CGAS_Assignment3_MT20075_Apriori

November 1, 2021

[ ]:
```
# main imports for the assignment
import pandas as pd
import numpy as np
import json
```

1 Converting Json to Pandas DataFrame

[ ]:
```
# Opening JSON file
f = open('/content/drive/MyDrive/CGAS/archive/train.json',)

# returns JSON object as
# a dictionary
data = json.load(f)

# Closing file
f.close()
```

[ ]:
```
# creating dataframe from Kaggle recipe data json file
df = pd.DataFrame.from_dict(pd.json_normalize(data), orient='columns')
#df = pd.DataFrame.from_dict('/content/drive/MyDrive/CGAS/archive/train.json')
df
```

[ ]:
```
          id      cuisine                                        ingredients
0      10259        greek  [romaine lettuce, black olives, grape tomatoes...
1      25693  southern_us  [plain flour, ground pepper, salt, tomatoes, g...
2      20130     filipino  [eggs, pepper, salt, mayonaise, cooking oil, g...
3      22213       indian                 [water, vegetable oil, wheat, salt]
4      13162       indian  [black pepper, shallots, cornflour, cayenne pe...
...      ...          ...                                                ...
39769  29109        irish  [light brown sugar, granulated sugar, butter, ...
39770  11462      italian  [KRAFT Zesty Italian Dressing, purple onion, b...
39771   2238        irish  [eggs, citrus fruit, raisins, sourdough starte...
39772  41882      chinese  [boneless chicken skinless thigh, minced garli...
39773   2362      mexican  [green chile, jalapeno chilies, onions, ground...

[39774 rows x 3 columns]
```

[ ]:
```
# number of recipes = number of rows in recipe data
len(np.array(df['id'].unique()))
```

[ ]:
```
39774
```

[ ]:
```
# for the longest ingredient list in recipe data and the number of recipes
max_len = 0
count = 0
for i in df['ingredients']:
  count += 1
  length = len(np.array(i))
  if length > max_len:
    max_len = length
  else:
    continue

print(max_len)
print(count)
```

```
65
39774
```

[ ]:
```
# creating ingredients data by expanding ingredient list in each row of the␣
 ↪dataframe
ingredients_data = pd.DataFrame(df['ingredients'].tolist())
#ingredients_data
```

[ ]:
```
# displaying the ingredient data after filling none values with numpy Nan
ingredients_data = ingredients_data.fillna(value=np.nan)
ingredients_data
```

[ ]:
```
                                  0                 1   ...   63   64
0                     romaine lettuce       black olives  ...  NaN  NaN
1                         plain flour      ground pepper  ...  NaN  NaN
2                                eggs             pepper  ...  NaN  NaN
3                               water      vegetable oil  ...  NaN  NaN
4                         black pepper           shallots  ...  NaN  NaN
...                                ...               ...  ...  ...  ...
39769                light brown sugar  granulated sugar  ...  NaN  NaN
39770    KRAFT Zesty Italian Dressing       purple onion  ...  NaN  NaN
39771                            eggs       citrus fruit  ...  NaN  NaN
39772  boneless chicken skinless thigh      minced garlic  ...  NaN  NaN
39773                      green chile   jalapeno chilies  ...  NaN  NaN

[39774 rows x 65 columns]
```

[ ]:
```
trans = []
for i in range(0, 39774):
    trans.append([str(ingredients_data.values[i,j]) for j in range(0, 65)])
```

```
# conveting it into an numpy array
trans = np.array(trans)

# checking the shape of the array
print(trans.shape)
print(trans)
```

2 For Q1(a)

2.1 Apriori Algorithm

Preprocessing on the Ingredient Data

[ ]:
```
from mlxtend.preprocessing import TransactionEncoder

te = TransactionEncoder()
ing_data = te.fit_transform(trans)
ing_data = pd.DataFrame(ing_data, columns = te.columns_)
ing_data
# getting the shape of the data
print(ing_data.shape)
```

```
(39774, 6715)
```

Ingredient Data based on ID from the Kaggle Data
Removing Nan column

[ ]:
```
# the data on which apriori algorithm needs to be applied
ing_data = ing_data.drop(['nan'], axis=1)
ing_data
```

[ ]:
```
       (    oz.) tomato sauce  ...  zucchini blossoms
0                         False  ...              False
1                         False  ...              False
2                         False  ...              False
3                         False  ...              False
4                         False  ...              False
...                         ...  ...                ...
39769                     False  ...              False
39770                     False  ...              False
39771                     False  ...              False
39772                     False  ...              False
39773                     False  ...              False

[39774 rows x 6714 columns]
```

For Apriori on the above preprocessed data

[ ]:
```
from mlxtend.frequent_patterns import apriori
```

Creating itemsets for the above data with minimum possible support to check all itemsets

[ ]:
```
# generation of frequent itemsets based on 0.1% minimum support value to␣
 ↪display for itemsets of size max length of 6
frequent_itemsets = apriori(ing_data, min_support = 0.001, use_colnames=True)
frequent_itemsets['length'] = frequent_itemsets['itemsets'].apply(lambda x:␣
 ↪len(x))
frequent_itemsets
```

[ ]:

	support	itemsets	length
0	0.004852	(1% low-fat milk)	1
1	0.001785	(2% reduced-fat milk)	1
2	0.001207	(Alfredo sauce)	1
3	0.001282	(Anaheim chile)	1
4	0.001257	(California bay leaves)	1
...	...	...	...
26071	0.001031	(oil, garam masala, ground turmeric, salt, oni...	6
26072	0.001182	(garam masala, ground turmeric, salt, onions, ...	6
26073	0.001157	(oil, ground turmeric, salt, onions, tomatoes,...	6
26074	0.001182	(oil, garam masala, ground turmeric, salt, oni...	6
26075	0.001483	(oil, pepper, water, salt, onions, garlic)	6

[26076 rows x 3 columns]

