Assignment 5

#Exercise 1.

1. Normalize the seven quantitative columns to a mean of 0 and standard deviation 1

```
# normalize the seven quantitative columns to a mean of 0 and standard deviat
for column in data[['Area',
    'Perimeter',
    'Major_Axis_Length',
    'Minor_Axis_Length',
    'Eccentricity',
    'Convex_Area',
    'Extent']]:
    data[column] = (data[column] - np.mean(data[column])) / np.std(data[colum
```

Check data["CLASS"]:

```
data.value_counts('Class')
Class
Osmancik    2180
Cammeo      1630
dtype: int64
```

1. Scatterplot for PC0 vs PC1 (color-coded by type of rice): In order to to do a scatterplot for PC0 vs PC1 (color-coded by type of rice), we first took the seven columns and saved into list my_cols

```
# save into list column_name
my_cols = data.columns.values.tolist()
```

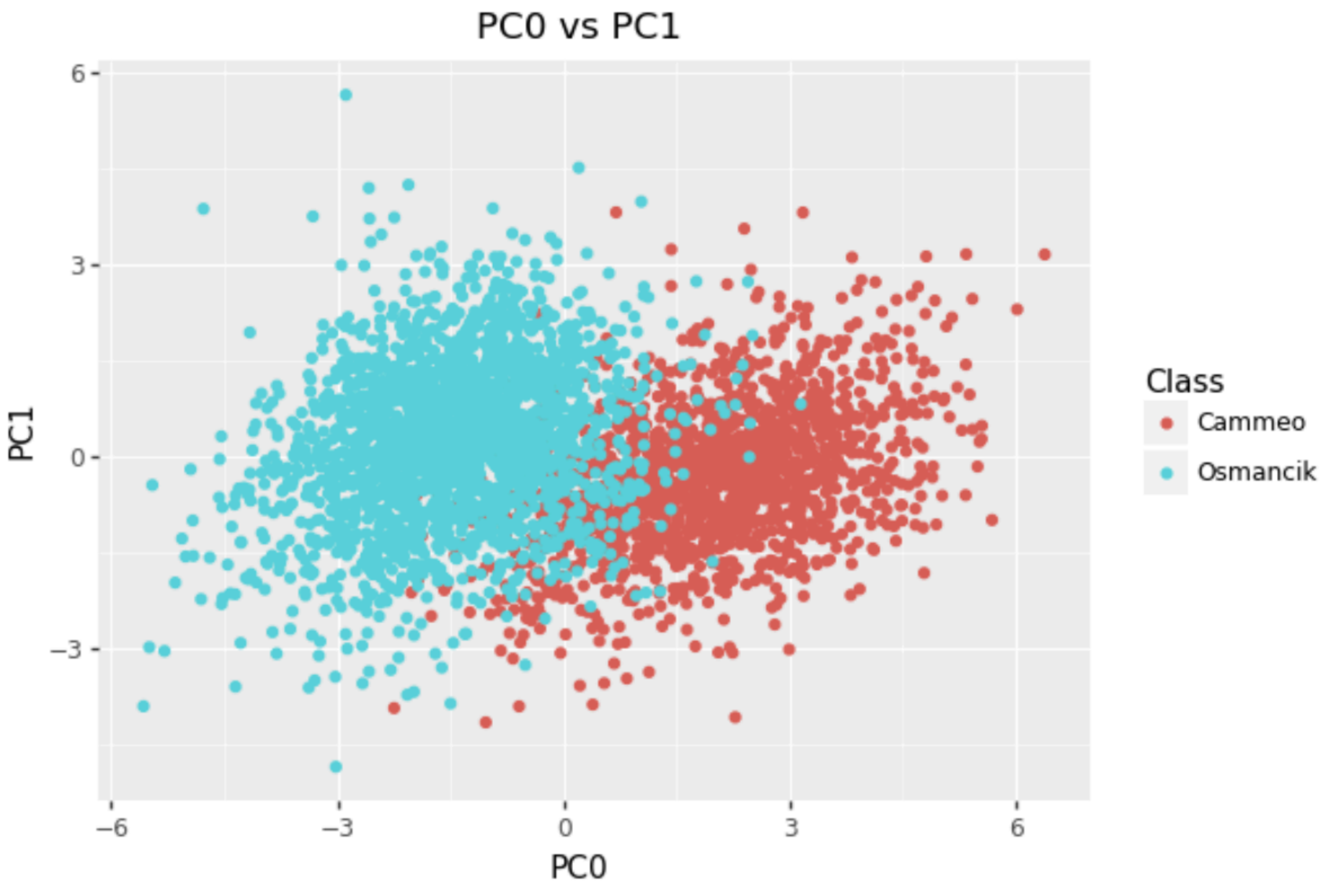
```
my_cols.pop()
```

Then the data was reduced into the two dimensions using the PCA from sklearn and the code provided for this hw:

```
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.h
pca = decomposition.PCA(n_components=2)
data_reduced = pca.fit_transform(data[my_cols])
pc0 = data_reduced[:, 0]
pc1 = data_reduced[:, 1]
```

And finally it was graphed the PCA (using a scatterplot graph):

```
#PC0 vs PC1
(p9.ggplot(data = data_pc0_pc1_list, mapping = p9.aes(x='PC0', y='PC1'))
+ p9.geom_point(p9.aes(x = 'PC0', y = 'PC1', color = 'Class'))
+ p9.labs(title = "PC0 vs PC1"))
```



The scatterplot shows that the PCA method of data classification is not entirely effective because some of the data points that are classified as "Cammeo" or "Osmancik" overlapped. However,, the PCA was able to classif/reduce data into a two dimension succesfuly, and showed the distance between two samples seens to be small if the were to belong into the same class.

2. Interpretation of what the confusion matrix results mean In order to interpretation of what the confusion matrix results mean we first needed to create a k-nearest neighbors and QuadTree: First, in order to set up data with points and create a quad tree, the first class Point was used. Each point holds the data x, y, and class. The width and height of the quadtree, in which (x, y) is the rectangle's center point and wid, hgt are the region's half-width and height, were saved using the class rectangle. To determine whether the rectangle is inside the circle,

whose center is at point.x, point.y, the function within() was created.

```
#https://jrtechs.net/data-science/implementing-a-quadtree-in-python
#https://scipython.com/blog/quadtrees-2-implementation-in-python/
#quadtree0.py code from class
class Point: #the points in the training set
    def __init__(self, x, y, payload=None):
        self.x, self.y = x, y
        self.payload = payload

    def distance_to(self, other):
        try:
            other_x, other_y = other.x, other.y
        except AttributeError:
            other_x, other_y = other
