```
!pip install -r requirements.txt
```

```
Requirement already satisfied: pandas==1.1.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: seaborn==0.11.1 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: matplotlib==3.2.2 in /usr/local/lib/python3.7/dist-packa
Collecting scikit-learn==0.22.2
  Using cached https://files.pythonhosted.org/packages/71/b0/471bfdb7741523dfbddd038cb5
Collecting plotly==4.5.0
  Using cached https://files.pythonhosted.org/packages/06/e1/88762ade699460dc3229c890f9
Requirement already satisfied: numpy==1.19.5 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: scipy>=1.0 in /usr/local/lib/python3.7/dist-packages (fr
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packa
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/l
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: retrying>=1.3.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from plot
Installing collected packages: scikit-learn, plotly
  Found existing installation: scikit-learn 0.22.2.post1
    Uninstalling scikit-learn-0.22.2.post1:
      Successfully uninstalled scikit-learn-0.22.2.post1
  Found existing installation: plotly 4.4.1
    Uninstalling plotly-4.4.1:
      Successfully uninstalled plotly-4.4.1
Successfully installed plotly-4.5.0 scikit-learn-0.22.2
```

Introduction

In This challenge my aim is to answer the questions one by one using the datasets provided. That is why you will find that I first worked with the Food Atlas dataset from cleaning and wrangling to regression and using the results for PCA. Then I merged the Medicare Beneficiaries Dataset with the cleaned dataset of Food Atlas on the FIPS column since I am interested in the counties locations rather than the whole states.

During the challenge, I was trying to use the simplest approach every time that is 100% clear in what it is doing with the data under the hood. Note that, the data cleaning process and the whole code is not the best for reproducing since I would have created specific functions for every cleaning and wrangling step possible for any dataset, but the time limit and the way the things can go wrong with such complex dataset can take more than 6 hours. Still, I will follow up with my full recommendations in the end.

Importing the Required Data and Packages

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.decomposition import PCA
import plotly.express as px
from urllib.request import urlopen
import json
import numpy as np
```

```
#it doesn't end with .xlsx
url_beneficiaries = 'https://drive.google.com/file/d/1Sio2Hq75qvUuXIGLIVnqA-bXTdjc6mzW/view?u
#that is why I used this to get the id
path_beneficiaries = 'https://drive.google.com/uc?export=download&id='+url_beneficiaries.spli
#now it will be openend with pandas smoothly.
beneficiaries_data = pd.read_excel(path_beneficiaries, header=1)
```

```
#same for .csv file
url_food_atlas = 'https://drive.google.com/file/d/1jfa4zJcwrCw-Q6PrRhXFh-0iUYLi1_1l/view?usp=
path_food_atlas = 'https://drive.google.com/uc?export=download&id='+url_food_atlas.split('/')
food_atlas_county = pd.read_csv(path_food_atlas)
```

Food Access Data Cleaning and Pre-Processing

```
#while exploring the dataset head(), I found the table wasn't set up properly
#for my needs, so I changed it into a pivot table using the location as index
food_atlas_county=food_atlas_county.pivot(index=['State', 'County', 'FIPS'],
                                          columns = ['Variable_Code'],
                                          values = 'Value')
```

```
#I then reset the location to be normal columns to be used later in the analysis
food_atlas_county.reset_index(inplace=True)
```

```
#the data had the similar column variables for different years, so I extracted
#the older variables using this code since the data columns was setup from
#older years to recent years. Note that I didn't care about what the date was
#for both datasets, but I cared about the recent years.
old_columns=[]
#since the index is being compared for column and its +1 neighbor, we have to
#stop at len(columns)-1 in order not to go over the end of the columns indix.
```

```
for indx in range(len(food_atlas_county.columns)-1):
  if food_atlas_county.columns.str.slice(stop= -2)[indx] == food_atlas_county.columns.str.sli
    old_columns.append(food_atlas_county.columns[indx])
```

```
#I dropped the old columns from the dataset
food_atlas_county = food_atlas_county.drop(columns=old_columns)
```

```
# the state total and FIPS was not important in the analysis,
#so I collected the indices of the state FIPS  using the fact
#that they are 2 or less digits each and dropped them in next cell
state_fips=[]
for i in range(food_atlas_county.shape[0]):
  if len(str(food_atlas_county['FIPS'][i]))<=2:
    state_fips.append(i)
```