2.2 Q1(b)

[ ]:
```
#Printing the itemset of sizes 1,2,3,4 and 5 together with their best support␣
 ↪values

size1_freqitem = frequent_itemsets.loc[(frequent_itemsets['length'] == 1) &␣
 ↪(frequent_itemsets['support'] == max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 1)]['support'])))]['itemsets']
print("The frequent itemset of size 1 is: ")
print(size1_freqitem)
max_support_size1freq = max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 1)]['support']))
print("The best support for this itemset is: ",max_support_size1freq)
print("\n")
size2_freqitem = frequent_itemsets.loc[(frequent_itemsets['length'] == 2) &␣
 ↪(frequent_itemsets['support'] == max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 2)]['support'])))]['itemsets']
print("The frequent itemset of size 2 is: ")
print(size2_freqitem)
max_support_size2freq = max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 2)]['support']))
print("The best support for this itemset is: ",max_support_size2freq)
print("\n")
```

```
size3_freqitem = frequent_itemsets.loc[(frequent_itemsets['length'] == 3) &␣
 ↪(frequent_itemsets['support'] == max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 3)]['support'])))]['itemsets']
print("The frequent itemset of size 3 is: ")
print(size3_freqitem)
max_support_size3freq = max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 3)]['support']))
print("The best support for this itemset is: ",max_support_size3freq)
print("\n")
size4_freqitem = frequent_itemsets.loc[(frequent_itemsets['length'] == 4) &␣
 ↪(frequent_itemsets['support'] == max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 4)]['support'])))]['itemsets']
print("The frequent itemset of size 4 is: ")
print(size4_freqitem)
max_support_size4freq = max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 4)]['support']))
print("The best support for this itemset is: ",max_support_size4freq)
print("\n")
size5_freqitem = frequent_itemsets.loc[(frequent_itemsets['length'] == 5) &␣
 ↪(frequent_itemsets['support'] == max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 5)]['support'])))]['itemsets']
print("The frequent itemset of size 5 is: ")
print(size5_freqitem)
max_support_size5freq = max(np.array(frequent_itemsets.
 ↪loc[(frequent_itemsets['length'] == 5)]['support']))
print("The best support for this itemset is: ",max_support_size5freq)
```

```
The frequent itemset of size 1 is:
912    (salt)
Name: itemsets, dtype: object
The best support for this itemset is:  0.45376376527379697


The frequent itemset of size 2 is:
9101    (onions, salt)
Name: itemsets, dtype: object
The best support for this itemset is:  0.11042389500678836


The frequent itemset of size 3 is:
17900    (onions, garlic, salt)
Name: itemsets, dtype: object
The best support for this itemset is:  0.04035299441846432


The frequent itemset of size 4 is:
24374    (onions, garlic, salt, pepper)
```

```
Name: itemsets, dtype: object
The best support for this itemset is:  0.014104691507014632


The frequent itemset of size 5 is:
25929    (olive oil, pepper, salt, onions, garlic)
Name: itemsets, dtype: object
The best support for this itemset is:  0.004827274098657414
```

2.2.1 Analysis and interpretations:

1. Salt is the most frequent ingredient item in this recipe dataset.
2. The ingredient item pair of salt and onion is the most frequent used pairs of ingredients in recipes found in this recipe data, which is around 11% support for the frequent itemsets of this recipe data.
3. The frequency of onion, garlic and salt ingredients are highest in frequency when it comes to 3-item frequent itemsets (which is approximately 4% i.e. 1605 recipes out of 39774 recipes).
4. The frequency of onion, garlic, salt and pepper ingredients are highest in frequency when it comes to 4-item frequent itemsets (which is approximately 1% i.e. 561 recipes out of 39774 recipes).
5. The frequency of onion, garlic, salt, pepper and olive oil ingredients are highest in frequency when it comes to 5-item frequent itemsets (which is 192 recipes out of 39774 recipes).
6. As the size of the best support value based frequent-itemset increases, we see that one item adds into the set of the frequent-itemset of size 1 item lesser than the succeeding itemset.
7. As the ingredient complexity increases, their frequency decreases and hence the frequency of their itemsets also decreases proportionally. For instance, 'frozen chopped spinach, thawed and squeezed dry' is a complex ingredient requirement that may be less frequently needed in other types of recipes and hence its support value is lesser than the more simplistic ingredients like 'garlic', 'salt' etc.

3 Generating PDF Report of Jupyter Notebook File

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('CGAS_Assignment3_MT20075_Apriori.ipynb')
```

```
--2021-11-01 18:01:54--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
```

```
Saving to: colab_pdf.py

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2021-11-01 18:01:54 (21.5 MB/s) - colab_pdf.py saved [1864/1864]

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```

[ ]:

CGAS_Assignment3_MT20075_RandomControls

November 2, 2021

1 Data Preprocessing...

[1]:
```
# main library imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import json
```

[2]:
```
# Opening JSON file
f = open('/content/drive/MyDrive/CGAS/archive/train.json',)

# returns JSON object as
# a dictionary
data = json.load(f)

# Closing file
f.close()
```

[3]:
```
# main recipe dataframe imported as json file from Kaggle
df = pd.DataFrame.from_dict(pd.json_normalize(data), orient='columns')
df
```

[3]:

	id	cuisine	ingredients
0	10259	greek	[romaine lettuce, black olives, grape tomatoes...
1	25693	southern_us	[plain flour, ground pepper, salt, tomatoes, g...
2	20130	filipino	[eggs, pepper, salt, mayonaise, cooking oil, g...
3	22213	indian	[water, vegetable oil, wheat, salt]
4	13162	indian	[black pepper, shallots, cornflour, cayenne pe...
...	...	...	...
39769	29109	irish	[light brown sugar, granulated sugar, butter, ...
39770	11462	italian	[KRAFT Zesty Italian Dressing, purple onion, b...
39771	2238	irish	[eggs, citrus fruit, raisins, sourdough starte...
39772	41882	chinese	[boneless chicken skinless thigh, minced garli...
39773	2362	mexican	[green chile, jalapeno chilies, onions, ground...

[39774 rows x 3 columns]

For ingredient information

[4]:
```
#creating dataFrame for all the ingredients used in the recipe data
ingredients_data = pd.DataFrame(df['ingredients'].tolist())
ingredients_data = ingredients_data.fillna(value=np.nan)
ingredients_data
```

[4]:

	0	1	...	63	64
0	romaine lettuce	black olives	...	NaN	NaN
1	plain flour	ground pepper	...	NaN	NaN
2	eggs	pepper	...	NaN	NaN
3	water	vegetable oil	...	NaN	NaN
4	black pepper	shallots	...	NaN	NaN
...	...	...	...	...	...
39769	light brown sugar	granulated sugar	...	NaN	NaN
39770	KRAFT Zesty Italian Dressing	purple onion	...	NaN	NaN
39771	eggs	citrus fruit	...	NaN	NaN
39772	boneless chicken skinless thigh	minced garlic	...	NaN	NaN
39773	green chile	jalapeno chilies	...	NaN	NaN

[39774 rows x 65 columns]

[5]:
```
from collections import Counter
```

[6]:
```
# getting unique ingredients from the recipe data and the count of the unique␣
 ↪ingredients of recipe data
ing_list = []
for ing in Counter(np.array(ingredients_data.values).reshape(-1)).keys():
  ing_list.append(ing)
ing_list.remove(np.nan)
ingredients = np.array(ing_list)
ing_num = len(ingredients)

print(ingredients)
print(ing_num)
```

```
['romaine lettuce' 'black olives' 'grape tomatoes' ... 'lop chong'
 'tomato garlic pasta sauce' 'crushed cheese crackers']
6714
```

For recipe size and recipe size distribution

[ ]:
```
# getting the recipe sizes (ingredient count) of each of 39774 recipes
recipe_sizes = []
#count = 0
for rec in df['ingredients']:
  #count += 1
  recipe_sizes.append(len(rec))
print(recipe_sizes)
print(len(recipe_sizes))
#print(count)
```

For Ingredient Frequency i.e. count of each ingredient usage in the recipe data

[ ]:
```
# storing the ingredient and their count in recipe data as dictionary keys and␣
 ↪corresponding values respectively
ing_dict = Counter(np.array(ingredients_data.values).reshape(-1))
del ing_dict[np.nan]
print(ing_dict)
```

[ ]:
```
# getting the array of ingredient counts from the dictionary created above
ing_freq = []
for ingg in ingredients:
  ing_freq.append(ing_dict[ingg])
print(ing_freq)
print(len(ing_freq))
```

2 Random Control - R0

Function for generating random control r0 to preserve:
* the number of ingredients (i.e. 6714)
* the number of recipes (i.e. 39774)
* the recipe size distribution of the data