        return np.hypot(self.x - other_x, self.y - other_y)

class Rect: #Rectangular store the shape and other attribute of the quadtree
    #cx: center point x-axis
    #cy: center point y-axis
    #w: the width of the rectangular
    #h: the height of the rectangular
    def __init__(self, cx, cy, w, h):
        self.cx, self.cy = cx, cy
        self.w, self.h = w, h
        self.west_edge, self.east_edge = cx - w/2, cx + w/2
        self.north_edge, self.south_edge = cy - h/2, cy + h/2

    def contains(self, point):
        try:
            point_x, point_y = point.x, point.y
        except AttributeError:
            point_x, point_y = point

        return (point_x >= self.west_edge and
```

```
                point_x <  self.east_edge and
                point_y >= self.north_edge and
                point_y < self.south_edge)

    def intersects(self, other):
        return not (other.west_edge > self.east_edge or
                    other.east_edge < self.west_edge or
                    other.north_edge > self.south_edge or
                    other.south_edge < self.north_edge)


import math
class newQuadTree:
    def __init__(self, points, bounds, max_points=4):
        self.bounds = bounds
        self.max_points = max_points
        self._points=points
        self.x0=bounds[0]
        self.x1=bounds[1]
        self.y0=bounds[2]
        self.y1=bounds[3]
        if len(points)>max_points:
            self.divide()
            self._points=None
        else:
            self._points=points
            self.children=[]

    def divide(self): #divide the quad tree into 4 sub quadtree
        self.children=[]
        xmid=(self.x0+self.x1)/2
        ymid=(self.y0+self.y1)/2
        newBounds=[[self.x0, xmid,self.y0,ymid],[self.x0,xmid,ymid,self.y1],
                   [xmid,self.x1,self.y0,ymid],[xmid,self.x1,ymid,self.y1]]
        for newBound in newBounds:
            self.children.append(newQuadTree(filter_range(self._points, newBou
```

```
    def size(self):
        if not self.children:
            return len(self.points)
        else:
            total=0;
            for child in self.children:
                total+=child.size()
            return total

    def get_within_radius(self, center, radius, found_points):
        #check for overlap
        if  ((center[0]+radius<=self.x0 or center[0]-radius>=self.x1) or
                (center[1]+radius<=self.y0 or center[1]-radius>=self.y1)):
            return []
        # see if the values are within boundary
        if not self._points:
          for child in self.children:
            child.get_within_radius(center,radius,found_points)
        else:
          for point in self._points:
            if (math.sqrt(pow((point[0]-center[0]),2)+pow((point[1]-center[1]
              found_points.append(point)
        return found_points


def filter_range(points, bounds):
    x,y,cond=zip(*points)
    return [[x[i],y[i],cond[i]] for i in range(len(x)) if bounds[0]<=x[i]<=bou
```