```
#I droped the states FIPS and totals from the dataset
food_atlas_county = food_atlas_county.drop(index = state_fips)
```

```
#Checking if a column is all null values
null_columns=[]
for index, i in enumerate(food_atlas_county.isnull().all()):
  if i is True:
    null_columns.append(food_atlas_county.columns[index])
```

```
# dropping these null columns
food_atlas_county = food_atlas_county.drop(columns=null_columns)
```

```
#fill the rest na cells
food_atlas_county = food_atlas_county.fillna(0)
```

```
#change the FIPS type to apply extra zeros on the states from 1 to 9
#becasue they need to match the json file for the plotly map
food_atlas_county['FIPS'] = food_atlas_county['FIPS'].astype('int64')
```

```
#apply the formula
food_atlas_county['FIPS'] = food_atlas_county['FIPS'].apply(lambda x: '{0:0>5}'.format(x))
```

```
#dropping all the location columns from the training set and taking only the
#values to prepare it for fitting in a linear regression model
x = food_atlas_county.drop(columns=['State','County','FIPS']).values
#choosing FIPS as the target
y = food_atlas_county['FIPS'].values
#normalizing the training set
x = StandardScaler().fit_transform(x)
```

Food Access Data Analysis

```
#fitting the data to linear regression
reg = LinearRegression().fit(x, y)
#calculating the score
#as you can see the score is small, but we care about the highest correlations
#between the location and the dataset variables rather than the score
reg.score(x, y)
```

```
0.40034360344598197
```

```
#matching the coefficients with their respective column variable names
#to know which factors are important in determining the location
coef_list =[]
for i in range(len(x[0])):
  coef_list.append((abs(reg.coef_)[i],
                    food_atlas_county.drop(columns=['State','County','FIPS']).columns[i]))
```

```
#soting the list from most important to least important and taking
#the first 20 factors to analyze
imp_factors = sorted(coef_list,reverse=True )[:20]
print(imp_factors)
```

```
[(4719918094656441.0, 'VLFOODSEC_12_14'), (4604015665802854.0, 'FOODINSEC_12_14'), (412
```

```
#removing the columns that has older dates in a different format and percentage of actual dat
#we didn't take the race data here now, because we will use it as a correlation
#with PCA later for question 2
imp_food_atlas = food_atlas_county[[imp_factors[i][1] for i in range(len(imp_factors[:9]))]].
```

```
#normalizing the new dataset
x_new = imp_food_atlas.values
x_new = StandardScaler().fit_transform(x_new)
```

```
#getting principal component for all the important general factors
pca = PCA(n_components=1)
pca_component = pca.fit_transform(x_new)
print(pca.explained_variance_ratio_)
```