[ ]:
```
# shuffling the ingredients array for proper randomization
from sklearn.utils import shuffle
random_ingredients = shuffle(ingredients, random_state=0)
print(random_ingredients)
```

[11]:
```
# creating R0 Random Control Function
def generating_random_recipes_r0(ingredients,recipe_sizes,ing_num):
  random_ingredients_data = []   # to store the new recipes list generated from␣
 ↪randomising the ingredients from the ingredient basket (ingredient array␣
 ↪meta data)
  ing_index = 0
  for ing_size in recipe_sizes:
    new_recipe = []             # to store the randomly selected ingredients␣
 ↪from the ingredient array meta data based on recipe sizes of each of 39774␣
 ↪recipes
    for num in range(ing_size):
      index = (ing_index + num) % ing_num
      #print(index)
      new_recipe.append(ingredients[index])
    random_ingredients_data.append(new_recipe)
    ing_index += ing_size
  new_recipe_data1 = pd.DataFrame()
  new_recipe_data1['ingredients'] = random_ingredients_data
  return new_recipe_data1
```

[12]:
```
random_data1 =␣
 ↪generating_random_recipes_r0(random_ingredients,recipe_sizes,ing_num)
random_data1
```

[12]:
```
                                              ingredients
0       [italian eggplant, chilli paste, tomato juice,...
1       [cauliflower, shichimi togarashi, creamer pota...
2       [navel oranges, flatbread, hamachi fillets, de...
3       [breasts halves, minced peperoncini, frozen se...
4       [persian cucumber, burgundy snails, cashew but...
...                                                   ...
39769   [forest mushroom, rutabaga, sugar syrup, blanc...
39770   [nonfat milk powder, shredded low-fat jarlsber...
39771   [cajun style stewed tomatoes, pasta wagon whee...
39772   [crumb crust, reduced fat milk, sweet turkey s...
39773   [crushed cornflakes, Tipo 00 flour, sourdough ...

[39774 rows x 1 columns]
```

The number of recipes has been preserved. Checking for the number of ingredients and the recipe size distribution.

[16]:
```
# checking whether the number of ingredients was maintained
random_ingredient_data1 = pd.DataFrame(random_data1['ingredients'].tolist())
random_ingredient_data1 = random_ingredient_data1.fillna(value=np.nan)

ing_list2 = []
for ing2 in Counter(np.array(random_ingredient_data1.values).reshape(-1)).
 ↪keys():
  ing_list2.append(ing2)
ing_list2.remove(np.nan)
ingredients2 = np.array(ing_list2)
ing_num2 = len(ingredients2)

#print(ingredients2)
print(ing_num2)

# checking whether the recipe size distribution array was maintained
recipe_sizes_rand1 = []
#count = 0
for rec2 in random_data1['ingredients']:
  #count += 1
  recipe_sizes_rand1.append(len(rec2))
#print(recipe_sizes_rand1)
#print(recipe_sizes)
```

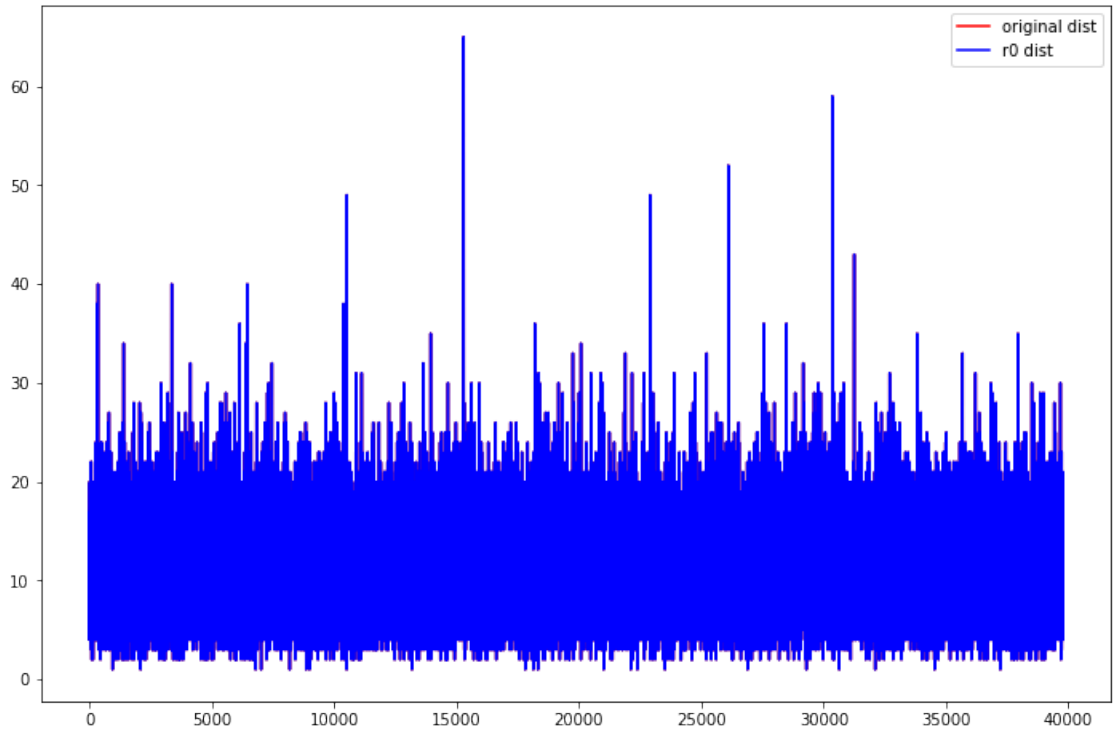
```
6714
```

[14]:
```
# function to compare whether the two arrays are the same
# made to compare the arrays for recipe size distribution
def compare(first, second, size):
    first.sort()
    second.sort()

    for i in range(size):
        if first[i] != second[i]:
            return False

    return True
```

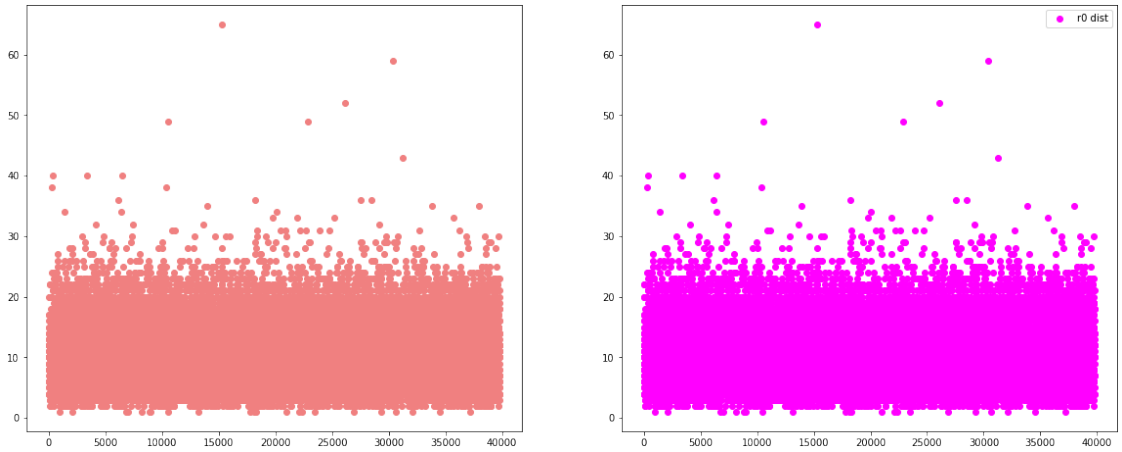
[15]:
```
# comparing the recipe size distribution arrays for the original data and for␣
 ↪the random control R1 generated data respectively
compare(recipe_sizes, recipe_sizes_rand1, 39774)
```