Then the function KNN(), where the quad tree was saved:

```
#https://www.fatalerrors.org/a/k-nearest-neighbor-query-based-on-k-dimension-
```

```
#https://machinelearningmastery.com/tutorial-to-implement-k-nearest-neighbors-
#https://stackabuse.com/k-nearest-neighbors-algorithm-in-python-and-scikit-lea
class KNN2D():
    def __init__(self, k = 1) -> None:
        self.k = k

    def fit(self, data, conds):
        x,y=zip(*data)
        points=[[_x[0],_x[1],_cond] for _x,_cond in zip(data,conds)]
        # points=[[x[i],y[i],conds[i]] for i in range(len(conds))]
        # print("creating quad tree")
        self.qtree=newQuadTree(points,[min(x),max(x),min(y),max(y)])



    def predict(self, X):
        k = self.k
        conds = []
        # binary search for radius
        r_max = 3
        r_min = 0
        while True:
            found_points = []
            r = (r_max + r_min) / 2
            # print("getting radius", X, r, found_points)
            self.qtree.get_within_radius((X[0], X[1]), r, found_points)
            # print("got radius")
            if len(found_points) > k:
                r_max = r
            elif len(found_points) < k:
                r_min = r
            elif ((r_max-r_min)<0.01):
                break
            else:
                break
        for p in found_points:
```

```
            conds.append(p[2])
        return np.bincount(conds).argmax()
```

- k = 1

```
# k=1

for i in range(0, len(X), int(len(X)/3)):
    i = int(i)
    x_test = X.iloc[i: i+int(len(X)/3)]
    x_train = X.drop(x_test.index)
    y_test = Y[x_test.index]
    y_train = Y.drop(x_test.index)

    knn = KNN2D(k=1)
    pca=decomposition.PCA(n_components=2)
    pca_x_train=pca.fit_transform(x_train)
    pca_x_test=pca.transform(x_test)
    knn.fit(pca_x_train, y_train)
    pred=[]
    for x_val in pca_x_test:
      pred.append(knn.predict(x_val))
    pred = np.array(pred)
    y_test = np.array(y_test)
    print("Matrix of k = 1 nearest neighbors:")
    print(confusion_matrix(y_test, pred))
```

Matrix of k = 1 nearest neighbors: [[661 83] [84 442]] Matrix of k = 1 nearest neighbors: [[635 80] [69 486]] Matrix of k = 1 nearest neighbors: [[651 70] [61 488]]