```
[0.57633003]
```

```
#creating a new dataframe for PCA resulting component and concatenating
#it with the FIPS to answer question 1 about the geographic location
pca_df = pd.DataFrame(data = pca_component, columns = ['principal_component_1'])
pca_df_fips = pd.concat([pca_df, food_atlas_county['FIPS'].reindex(pca_df.index)], axis=1)
```

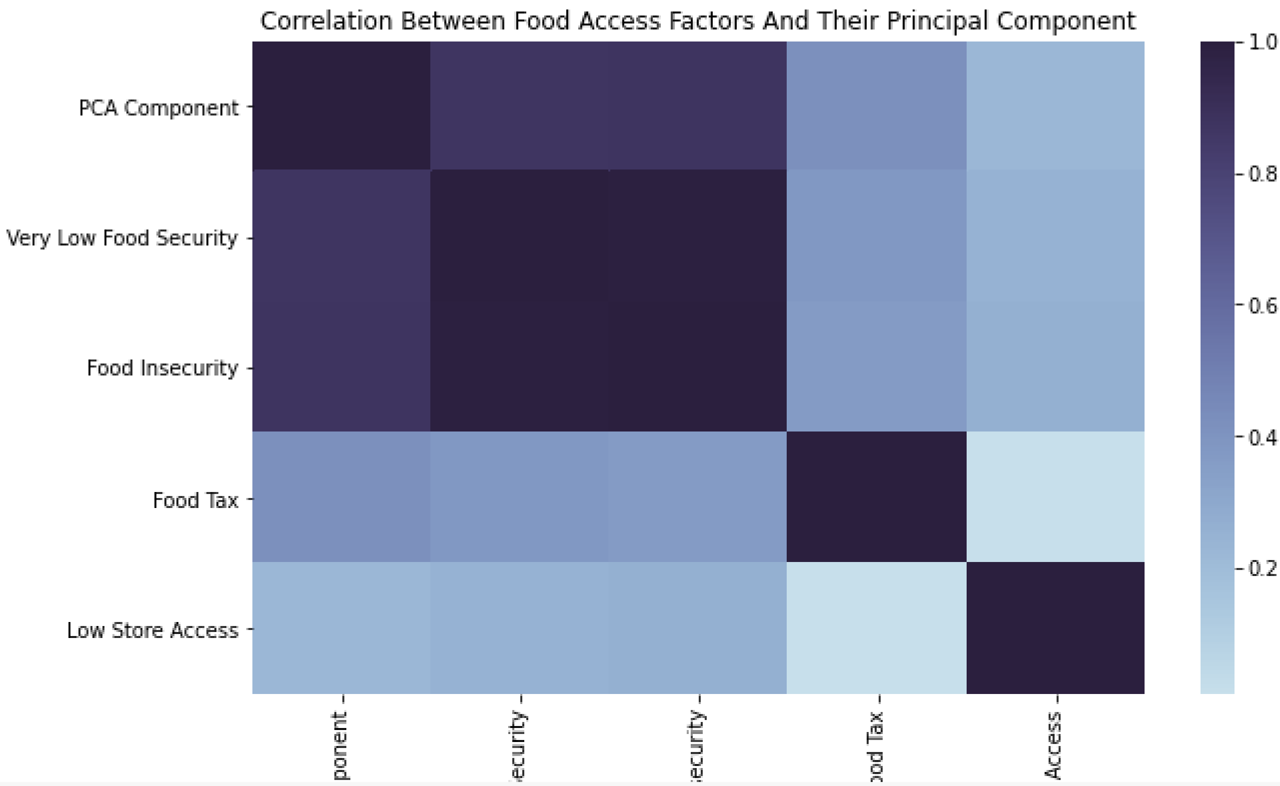
Question One

```
#I used plotly and this standard json data file to plot this map
with urlopen('https://raw.githubusercontent.com/plotly/datasets/master/geojson-counties-fips.
    counties = json.load(response)



fig = px.choropleth(pca_df_fips, geojson = counties, locations='FIPS',
                    color='principal_component_1', locationmode='geojson-id',
                    color_continuous_scale="Viridis", range_color=(0, 12),
                    scope="usa",
                    labels={'principal_component_1':'pricipal component value'})
fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})
fig.show()
```

```
#looking at the correlation between the important factors specified and their
#principal component
pca_imp_food_atlas = pd.concat([pca_df, imp_food_atlas.reindex(pca_df.index)], axis=1)
pca_imp_food_atlas = pca_imp_food_atlas.rename(columns={"principal_component_1": "PCA Compone
                                   "FOODINSEC_15_17":"Food Insecurity", "FOOD_TAX14":"Food Ta
```