[15]:
```
True
```

[ ]:
```
#Plotting the two distribution plots for comparison
plt.figure(figsize=(12,8))
#plotting original recipe size distribution
plt.plot(recipe_sizes, color='r', label = 'original dist')
#plotting random control - r0 recipe size distribution
plt.plot(recipe_sizes_rand1, color='b', label = 'r0 dist')
plt.legend()
plt.show()
```



[ ]:
```
#scatter plot
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))
axes[0].scatter(range(39774),recipe_sizes,color='lightcoral',label='original␣
 ↪dist')
#plt.legend()
axes[1].scatter(range(39774),recipe_sizes_rand1,color='fuchsia',label='r0 dist')
plt.legend()
plt.show()
```



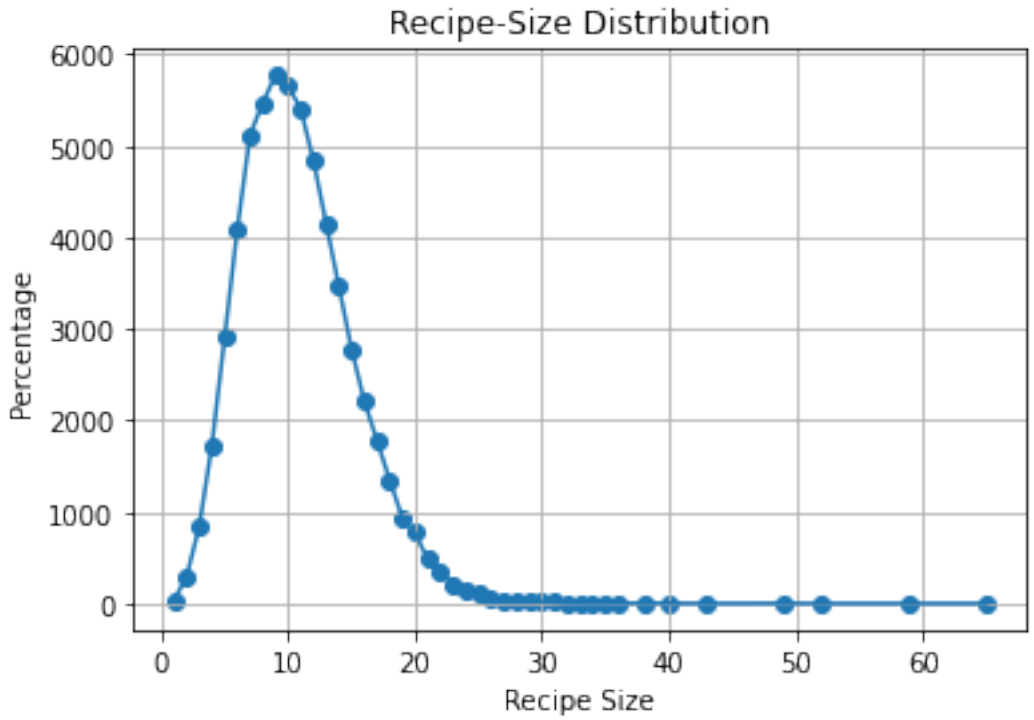
In the above figure, scatter plots for: Left: Original Recipe Size Distribution Right: Random control R0 generated Recipe Size Distribution for comparison.

[ ]:
```
# Generating another set of plots for comparison of recipe size distribution␣
 ↪plot
# Creating normalized data dictionary of recipe sizes for plot
def RecDistributionCuisine(recipe_sizes):
  recipe_size_dict = {} #storing size of each recipe and total number of␣
 ↪recipes having that size
  for ing in recipe_sizes:
    if ing not in recipe_size_dict:
      recipe_size_dict[ing]=1
    else:
      recipe_size_dict[ing]+=1
  #print("\n Recipe Size Distributions: \n", recipe_size_dict)
  #print("df:",df.shape[0])
  for i in recipe_size_dict:
    recipe_size_dict[i] = (recipe_size_dict[i]/65)*100 #normalizing recipe␣
 ↪sizes with total number of recipes and computing percentage
  #print("\n Recipe Size Distribution in Percentage: \n", recipe_size_dict)
  return recipe_size_dict
```

[ ]:
```
# Generate plot from the dictionary created above
def generatePlot2(recipe_size):
  recipe_sort = dict(sorted(recipe_size.items(), key=lambda x: x[0])) #sorting␣
 ↪recipe sizes
  X=[] #storing recipe_sizes
  Y=[] #number of recipes having that size
  for i in recipe_sort:
    X.append(i)
    Y.append(recipe_sort[i])
  plt.scatter(X, Y)
  plt.plot(X,Y)
  plt.title('Recipe-Size Distribution')
  plt.ylabel('Percentage')
  plt.xlabel('Recipe Size')
  plt.grid()
```

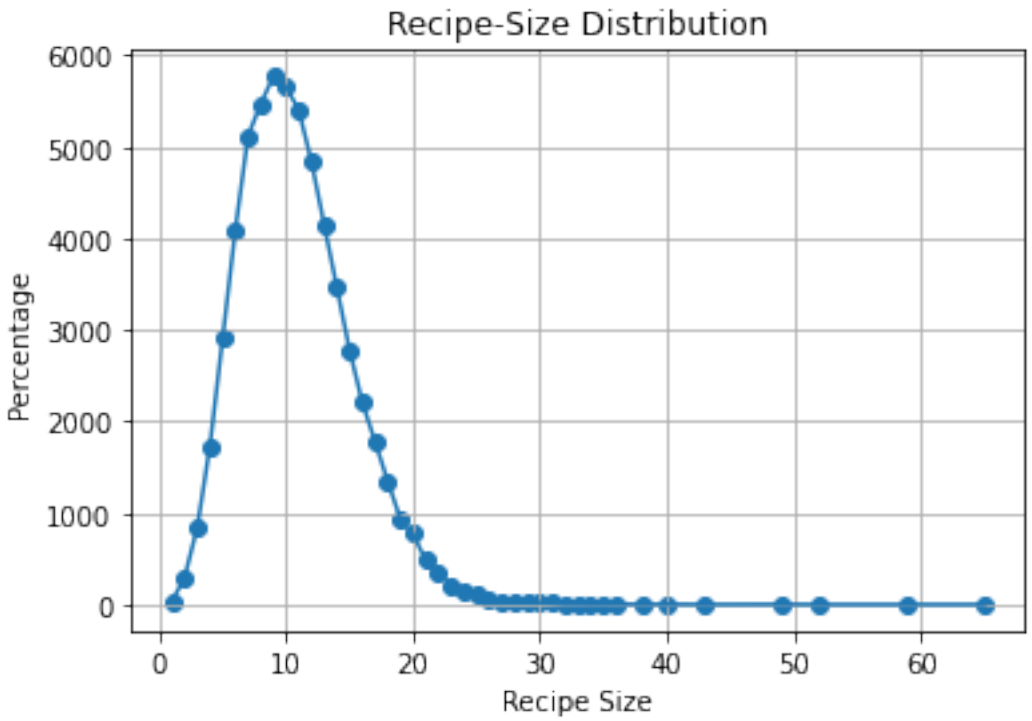
[ ]:
```
print("************Original Plot for recipe size distribution************")
print("_____________________________________________________________________________")
recipe_size_dist = RecDistributionCuisine(recipe_sizes)
generatePlot2(recipe_size_dist)
```

```
************Original Plot for recipe size distribution************
------------------------------------------------------------------------------
```