- k = 5

```
 # k=5
for i in range(0, len(X), int(len(X)/3)):
    i = int(i)
    x_test = X.iloc[i: i+int(len(X)/3)]
    x_train = X.drop(x_test.index)
    y_test = Y[x_test.index]
    y_train = Y.drop(x_test.index)

    knn = KNN2D(k=5)
    pca=decomposition.PCA(n_components=2)
    pca_x_train=pca.fit_transform(x_train)
    pca_x_test=pca.transform(x_test)
    knn.fit(pca_x_train, y_train)
    pred=[]
    for x_val in pca_x_test:
      pred.append(knn.predict(x_val))
    pred = np.array(pred)
    y_test = np.array(y_test)
    print("Matrix of k = 5 nearest neighbors:")
    print(confusion_matrix(y_test, pred))
```

Matrix of k = 5 nearest neighbors: [[681 63] [63 463]] Matrix of k = 5 nearest neighbors: [[668 47] [60 495]] Matrix of k = 5 nearest neighbors: [[669 52] [48 501]]

In contrast, the overall false positive (Osmancik is incorrectly classified as Cammeo) and false negative (Cammeo is incorrectly classified as Osmancik) is lower in the results when k = 5 than in the results when k = 1. It is obvious that the overall true positive (Osmancik is correctly classified as Osmancik) and true negative (Cammeo is correctly classified as Cammeo) is higher in the results when k = 5 than k = 1. As a result, we can colnclude that k=5 is better.

#Exercise 2.

1. What's your data? The online data set I chose is the Life Expectancy(WHO), which consists of data from the period 2000 to 2015 for 193 countries. World Health Organization (WHO) keeps track of the health status as well as many other related factors for all countries. The datasets are made available to the public for health data analysis. The dataset related to life expectancy, and health factors for 193 countries during 2000-2015, has been collected from the same WHO data repository website and its corresponding economic data was collected from the United Nation website. Among all categories of health-related factors, only those critical factors were chosen which are more representative. It has been observed that there have been improvements in human mortality rates, especially in developing nations. Thus, I decided to look into further how some of the factors/variables play a role in developing and developed countries when it comes to health aspects and economic one. As the datasets were from WHO, we found no evident errors. Missing data were handled by filling null values with the means. The result indicated that most of the missing data were for population, Hepatitis B, and GDP. The data consists of 22 Columns and 2938 rows.

2. What analyses do you want to run and why are they interesting? First, I would like to start with a set of descriptive analyses to demonstrate the dataset and then move into showing the correlations and sitrubution of variables. And lastly, I would like to run a PCA test as well as maybe the K-nearest neighbor to demonstrate the variable predictivity of a life expectancy in developing vs developed counties.

3. Which ones will be interactive, where the user can provide a parameter? I am planning to have an interactive PCA and hopefully a worls map to show the parameters of life expectancy in each presented county as well as a set of interactive graphs based on each variable of the life expectancy dataset to show the parameters and how they play a role in each country presents. And finally I am hoping to implemnet a interactive describptive analysis dashboard using `pandas-profiling` library.

4. What graphs will you make?

- Linear plots for each specific variable
- Histograph to show distribution of life expectancy
- World map _ Correlation matrix However, it should be noted that most of the graphs that will be presented are going to be line graphs that indicate either growth or decline of life expectancy given a specific variable.

5. Describe how you want your website to work. For the website, I am thinking of implementing the `pandas-profiling` library and setting it as a dashboard page on the first main page (if not just a basic descriptive analysis page telling a bit more about the data). Then, there would be different tabs for each parameter and/or the interactive graph (they might be included in either a separate tab or within the parameters).

6. What do you see as your biggest challenge to completing this, and how do you expect to overcome this challenge?

One of the first challenges of this assignment was finding the dataset and then coming up with the right set of analyses to do for it. And the second challenge that I think is the biggest one is setting up a webpage for the interactive data visualizations as well as the descriptive analysis.