```
# I am going to use the heatmap more than once, so I created a function for it
#with the same color map of Algorex Logo to fit the presentation style
def heatmap(data, title_str):
  plt.figure(figsize=(10,6))
  color=sns.color_palette("ch:s=.25,rot=-.25", as_cmap=True)
  sns.heatmap(data.corr(), cmap=color)
  plt.title(title_str)
  plt.show()
```

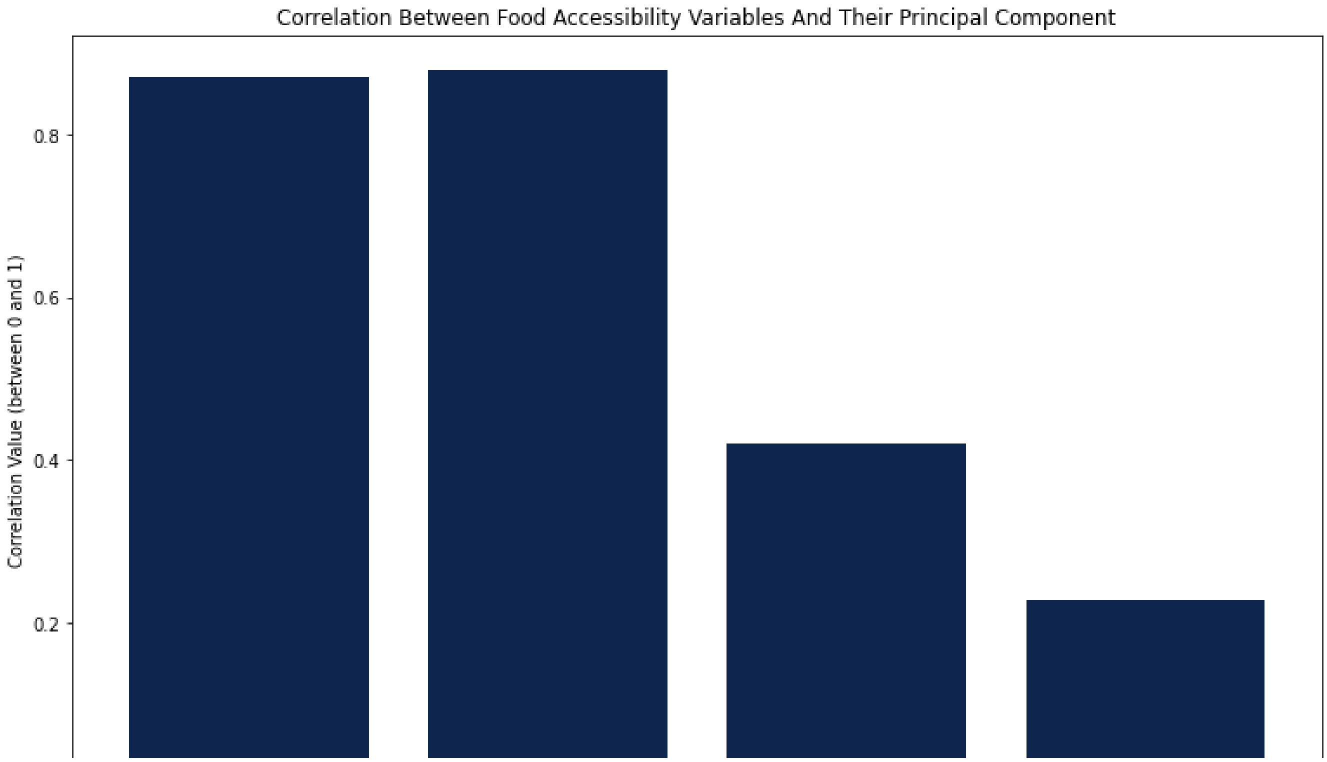
```
heatmap(pca_imp_food_atlas, 'Correlation Between Food Access Factors And Their Principal Comp
```




```
#since the array of the corr values is unhashable, I created a list to plot
#the bar plots
corr_vals_imp=[]
for i in np.delete(pca_imp_food_atlas.corr()[:1][:].values, 0):
  corr_vals_imp.append(i)
```

```
#creating another function for barplots with the same colors of Algorex Logo
def barplot(columns, values, title_str, xlabel):
  fig = plt.figure(figsize=(10,6))
  ax = fig.add_axes([0,0,1,1])
  x = columns
  y = values
  ax.bar(x, y, color=(0.055,0.145,0.306,1))
  plt.title(title_str)
  plt.xlabel(xlabel)
  plt.ylabel('Correlation Value (between 0 and 1)')
  plt.show()
```

