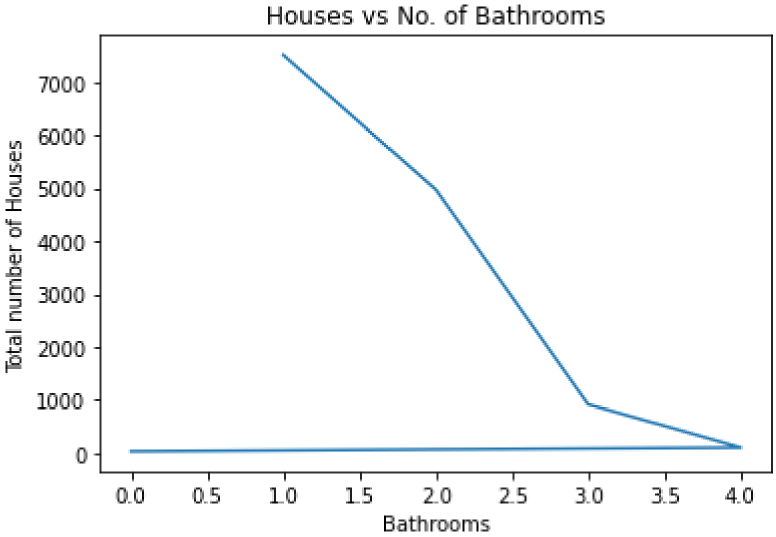
Bar Graph

```
plt.bar(a.index, a)
plt.title("Houses vs No. of Bathrooms")
plt.xlabel("Bathrooms")
plt.ylabel("Total number of Houses")
plt.show()
```



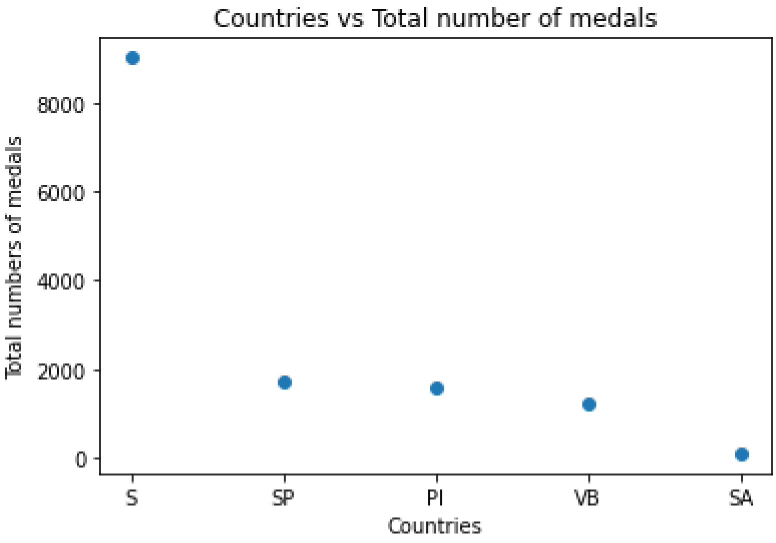
Line Graph

```
plt.plot(a)
plt.title("Houses vs No. of Bathrooms")
plt.xlabel("Bathrooms")
plt.ylabel("Total number of Houses")
plt.show()
```