I am planning on overcoming those challenges by using the resources given to me through class lectures as well as resources found online. Furthermore, I belive that the resources shared, and the hands-on experience that we get in class regarding the flask and parametric test that we have done so far for homework assignments would be great resources to use to overcome the challenges.

#Exercise 3.

1. Perform any necessary data cleaning

> Original data citation: https://statecancerprofiles.cancer.gov/incidencerates/index.php?stateFIPS=00&areatype=state&cancer=001&race=00&sex=0&age=001&stage=999&year=0&type=incd&sortVariableName=rate&sortOrder=default&output=0#results

```
# https://moonbooks.org/Articles/How-to-remove-one-or-multiple-rows-in-a-panda
# https://stackoverflow.com/questions/13682044/remove-unwanted-parts-from-str

# remove rows that don't contain data
data = data.drop(data.index[53:74])

# remove the number and brackets in the Stats
data['State'] = data['State'].map(lambda x: x[:-3])
# rename column for numbers taken form States
data = data.rename(columns = {" FIPS": "FIPS"})
# edit the new column FIPS
#https://www.digitalocean.com/community/tutorials/update-rows-and-columns-pytl
#https://stackoverflow.com/questions/12604909/pandas-how-to-change-all-the-va
data['FIPS'] = (data['FIPS']/1000).map("{:,.0f}".format)
# save the clean data into a CSV file
data.to_csv('clean_data.csv', index=False)
```

```
clean_data = pd.read_csv("clean_data.csv")
clean_data
```

2. Make the page

```
import pandas as pd
import json

data = pd.read_csv("clean_data.csv")

age_adjusted_IR = data['Age-Adjusted Incidence Rate([rate note]) - cases per 1
state_name1 = data['State'].tolist()
state_name = []
for state in state_name1:
    state_name.append(state.lower())

from flask import Flask, render_template, request, url_for
app = Flask(__name__)

@app.route("/")
def homepage():
    return render_template("index.html")

@app.route("/state/<string:name>")
def get_info():
    dics = {}
    for item in state_name:
        dics[item] = age_adjusted_IR[state_name.index(item)]
    json_object = json.dumps(dics)
    return json_object

@app.route("/info", methods=["GET"])
def info():
```

Assignment 5

December 9, 2022

Assignment 5

#1

[ ]:
```
import pandas as pd
import plotnine as p9
from sklearn import decomposition
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import KFold
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score, cross_val_predict,␣
↪cross_validate
```

[ ]:
```
pip install --trusted-host pypi.org --trusted-host files.pythonhosted.org␣
↪tf-models-official
```

[ ]:
```
data = pd.read_excel('/Users/polina/Desktop/Rice_Cammeo_Osmancik.xlsx')
data
```

[ ]:
```
data.value_counts('Class')
```

[ ]:
```
# normalize the seven quantitative columns to a mean of 0 and standard␣
↪deviation 1.
for column in data[['Area',
    'Perimeter',
    'Major_Axis_Length',
    'Minor_Axis_Length',
    'Eccentricity',
    'Convex_Area',
    'Extent']]:
    data[column] = (data[column] - np.mean(data[column])) / np.std(data[column])
```

[ ]:
```
data
```

[ ]:
```
# save into list column_name
my_cols = data.columns.values.tolist()
my_cols.pop()
```

[ ]:
```
#https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.
 ↪html
pca = decomposition.PCA(n_components=2)
data_reduced = pca.fit_transform(data[my_cols])
pc0 = data_reduced[:, 0]
pc1 = data_reduced[:, 1]
```

[ ]:
```
pc0_list = pc0.tolist()
pc1_list = pc1.tolist()
data_pc0_pc1_list = pd.DataFrame({'PC0': pc0_list,'PC1': pc1_list,'Class':␣
 ↪data['Class']})
```

[ ]:
```
#PC0 vs PC1
(p9.ggplot(data = data_pc0_pc1_list, mapping = p9.aes(x='PC0', y='PC1'))
+ p9.geom_point(p9.aes(x = 'PC0', y = 'PC1', color = 'Class'))
+ p9.labs(title = "PC0 vs PC1"))
```