```
barplot(pca_imp_food_atlas.corr()[:1][:].columns[1:], corr_vals_imp,
        'Correlation Between Food Accessibility Variables And Their Principal Component',
        'Food Accessibility Variables')
```

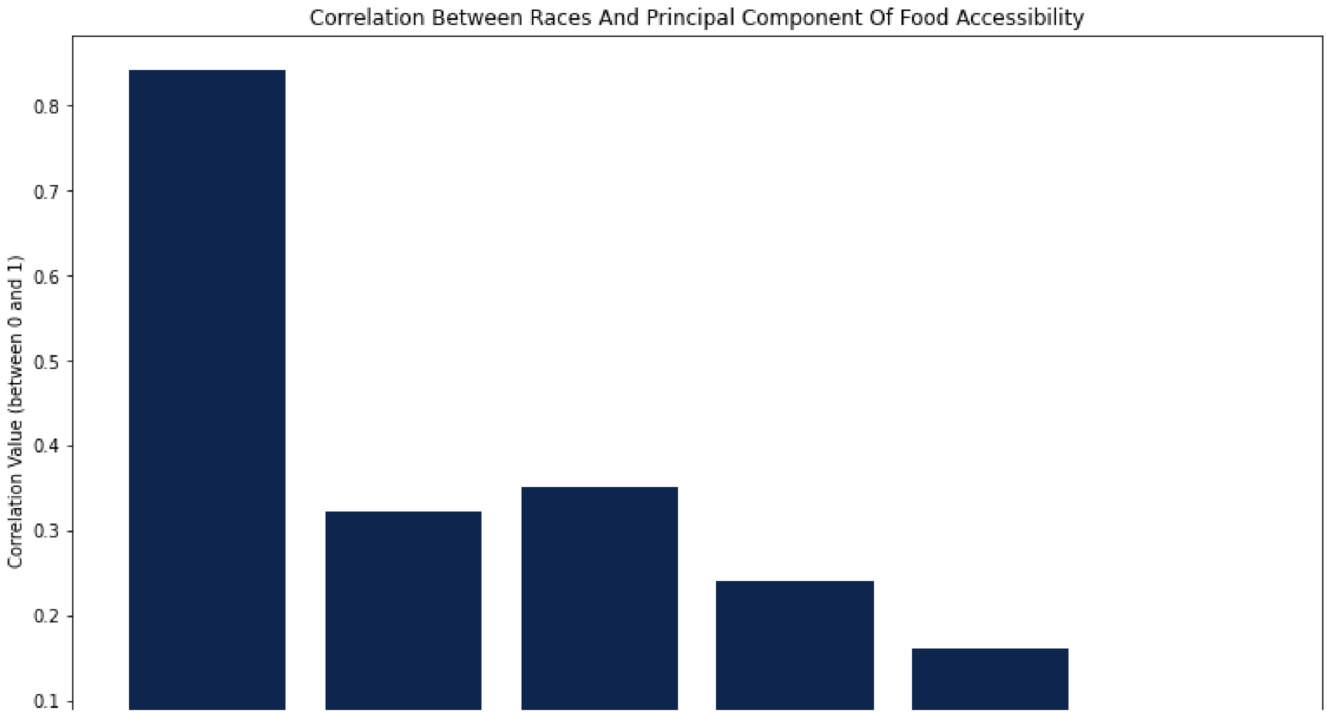


Question 2

```
#as we saw, the race with a differentiating factor between places and correlated
#highly with the general important factors principal component
#as we are going to see now
pca_race = pd.concat([pca_df, food_atlas_county[['PCT_NHWHITE10','PCT_NHBLACK10','PCT_HISP10'
                                                             'PCT_NHNA10','PCT_NHPI10']].reinde
pca_race = pca_race.rename(columns={"principal_component_1": "PCA Component", 'PCT_NHWHITE10'
                                                        'PCT_NHBLACK10': 'Black', 'PCT_HISP10
                                                        'PCT_NHNA10':'American Indian','PCT_N
```

```
corr_vals=[]
for i in np.delete(pca_race.corr()[:1][:].values, 0):
  corr_vals.append(i)
```

```
barplot(pca_race.corr()[:1][:].columns[1:], corr_vals,
        'Correlation Between Races And Principal Component Of Food Accessibility',
        'Race')
```



Medicare Data Cleaning and Pre-processing

```
#now that I need to measure the effect of the program on the costs,
#I want to merge the two datasets on FIPS, so I cleaned the FIPS to match the
#values in the Food Atlas data
beneficiaries_data=beneficiaries_data.rename(columns={'State and County FIPS Code':'FIPS'})
```

```
beneficiaries_data['FIPS'] = beneficiaries_data['FIPS'].fillna(0)
```

```
beneficiaries_data['FIPS'] = beneficiaries_data['FIPS'].astype('int64')
```

```
beneficiaries_data['FIPS'] = beneficiaries_data['FIPS'].apply(lambda x: '{0:0>5}'.format(x))
```