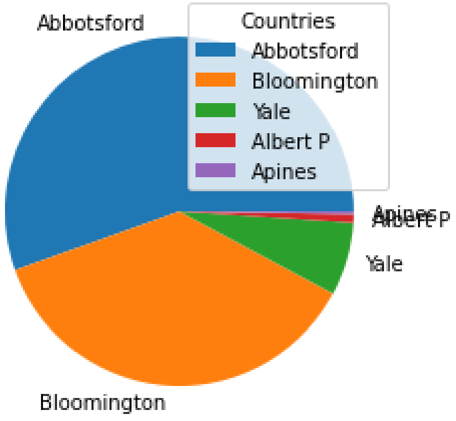
Scatter Plot

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



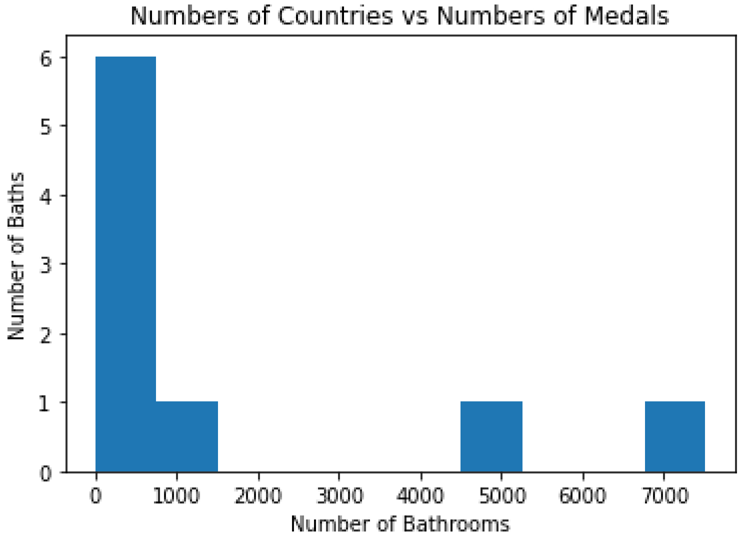
Pie Chart

```
from turtle import title

mylabels = ["Abbotsford", "Bloomington", "Yale", "Albert P", "A
plt.pie(a, labels=mylabels)
plt.legend(title = "Countries")
plt.show()
```



Histogram

```
c = df.Bathroom.value_counts().sort_values(ascending=False).hea
plt.hist(c)
plt.ylabel("Number of Baths")
plt.xlabel("Number of Bathrooms")
plt.show()
```



Box Plot

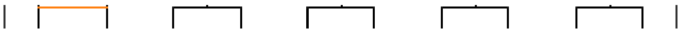
```
data1 = df["Rooms"].value_counts().head(5)
data2 = df["Rooms"].value_counts().head(15)
data3 = df["Rooms"].value_counts().head(700)
data4 = df["Rooms"].value_counts().head(35)
data5 = df["Rooms"].value_counts().head(200)
boxplot_data = [data1, data2, data3, data4, data5]
plt.boxplot(boxplot_data)
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleDepr
  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("melb_data.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["Rooms"]

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

```
0       -0.981463
1       -0.981463
2        0.064876
3        0.064876
4        1.111216
           ...
13575    1.111216
13576    0.064876
13577    0.064876
13578    1.111216
13579    1.111216
Name: Rooms, Length: 13580, dtype: float64
```

Practical 18 Write a program to calculate the TF-IDF score of the given documents using NLTK.

```
corpus1 = "Although staff at Dublin’s Twitter office has shrunk
corpus1
```

```
'Although staff at Dublin’s Twitter office has shrunk by about a third following the
recent exodus, the spat is highlighting a growing problem for the Irish economy as i
t faces the prospect of recession. Dublin has flourished in recent years by creating
```

```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer


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

```
 'spat',
 'is',
 'highlighting',
 'a',
 'growing',
 'problem'
 'problem',
 'for',
 'the',
 'Irish',
 'economy',
 'as',
 'it',
 'faces',
 'the',
 'prospect',
 'of',
 'recession.',
 'Dublin',
 'has',
 'flourished',
 'in',
 'recent',
 'years',
 'by',
 'creating',
 'attractive',
 'conditions',
 'for',
 'major',
 'companies',
 'to',
 'set',
 'up',
 'big',
 'offices.',
 'The',
 'city',
 'of',
 '1.3',
 'million',
 'will',
 'quickly',
 'lose',
 'its',
 'edge',
 'if',
 'it',
 'can’t',
 'provide',
 'housing',
 'for',
 'firms',
 'eager',
 'to',
 'get',
 'employees',
 'back',
 'to',
 'work.']
```