[ ]:
```
print("************Random Control R0 Plot for recipe size␣
 ↪distribution************")
print("_____________________________________________________________________________")
recipe_size_dist2 = RecDistributionCuisine(recipe_sizes_rand1)
generatePlot2(recipe_size_dist2)
```

```
************Random Control R0 Plot for recipe size distribution************
------------------------------------------------------------------------------
```



3 Random Control - R1

Function for generating random control R1 to preserve:
* the number of ingredients (i.e. 6714)
* the number of recipes (i.e. 39774)
* the recipe size distribution of the data
* the count of ingredients in the recipe data - ingredient frequency

[ ]:
```
# ingredient dictionary with ingredient name as keys and their count/ frequency␣
 ↪in the recipe data as columns
print(ing_dict)
```

[ ]:

```
# copying the ingredient dictionary in order to keep the original dictionary␣
 ↪intact
ing_dict_copy = dict(ing_dict)
print(ing_dict_copy)
print(len(ing_dict_copy.keys()))
```

[ ]:
```
import sys
```

[ ]:
```
# extending the recursion limit to 10^6 instead of normal limit of 1000 to␣
 ↪retrieve for 6K ingredients using recursive function
sys.setrecursionlimit(10**6)
```

[ ]:
```
# function to retrieve ingredients as per their original count in the kaggle␣
 ↪recipe data
# this is done to preserve the ingredient frequency while creating new␣
 ↪randomized data using random control R1
def get_ingredients(index,ing_dict,ingredients,ing_num):
  ingredient = ''
  index = index % ing_num
  if ingredients[index] in ing_dict.keys():
    # if(ing_dict[ingredients[index]] == 0):
    #   del ing_dict[ingredients[index]]
    #   ingredient = get_ingredients(index+1,ing_dict,ingredients,ing_num)
    # else:
    #   ingredient = ingredients[index]
    #   ing_dict[ingredients[index]] -= 1
    ingredient = ingredients[index]
    ing_dict[ingredients[index]] -= 1
    if ing_dict[ingredients[index]] == 0:
      #del ing_dict[ingredients[index]]
      np.delete(ingredients,index)
  else:
    ingredient = get_ingredients(index+1,ing_dict,ingredients,ing_num)
  return ingredient
```

[ ]:
```
# Random Control R1 function to preserve no. of ingredients, no. of recipes,␣
 ↪recipe size dist and ingredient frequency
def generating_random_recipes_r1(ingredients,recipe_sizes,ing_num,ing_dict):
  random_ingredients_data2 = []
  ing_index = 0
  for ing_size in recipe_sizes:
    new_recipe2 = []
    temp_index = ing_index % ing_num
    for num in range(ing_size):
      index = (temp_index + num)%ing_num
      #print(index)
      new_recipe2.append(get_ingredients(index,ing_dict,ingredients,ing_num))
      # if(ing_dict[ingredients[index]]>0):
```

```
      #   new_recipe2.append(ingredients[index])
      #   ing_dict[ingredients[index]] -= 1
      # else:
      #   continue
    random_ingredients_data2.append(new_recipe2)
    ing_index += ing_size
  new_recipe_data2 = pd.DataFrame()
  new_recipe_data2['ingredients'] = random_ingredients_data2
  return new_recipe_data2
```

Copying the shuffled array of ingredients to randomize the recipe data creation for random control R1

[ ]:
```
# shuffling the ingredients once again to maintain randomness while generating␣
 ↪recipes
random_ingredients2 = shuffle(random_ingredients, random_state=0)
print(random_ingredients2)
print(len(random_ingredients2))
```

[ ]:
```
# generating the new randomised data based on random control R1
random_data2 =␣
 ↪generating_random_recipes_r1(random_ingredients,recipe_sizes,ing_num,ing_dict_copy)
#random_data2
```

[ ]:
```
# checking whether the number of ingredients was maintained
random_ingredient_data2 = pd.DataFrame(random_data2['ingredients'].tolist())
random_ingredient_data2 = random_ingredient_data2.fillna(value=np.nan)

ing_list3 = []
for ing3 in Counter(np.array(random_ingredient_data2.values).reshape(-1)).
 ↪keys():
  ing_list3.append(ing3)
ing_list3.remove(np.nan)
ingredients3 = np.array(ing_list3)
ing_num3 = len(ingredients3)

# print(ingredients3)
# print(ing_num3)

print(random_ingredients2)
print(len(random_ingredients2))

# checking whether the recipe size distribution array was maintained
recipe_sizes_rand2 = []
#count = 0
for rec3 in random_data2['ingredients']:
  #count += 1
  recipe_sizes_rand2.append(len(rec3))
print(recipe_sizes_rand2)                    # sorted print
```

```
print(recipe_sizes)                              # sorted print
print(len(recipe_sizes_rand2))
#print(count)
```

Comparing both the recipe size distributions of original data and the data generated using random control 1

[ ]:
```
# function to compare whether the two arrays are the same
# made to compare the arrays for recipe size distribution
def compare(first, second, size):
    first.sort()
    second.sort()

    for i in range(size):
        if first[i] != second[i]:
            return False

    return True
```

[ ]:
```
# comparing the recipe size distribution arrays for the original data and for␣
 ↪the random control R1 generated data respectively
compare(recipe_sizes, recipe_sizes_rand2, 39774)
```

[ ]:
```
True
```

Comparing the Ingredient Frequencies of original data with the random control 1 data