```
merged_df = pd.merge(food_atlas_county,beneficiaries_data, how="left", on='FIPS')
```

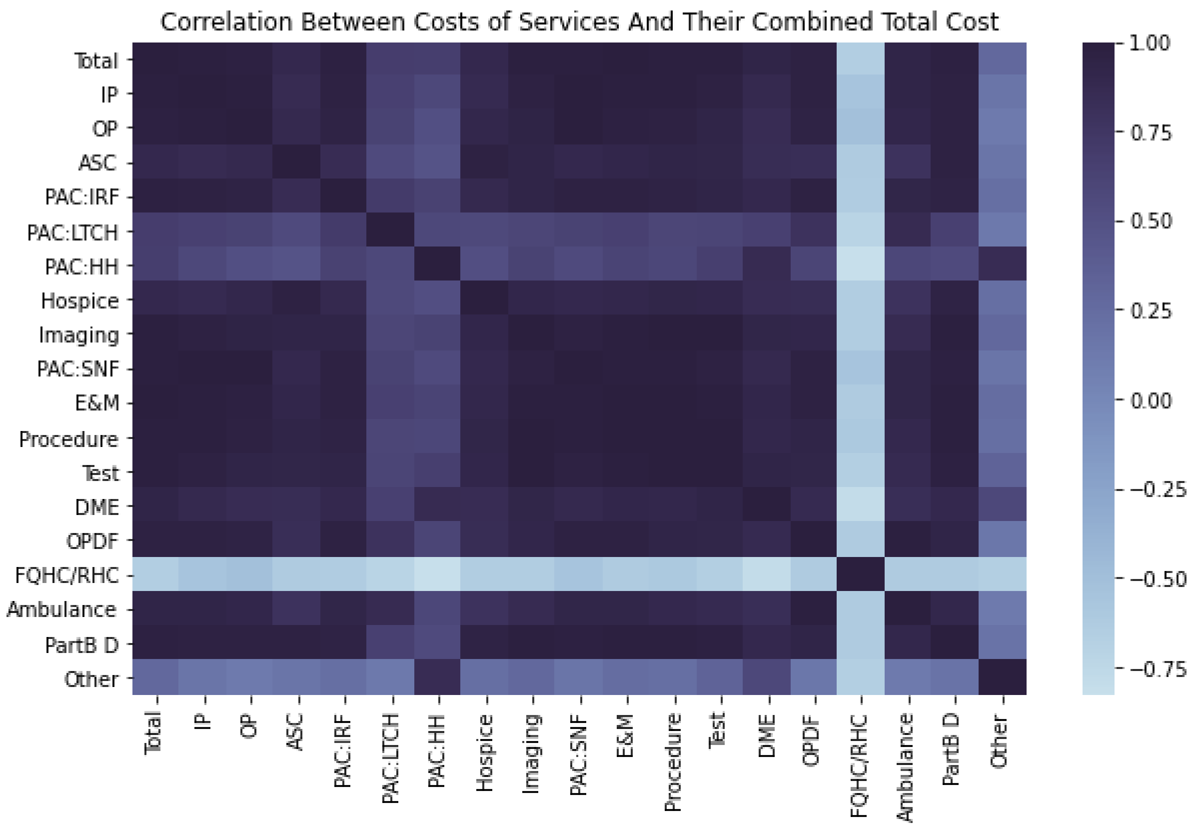
```
#now I cleaned a bit the data as a whole by removing the stars and filling NAs
#with zeros for preprocessing
merged_df = merged_df.fillna(0).replace('*', 0)
```

```
# every service cost is part of total costs, so I want to know the correlation
#between them so that we have ideas about the effect of the program on such costs
costs_df = merged_df[['Total Actual Costs','IP Actual Costs', 'OP Actual Costs', 'ASC Actual
               'PAC: LTCH Actual Costs', 'PAC: HH Actual Costs','Hospice Actual Costs','Imagi
               'E&M Actual Costs','Procedures Actual Costs','Tests Actual Costs', 'DME Actual
               'FQHC/RHC Actual Costs', 'Ambulance Actual Costs', 'Part B Drugs Actual Costs'
```