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

```
 'spat',
 'is',
 'highlighting'
 'highlighting',
 'a',
 'growing',
 'problem',
 'for',
 'the',
 'Irish',
 'economy',
 'as',
 'it',
 'faces',
 'the',
 'prospect',
 'of',
 'recession.',
 'Dublin',
 'has',
 'flourished',
 'in',
 'recent',
 'years',
 'by',
 'creating',
 'attractive',
 'conditions',
 'for',
 'major',
 'companies',
 'to',
 'set',
 'up',
 'big',
 'offices.',
 'The',
 'city',
 'of',
 '1.3',
 'million',
 'will',
 'quickly',
 'lose',
 'its',
 'edge',
 'if',
 'it',
 'can’t',
 'provide',
 'housing',
 'for',
 'firms',
 'eager',
 'to',
 'get',
 'employees',
 'back',
 'to',
 'work.']
```

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

```
unique_words
```

```
    'The',
    'Twitter',
    'a',
    'about',
    'as',
    'at',
    'attractive',
    'back',
    'big',
    'by',
    'can’t',
    'city',
    'companies',
    'conditions',
    'creating',
    'eager',
    'economy',
    'edge',
    'employees',
    'exodus,',
    'faces',
    'firms',
    'flourished',
    'following',
    'for',
    'get',
    'growing',
    'has',
    'highlighting',
    'housing',
    'if',
    'in',
    'is',
    'it',
    'its',
    'lose',
    'major',
    'million',
    'of',
    'office',
    'offices.',
    'problem',
    'prospect',
    'provide',
    'quickly',
    'recent',
    'recession.',
    'set',
    'shrunk',
    'spat',
    'staff',
    'the',

    'third',
    'to',
    'up',
    'will',
    'work.',
    'years'}
    years }
```

```
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:
  num_words_2[word] += 1


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
```

```
     'shrunk': 0.013157894736842105,
     'provide': 0.013157894736842105,
     'third': 0.013157894736842105,
     'million': 0.013157894736842105,
     'about': 0.013157894736842105,
     'back': 0.013157894736842105,
     'major': 0.013157894736842105,
     'conditions': 0.013157894736842105,
     'will': 0.013157894736842105,
     'if': 0.013157894736842105,
     'work.': 0.013157894736842105,
     'office': 0.013157894736842105,
     'staff': 0.013157894736842105,
     'is': 0.013157894736842105,
     'attractive': 0.013157894736842105,
     'lose': 0.013157894736842105,
     'edge': 0.013157894736842105,
     'recent': 0.02631578947368421,
     'firms': 0.013157894736842105,
     'Dublin': 0.013157894736842105,
     '1.3': 0.013157894736842105,
     'problem': 0.013157894736842105,
     'has': 0.02631578947368421,
     'of': 0.02631578947368421,
     'Irish': 0.013157894736842105,
     'set': 0.013157894736842105,
     'big': 0.013157894736842105,

     'city': 0.013157894736842105,
     'faces': 0.013157894736842105,
     'housing': 0.013157894736842105,
     'for': 0.039473684210526314,
     'as': 0.013157894736842105,
     'in': 0.013157894736842105,
      'in': 0.013157894736842105,
     'flourished': 0.013157894736842105,
     'creating': 0.013157894736842105,
     'at': 0.013157894736842105,
     'The': 0.013157894736842105,
     'Although': 0.013157894736842105,
     'a': 0.02631578947368421,
     'can’t': 0.013157894736842105,
     'following': 0.013157894736842105,
     'Twitter': 0.013157894736842105,
     'its': 0.013157894736842105,
     'get': 0.013157894736842105,
     'exodus,': 0.013157894736842105,
     'employees': 0.013157894736842105,
     'spat': 0.013157894736842105,
     'up': 0.013157894736842105,
     'economy': 0.013157894736842105,
     'Dublin’s': 0.013157894736842105,
     'it': 0.02631578947368421,
     'companies': 0.013157894736842105,
     'growing': 0.013157894736842105,
     'offices.': 0.013157894736842105,
     'to': 0.039473684210526314,
     'the': 0.05263157894736842,
     'eager': 0.013157894736842105,
     'by': 0.02631578947368421}
```