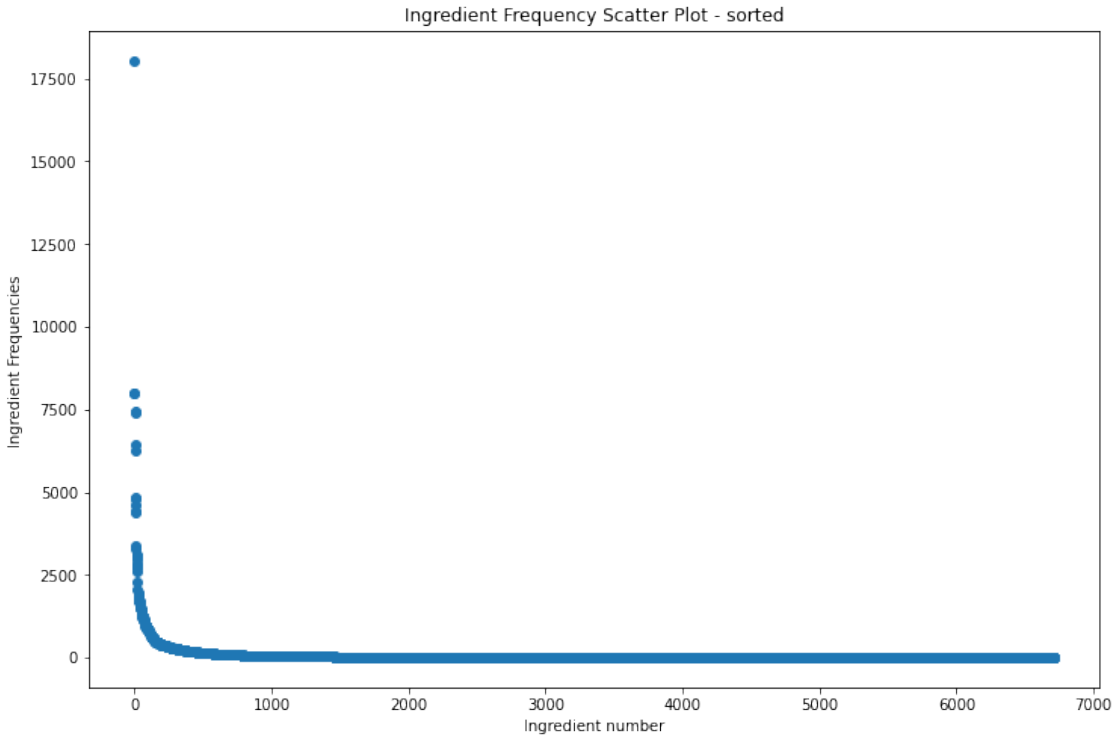
[ ]:
```
# creating sorted ingredient frequency dictionaries for original data and the␣
 ↪random control R1 generated data and comparing them together
import operator
sorted_ing_dict = sorted(ing_dict.items(), key=operator.
 ↪itemgetter(1),reverse=True)

rand1_ing_dict = Counter(np.array(random_ingredient_data2.values).reshape(-1))
del rand1_ing_dict[np.nan]
sorted_rand1_ing_dict = sorted(rand1_ing_dict.items(), key=operator.
 ↪itemgetter(1),reverse=True)

print("Sorted Ingredient Frequencies")
print(sorted_ing_dict)
print('\n')
print("Sorted Random Control 1 Ingredient Frequencies")
print(sorted_rand1_ing_dict)
```

Plotting the Ingredient Frequencies Plot

Ingredient Frequency Scatter Plot - sorted for Original Recipe Data

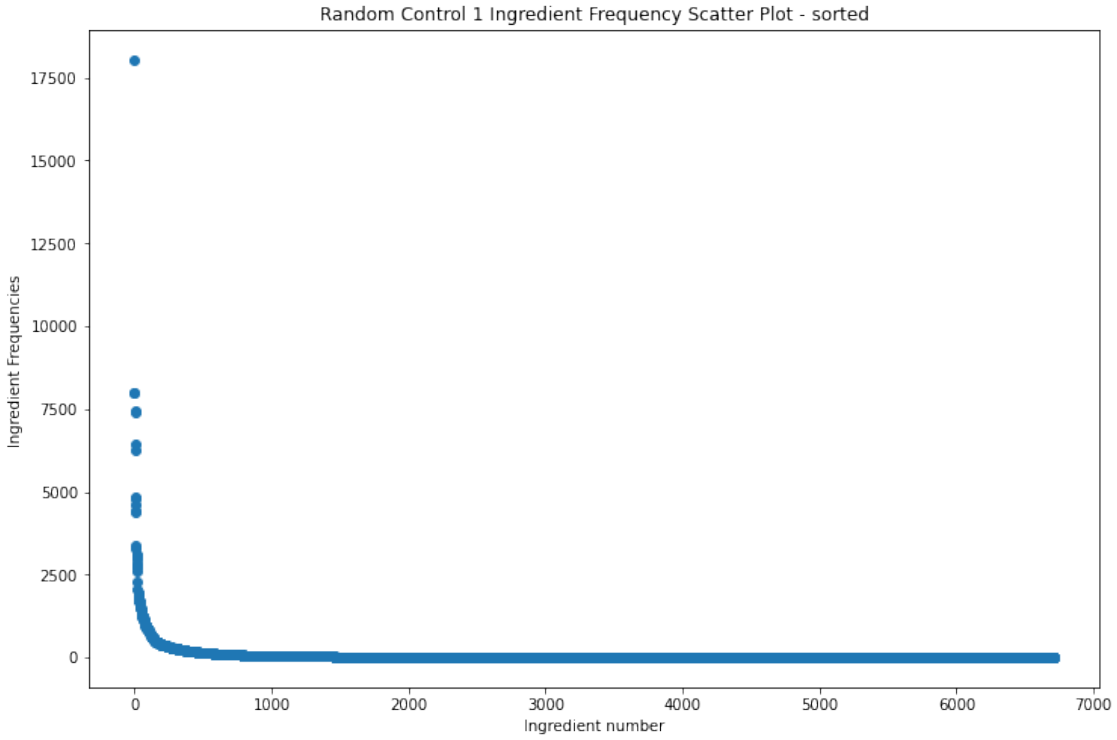
[ ]:
```
# plotting scatter plot for the original ingredient frequencies from the kaggle␣
 ↪recipes data
plt.figure(figsize=(12,8))
plt.scatter(range(6714),dict(sorted_ing_dict).values())
```

```
plt.title("Ingredient Frequency Scatter Plot - sorted")
plt.ylabel("Ingredient Frequencies ")
plt.xlabel("Ingredient number") #since ingredient names would take up space and␣
 ↪look messy while visualising
plt.show()
```



Ingredient Frequency Scatter Plot - sorted for Random Control R1 Recipe Data

[ ]:
```
# plotting scatter plot for the ingredient frequencies from the random control␣
 ↪R1 generated data
plt.figure(figsize=(12,8))
plt.scatter(range(6714),dict(sorted_rand1_ing_dict).values())
plt.title("Random Control 1 Ingredient Frequency Scatter Plot - sorted")
plt.ylabel("Ingredient Frequencies ")
plt.xlabel("Ingredient number") #since ingredient names would take up space and␣
 ↪look messy while visualising
plt.show()
```