```
costs_df = costs_df.rename(columns={'Total Actual Costs': 'Total','IP Actual Costs': 'IP', '(
                'PAC: LTCH Actual Costs':'PAC:LTCH', 'PAC: HH Actual Costs':'PAC:HH','Hospice
                'E&M Actual Costs':'E&M','Procedures Actual Costs':'Procedure','Tests Actual (
                'FQHC/RHC Actual Costs':'FQHC/RHC', 'Ambulance Actual Costs':'Ambulance', 'Par
```

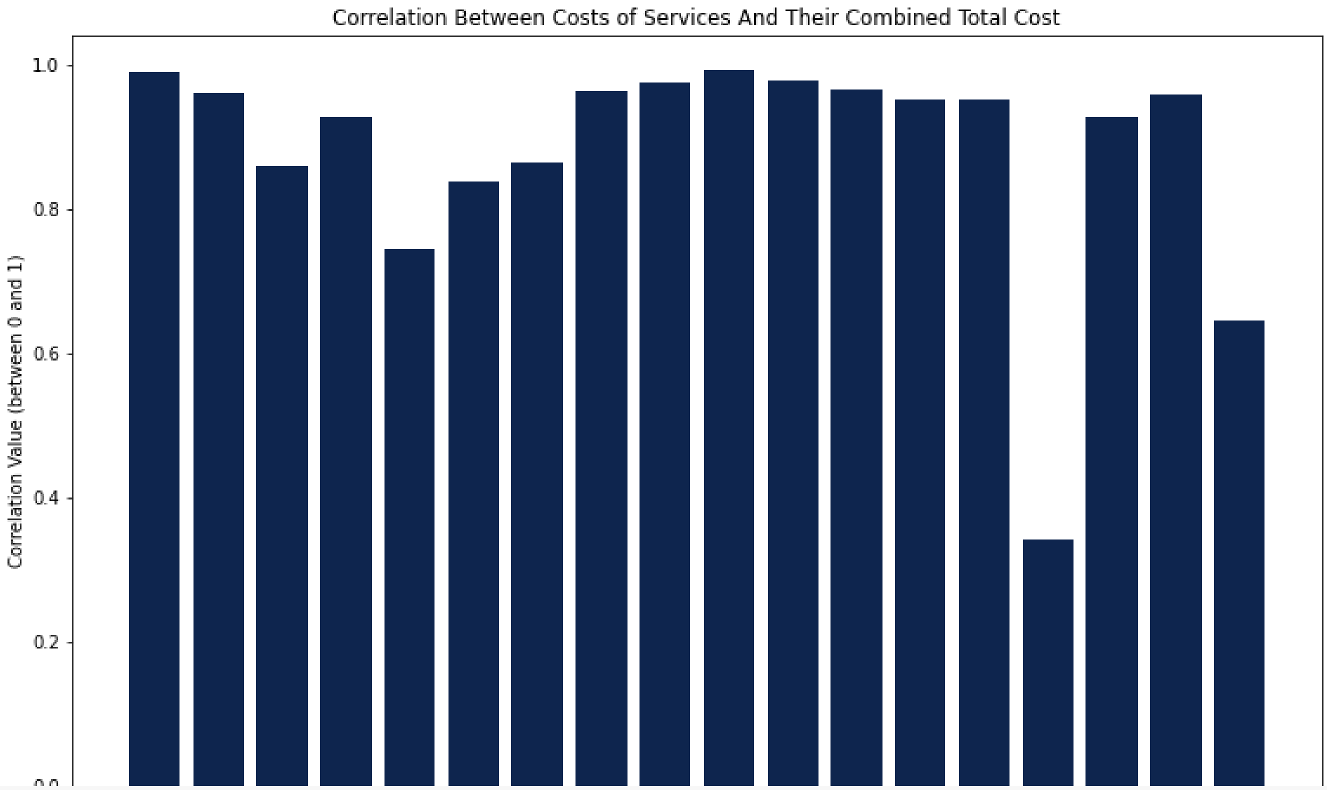
Question 3

```
heatmap(costs_df.corr(), 'Correlation Between Costs of Services And Their Combined Total Cost
```




```
cost_corr_vals=[]
for i in np.delete(costs_df.corr()[:1][:].values, 0):
  cost_corr_vals.append(i)
```

```
barplot([str(i) for i in range(1, len(cost_corr_vals)+1)], cost_corr_vals,
        'Correlation Between Costs of Services And Their Combined Total Cost',
        'Cost of Services')
```




```
regression_data = merged_df[['Total Actual Costs','VLFOODSEC_15_17', 'FOODINSEC_15_17', 'FOOD
regression_data = regression_data.rename(columns={'Total Actual Costs':'Total Costs','VLFOODS
                                                   'FOODINSEC_15_17':'Food Insecurity','FOOD_1
```