[ ]:
```
#https://jrtechs.net/data-science/implementing-a-quadtree-in-python
# code: https://scipython.com/blog/quadtrees-2-implementation-in-python/
#quadtree0.py code from class
class Point: #the points in the training set
    def __init__(self, x, y, payload=None):
        self.x, self.y = x, y
        self.payload = payload

    def distance_to(self, other):
        try:
            other_x, other_y = other.x, other.y
        except AttributeError:
            other_x, other_y = other
        return np.hypot(self.x - other_x, self.y - other_y)

class Rect: #Rectangular store the shape and other attribute of the quadtree
    #cx: center point x-axis
    #cy: center point y-axis
    #w: the width of the rectangular
    #h: the height of the rectangular
    def __init__(self, cx, cy, w, h):
        self.cx, self.cy = cx, cy
        self.w, self.h = w, h
        self.west_edge, self.east_edge = cx - w/2, cx + w/2
        self.north_edge, self.south_edge = cy - h/2, cy + h/2

    def contains(self, point):
        try:
            point_x, point_y = point.x, point.y
        except AttributeError:
```

```
            point_x, point_y = point

        return (point_x >= self.west_edge and
                point_x <  self.east_edge and
                point_y >= self.north_edge and
                point_y < self.south_edge)

    def intersects(self, other):
        return not (other.west_edge > self.east_edge or
                    other.east_edge < self.west_edge or
                    other.north_edge > self.south_edge or
                    other.south_edge < self.north_edge)
```

[ ]:
```
import math
class newQuadTree:
    def __init__(self, points, bounds, max_points=4):
        self.bounds = bounds
        self.max_points = max_points
        self._points=points
        self.x0=bounds[0]
        self.x1=bounds[1]
        self.y0=bounds[2]
        self.y1=bounds[3]
        if len(points)>max_points:
            self.divide()
            self._points=None
        else:
            self._points=points
            self.children=[]

    def divide(self): #divide the quad tree into 4 sub quadtree
        self.children=[]
        xmid=(self.x0+self.x1)/2
        ymid=(self.y0+self.y1)/2
        newBounds=[[self.x0, xmid,self.y0,ymid],[self.x0,xmid,ymid,self.y1],
                   [xmid,self.x1,self.y0,ymid],[xmid,self.x1,ymid,self.y1]]
        for newBound in newBounds:
            self.children.append(newQuadTree(filter_range(self._points,␣
↪newBound),newBound,self.max_points))


    def size(self):
        if not self.children:
            return len(self.points)
        else:
            total=0;
            for child in self.children:
```

```
                total+=child.size()
            return total

    def get_within_radius(self, center, radius, found_points):
        #check for overlap
        if  ((center[0]+radius<=self.x0 or center[0]-radius>=self.x1) or
                (center[1]+radius<=self.y0 or center[1]-radius>=self.y1)):
            return []
        # see if the values are within boundary
        if not self._points:
          for child in self.children:
            child.get_within_radius(center,radius,found_points)
        else:
          for point in self._points:
            if (math.
↪sqrt(pow((point[0]-center[0]),2)+pow((point[1]-center[1]),2))<=radius):
              found_points.append(point)
        return found_points


def filter_range(points, bounds):
    x,y,cond=zip(*points)
    return [[x[i],y[i],cond[i]] for i in range(len(x)) if␣
↪bounds[0]<=x[i]<=bounds[1] and bounds[2]<=y[i]<=bounds[3]]
```