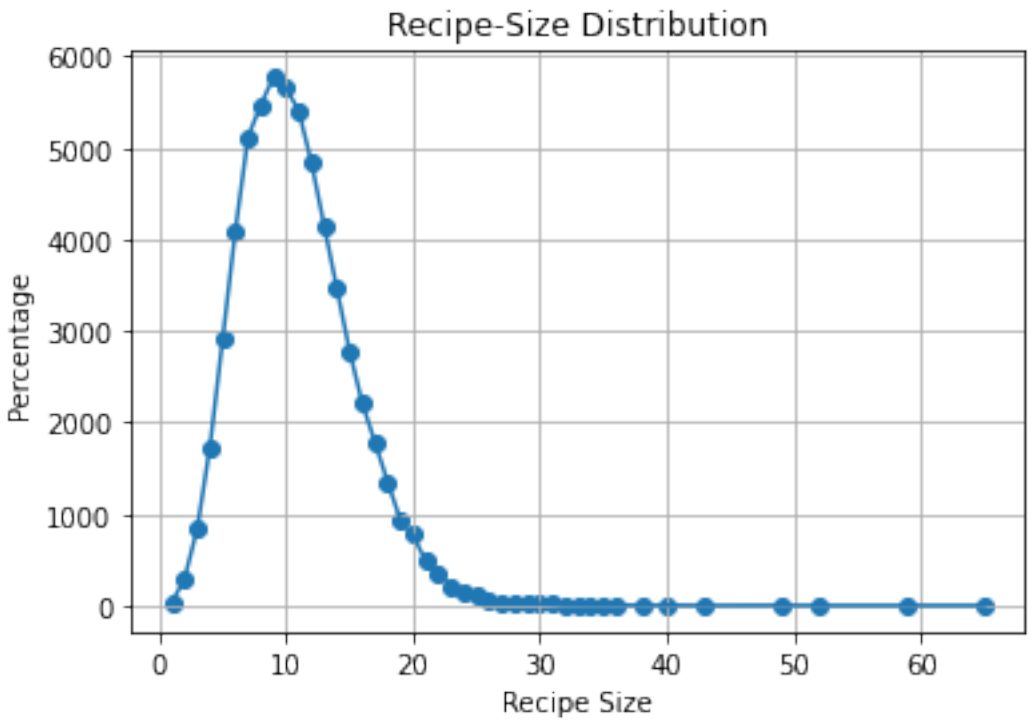


Since the two plots are on sorted ingredient frequency dictionaries hence the ingredients corresponding to index 1, 2, 3 etc are the same.

Recipe Size Distribution plots to show comparison after random control R1

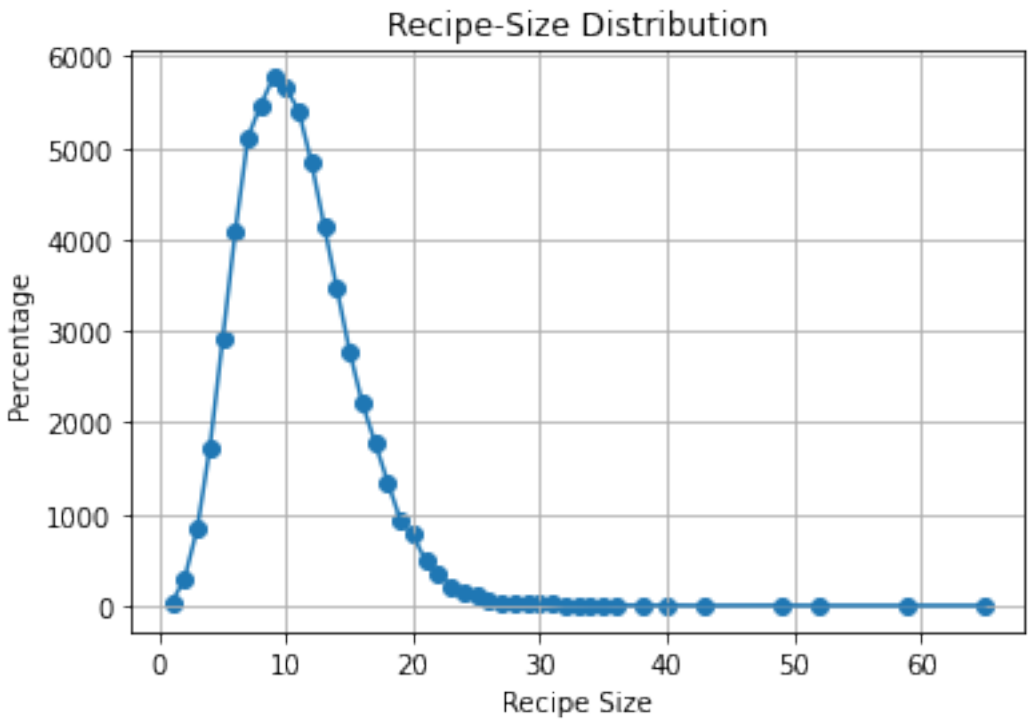
[ ]:
```
print("************Original Plot for recipe size distribution************")
print("_____________________________________________________________________________")
recipe_size_dist = RecDistributionCuisine(recipe_sizes)
generatePlot2(recipe_size_dist)
```

```
************Original Plot for recipe size distribution************
------------------------------------------------------------------------
```



[ ]:
```
print("************Random Control R1 Plot for recipe size␣
 ↪distribution************")
print("_____________________________________________________________________________")
recipe_size_dist3 = RecDistributionCuisine(recipe_sizes_rand2)
generatePlot2(recipe_size_dist3)
```

```
************Random Control R1 Plot for recipe size distribution************
------------------------------------------------------------------------------
```



4 Generating PDF Report of this Jupyter Notebook File

[17]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('CGAS_Assignment3_MT20075_RandomControls.ipynb')
```

```
--2021-11-02 04:50:58--  https://raw.githubusercontent.com/brpy/colab-
pdf/master/colab_pdf.py
Resolving raw.githubusercontent.com (raw.githubusercontent.com)...
185.199.111.133, 185.199.109.133, 185.199.108.133, ...
Connecting to raw.githubusercontent.com
(raw.githubusercontent.com)|185.199.111.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1864 (1.8K) [text/plain]
Saving to: colab_pdf.py

colab_pdf.py        100%[===================>]   1.82K  --.-KB/s    in 0s

2021-11-02 04:50:58 (21.7 MB/s) - colab_pdf.py saved [1864/1864]