```
#preprocessing for regression again
x_cost = regression_data.drop(columns=['Total Costs']).values
y_cost = regression_data['Total Costs'].values
x_cost = StandardScaler().fit_transform(x_cost)
```

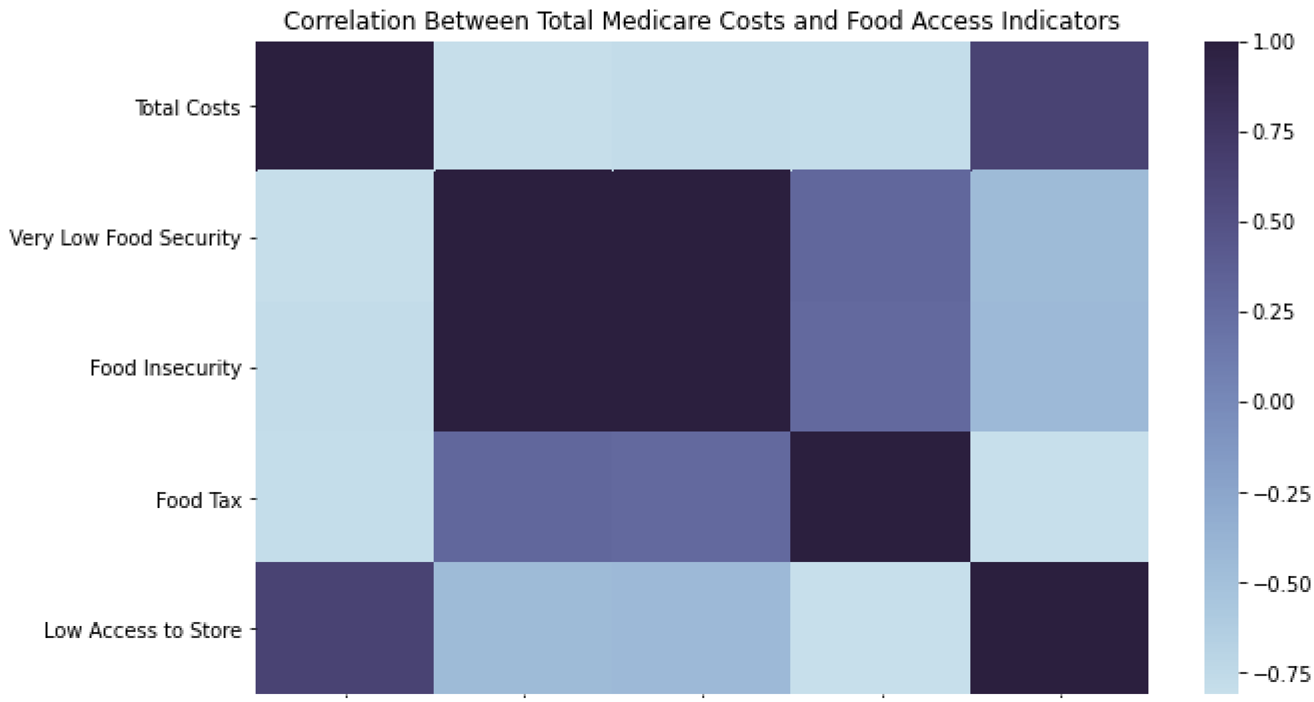
```
reg = LinearRegression().fit(x_cost, y_cost)
reg.score(x_cost, y_cost) #quite odd score
```

```
0.28874260082432845
```

```
reg.coef_ #only food access is positively correlated with the total costs
```

```
array([-2.36386482e+06, -3.20567058e+07, -3.72233693e+06,  1.38408210e+08])
```

```
heatmap(regression_data.corr(),'Correlation Between Total Medicare Costs and Food Access Indi
```



Data References:

CMS – State/County Medicare Utilization Summary - https://www.cms.gov/Research-Statistics-Data-and-Systems/Statistics-Trends-and-Reports/Medicare-Geographic-Variation/GV_PUF
FDA – Food Atlas - https://www.ers.usda.gov/data-products/food-environment-atlas/data-access-and-documentation-downloads/

Recommendations

1. Algorithmic Cleaning Process: As you can see, the data has lots of differences in cleaning, so we need a RL-based algorithm that decreases the score whenever an error happens and allows the addition of new rules by specifing what action it should take when such error happens again. #AIAlgorithms
2. Code Encapsulation: I didn't add a lot of functions, but for the graphs and the ML algorithms, functions would make the process a lot easier and systematic as well.
3. Databases: I worked here with the data files directly, but a better way would be to divide the data into columns where each column has the same index/id and cleaning each column without resetting the index, and when joining the data for any purpose, merge the columns and fill nas with zeros based on the length of the index/key column. Also, for the common columns as FIPS, they have to be one column with indices/key/ids of all the connected datasets in the database.

Notebook Colab Link:

https://colab.research.google.com/drive/1UOwL0Jn3F7mGvxfLG3x7GrcitFTDWER0?usp=sharing