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

```
 'shrunk': 0.013157894736842105,
 'provide': 0.013157894736842105,
 'third': 0.013157894736842105,
 'million': 0.013157894736842105,
 'about': 0.013157894736842105,
 'back': 0.013157894736842105,
 'major': 0.013157894736842105,
 'conditions': 0.013157894736842105,
 'will': 0.013157894736842105,
 'if': 0.013157894736842105,
 'work.': 0.013157894736842105,
 'office': 0.013157894736842105,
 'staff': 0.013157894736842105,
 'is': 0.013157894736842105,
 'attractive': 0.013157894736842105,
 'lose': 0.013157894736842105,
 'edge': 0.013157894736842105,
 'recent': 0.02631578947368421,
 'firms': 0.013157894736842105,
 'Dublin': 0.013157894736842105,
 '1.3': 0.013157894736842105,
 'problem': 0.013157894736842105,
 'has': 0.02631578947368421,
 'of': 0.02631578947368421,

 'Irish': 0.013157894736842105,
 'set': 0.013157894736842105,
 'big': 0.013157894736842105,
 'city': 0.013157894736842105,
 'faces': 0.013157894736842105,
 'housing': 0.013157894736842105
 'housing': 0.013157894736842105,
 'for': 0.039473684210526314,
 'as': 0.013157894736842105,
 'in': 0.013157894736842105,
 'flourished': 0.013157894736842105,
 'creating': 0.013157894736842105,
 'at': 0.013157894736842105,
 'The': 0.013157894736842105,
 'Although': 0.013157894736842105,
 'a': 0.02631578947368421,
 'can’t': 0.013157894736842105,
 'following': 0.013157894736842105,
 'Twitter': 0.013157894736842105,
 'its': 0.013157894736842105,
 'get': 0.013157894736842105,
 'exodus,': 0.013157894736842105,
 'employees': 0.013157894736842105,
 'spat': 0.013157894736842105,
 'up': 0.013157894736842105,
 'economy': 0.013157894736842105,
 'Dublin’s': 0.013157894736842105,
 'it': 0.02631578947368421,
 'companies': 0.013157894736842105,
 'growing': 0.013157894736842105,
 'offices.': 0.013157894736842105,
 'to': 0.039473684210526314,
 'the': 0.05263157894736842,
 'eager': 0.013157894736842105,
 'by': 0.02631578947368421}
```

```
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
```

```
 'shrunk': 0.0,
 'provide': 0.0,
 'third': 0.0,
 'million': 0.0,
 'about': 0.0,
 'back': 0.0,
 'major': 0.0,
  'major': 0.0,
 'conditions': 0.0,
 'will': 0.0,
 'if': 0.0,
 'work.': 0.0,
 'office': 0.0,
 'staff': 0.0,
 'is': 0.0,
 'attractive': 0.0,
 'lose': 0.0,
 'edge': 0.0,
 'recent': 0.0,
 'firms': 0.0,
 'Dublin': 0.0,
 '1.3': 0.0,
 'problem': 0.0,
 'has': 0.0,
 'of': 0.0,
 'Irish': 0.0,
 'set': 0.0,
 'big': 0.0,
 'city': 0.0,
 'faces': 0.0,
 'housing': 0.0,
 'for': 0.0,
 'as': 0.0,
 'in': 0.0,
 'flourished': 0.0,
 'creating': 0.0,
 'at': 0.0,
 'The': 0.0,
 'Although': 0.0,
 'a': 0.0,
 'can’t': 0.0,
 'following': 0.0,
 'Twitter': 0.0,
 'its': 0.0,
 'get': 0.0,
 'exodus,': 0.0,
 'employees': 0.0,
 'spat': 0.0,
 'up': 0.0,
 'economy': 0.0,
 'Dublin’s': 0.0,
 'it': 0.0,
 'companies': 0.0,
 'growing': 0.0,
 'offices.': 0.0,
 'to': 0.0,
 'the': 0.0,
 'eager': 0.0,
 'by': 0.0}
```

```
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
```

```
   years  highlighting  prospect  recession.  quickly  shrunk  provide  third  millic
0    0.0           0.0       0.0         0.0      0.0     0.0      0.0    0.0       0
1    0.0           0.0       0.0         0.0      0.0     0.0      0.0    0.0       0

2 rows × 63 columns
```