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

```
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/CGAS_Assignment3_MT20075_RandomControls.ipynb to pdf
[NbConvertApp] Support files will be in
CGAS_Assignment3_MT20075_RandomControls_files/
[NbConvertApp] Making directory ./CGAS_Assignment3_MT20075_RandomControls_files
[NbConvertApp] Making directory ./CGAS_Assignment3_MT20075_RandomControls_files
[NbConvertApp] Making directory ./CGAS_Assignment3_MT20075_RandomControls_files
[NbConvertApp] Making directory ./CGAS_Assignment3_MT20075_RandomControls_files
[NbConvertApp] Making directory ./CGAS_Assignment3_MT20075_RandomControls_files
[NbConvertApp] Making directory ./CGAS_Assignment3_MT20075_RandomControls_files
[NbConvertApp] Making directory ./CGAS_Assignment3_MT20075_RandomControls_files
[NbConvertApp] Making directory ./CGAS_Assignment3_MT20075_RandomControls_files
[NbConvertApp] Writing 248922 bytes to ./notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: [u'xelatex', u'./notebook.tex',
'-quiet']
[NbConvertApp] CRITICAL | xelatex failed: [u'xelatex', u'./notebook.tex',
'-quiet']
This is XeTeX, Version 3.14159265-2.6-0.99998 (TeX Live 2017/Debian) (preloaded
format=xelatex)
 restricted \write18 enabled.
entering extended mode
(./notebook.tex
LaTeX2e <2017-04-15>
Babel <3.18> and hyphenation patterns for 3 language(s) loaded.
(/usr/share/texlive/texmf-dist/tex/latex/base/article.cls
Document Class: article 2014/09/29 v1.4h Standard LaTeX document class
(/usr/share/texlive/texmf-dist/tex/latex/base/size11.clo))
(/usr/share/texlive/texmf-dist/tex/latex/tcolorbox/tcolorbox.sty
(/usr/share/texlive/texmf-dist/tex/latex/pgf/basiclayer/pgf.sty
(/usr/share/texlive/texmf-dist/tex/latex/pgf/utilities/pgfrcs.sty
(/usr/share/texlive/texmf-dist/tex/generic/pgf/utilities/pgfutil-common.tex
(/usr/share/texlive/texmf-dist/tex/generic/pgf/utilities/pgfutil-common-lists.t
ex)) (/usr/share/texlive/texmf-dist/tex/generic/pgf/utilities/pgfutil-latex.def
(/usr/share/texlive/texmf-dist/tex/latex/ms/everyshi.sty))
(/usr/share/texlive/texmf-dist/tex/generic/pgf/utilities/pgfrcs.code.tex))
(/usr/share/texlive/texmf-dist/tex/latex/pgf/basiclayer/pgfcore.sty
(/usr/share/texlive/texmf-dist/tex/latex/graphics/graphicx.sty
(/usr/share/texlive/texmf-dist/tex/latex/graphics/keyval.sty)
(/usr/share/texlive/texmf-dist/tex/latex/graphics/graphics.sty
(/usr/share/texlive/texmf-dist/tex/latex/graphics/trig.sty)
(/usr/share/texlive/texmf-dist/tex/latex/graphics-cfg/graphics.cfg)
(/usr/share/texlive/texmf-dist/tex/latex/graphics-def/xetex.def)))
```

```
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/ot1ppl.fd)
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/omlzplm.fd)
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/omszplm.fd)
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/omxzplm.fd)
(/usr/share/texlive/texmf-dist/tex/latex/psnfss/ot1zplm.fd)
(/usr/share/texlive/texmf-dist/tex/latex/jknapltx/ursfs.fd)

LaTeX Warning: No \author given.

(/usr/share/texlive/texmf-dist/tex/generic/oberdiek/se-ascii-print.def)
(/usr/share/texmf/tex/latex/lm/t1lmtt.fd)
(/usr/share/texmf/tex/latex/lm/ts1lmtt.fd) [1] [2]
Underfull \hbox (badness 10000) in paragraph at lines 532--536

[3] [4]
Underfull \hbox (badness 10000) in paragraph at lines 680--681

[5]
Underfull \hbox (badness 10000) in paragraph at lines 698--699

[6]
Overfull \hbox (3.8924pt too wide) in paragraph at lines 749--749
[]|[]\T1/lmtt/b/n/10.95 print[]\T1/lmtt/m/n/10.95 ([]"[][]_____________________
_____________________________________________[][]"[])|

Underfull \hbox (badness 10000) in paragraph at lines 763--764


Overfull \hbox (3.8924pt too wide) in paragraph at lines 769--769
[]|[]\T1/lmtt/b/n/10.95 print[]\T1/lmtt/m/n/10.95 ([]"[][]_____________________
_____________________________________________[][]"[])|
[7]
Underfull \hbox (badness 10000) in paragraph at lines 783--784

[8] [9] [10] [11] [12] [13] [14] [15] [16] [17] [18] [19] [20] [21] [22]
[23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37]
[38] [39] [40] [41] [42] [43] [44] [45] [46] [47] [48] [49] [50]
Overfull \hbox (29.53339pt too wide) in paragraph at lines 2879--2879
[]\T1/lmtt/m/n/10.95 generating_random_recipes_r1(random_ingredients,recipe_siz
es,ing_num,ing_dict_copy)|
[51] [52]
Underfull \hbox (badness 10000) in paragraph at lines 2994--2995

[53]
Underfull \hbox (badness 10000) in paragraph at lines 3015--3016
```

```
vrxXEwXTe4vhzkH1XHb7rt914pdOxZ2O9uxGObjepTtk/aHxyho3jjt0jXbeFl9HI2rdy1zPHt9COTs
fs0n5kWx9XVS7jf9XI49D2LrSTLOw1U1Ttt10/f2c9PPeuyssaquteWzvlaUNe+14/nIZ+Oo2qvyZ/0
snmv1Oeb3TuaS2X5vbM01tojJ6HNx5J7jcyXHzLn1CRNr86F5NhM5boM+jOeztt7GdSp53r33ft1eHV
2yOZeDNZrIv41g8nwLXoFpE2OWJqjBQo9T8uR/fPH+TCIT8Eom4cgcOcIc+qVQJyXq5G7pQfWLr4nBZ
42ZtZniev4PErctE5+HPitytjAxH97r+dhfG95r/CeuYe9se8kyjvm8sZG4PpHXblvkhRHPCljxTF74
suvJF/MtngM/49+Rn+W5BTOluW7SBlTw2X/qPV+G5+NSZA64ZnNuwYSUyULz9sVOBYnFMIdmx+ybTBR
X+xYPinkxnDMBH8Ec+kVQJxXq5Gbf4iVl8Fljdm0+eb2Uy9Zzz/MvrT81crbwhZ+PpX6scX6nl77fY8
0JzjmTazON89lNk04NdXIL3wzP5AXet3wOs8fpPX3KvyM/yXML5kpjO6keJYe+wuYcz8eF4PPxZZtz3
2Uh6UlFdCRmX1sq2H4+vCxm3+Rh/Bq5tr/BH7oysXYXTDln/lzHHPq5UScV6uRm8jA5GqcvYPys8cLa
DD6MW/Lzjx6f1J+vP24tcrb09Z+PnRe7uX13/y31sjXs3cj82I3362oz1cK5lzv2d/N8z0vS741n8hL
vWz6Hb/Mu8dP+Hfnmzy14iVSXk/JQcmX0LDX1nrPxfFw49nz8/Oac3Gw7hgtvatzsB3d8QtvEc/MfRt
vE8dUnDcyTJuB7/4hjDsW7o07wRt7gWWP7A+rkH+SPFzHVwXPLl8bzcWH/o7Y8vvzL9Hdyey4890zRv
KtYD16Q4tt4g2dy3rfgkM/+dyTPLd8Nm3Pf1AvWtqc25wAAAAAAAAAAAADEsTkHAAAAAAAAAAAAXITN
OQAAAAAAAAAAOAibM4BAAAAAAAAAAAAF2FzDgAAAAAAAAAALgIm3MAAAAAAAAAAADARdicAwAAAAA
AAAAAC7C5hwAAAAAAAAAABwETbnAAAAAAAAAAAAgIuwOQcAAAAAAAAAABchM05AAAAAAAAAAAA4C
JszgEAAAAAAAAAACX+Pv3/wEpQaoLuVKWXwAAAABJRU5ErkJggg==' not found.

See the LaTeX manual or LaTeX Companion for explanation.
Type  H <return>  for immediate help.
 ...

l.3022 ...ACX+Pv3/wEpQaoLuVKWXwAAAABJRU5ErkJggg==}

?
! Emergency stop.
 ...

l.3022 ...ACX+Pv3/wEpQaoLuVKWXwAAAABJRU5ErkJggg==}

Output written on notebook.pdf (53 pages).
Transcript written on notebook.log.

[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 281059 bytes to /content/drive/My
Drive/CGAS_Assignment3_MT20075_RandomControls.pdf

<IPython.core.display.Javascript object>


<IPython.core.display.Javascript object>
```

[17]:
```
'File ready to be Downloaded and Saved to Drive'
```

[ ]:
```

```