[ ]:
```
#https://www.fatalerrors.org/a/
↪k-nearest-neighbor-query-based-on-k-dimension-tree-of-knn-algorithm.html
#https://machinelearningmastery.com/
↪tutorial-to-implement-k-nearest-neighbors-in-python-from-scratch/
#https://stackabuse.com/
↪k-nearest-neighbors-algorithm-in-python-and-scikit-learn/
class KNN2D():
    def __init__(self, k = 1) -> None:
        self.k = k

    def fit(self, data, conds):
        x,y=zip(*data)
        points=[[_x[0],_x[1],_cond] for _x,_cond in zip(data,conds)]
        # points=[[x[i],y[i],conds[i]] for i in range(len(conds))]
        # print("creating quad tree")
        self.qtree=newQuadTree(points,[min(x),max(x),min(y),max(y)])


    def predict(self, X):
        k = self.k
        conds = []
```

```
        # binary search for radius
        r_max = 3
        r_min = 0
        while True:
            found_points = []
            r = (r_max + r_min) / 2
            # print("getting radius", X, r, found_points)
            self.qtree.get_within_radius((X[0], X[1]), r, found_points)
            # print("got radius")
            if len(found_points) > k:
                r_max = r
            elif len(found_points) < k:
                r_min = r
            elif ((r_max-r_min)<0.01):
                break
            else:
                break
        for p in found_points:
            conds.append(p[2])
        return np.bincount(conds).argmax()
```

[ ]:
```
_data = data.sample(frac=1)
X = _data[[
    'Area',
    'Perimeter',
    'Major_Axis_Length',
    'Minor_Axis_Length',
    'Eccentricity',
    'Convex_Area',
    'Extent'
    ]]
 # reference: https://www.askpython.com/python/examples/k-fold-cross-validation
Y = _data.Class.map({'Osmancik': 0, 'Cammeo': 1})
```

[ ]:
```
# k=1

for i in range(0, len(X), int(len(X)/3)):
    i = int(i)
    x_test = X.iloc[i: i+int(len(X)/3)]
    x_train = X.drop(x_test.index)
    y_test = Y[x_test.index]
    y_train = Y.drop(x_test.index)

    knn = KNN2D(k=1)
    pca=decomposition.PCA(n_components=2)
```

```
    pca_x_train=pca.fit_transform(x_train)
    pca_x_test=pca.transform(x_test)
    knn.fit(pca_x_train, y_train)
    pred=[]
    for x_val in pca_x_test:
      pred.append(knn.predict(x_val))
    pred = np.array(pred)
    y_test = np.array(y_test)
    print("Matrix of k = 1 nearest neighbors:")
    print(confusion_matrix(y_test, pred))
```

[ ]:
```
# k=5
for i in range(0, len(X), int(len(X)/3)):
    i = int(i)
    x_test = X.iloc[i: i+int(len(X)/3)]
    x_train = X.drop(x_test.index)
    y_test = Y[x_test.index]
    y_train = Y.drop(x_test.index)

    knn = KNN2D(k=5)
    pca=decomposition.PCA(n_components=2)
    pca_x_train=pca.fit_transform(x_train)
    pca_x_test=pca.transform(x_test)
    knn.fit(pca_x_train, y_train)
    pred=[]
    for x_val in pca_x_test:
      pred.append(knn.predict(x_val))
    pred = np.array(pred)
    y_test = np.array(y_test)
    print("Matrix of k = 5 nearest neighbors:")
    print(confusion_matrix(y_test, pred))
```

[ ]:
```
# Test
test_x=[]
test_y=[]
test_cond=[]
test_points=[]
for i in range(10):
  for j in range(5):
    test_x.append(i)
    test_y.append(j)
    test_cond.append(1)
    test_points.append((i,j))

  for j in range(5,10):
    test_x.append(i)
    test_y.append(j)
```

```
    test_cond.append(2)
    test_points.append((i,j))

knn=KNN2D()
knn.fit(test_points, test_cond)
knn.predict([1.5,1.5])
knn.predict([9.2,9.3])
```

#3

[ ]:
```
%conda install -c anaconda flask -y
```

[1]:
```
import pandas as pd
import numpy as np
import csv
```

[ ]:
```
data = pd.read_csv("/Users/polina/Desktop/incd.csv", skiprows = 8)
data
```

Original data citation: https://statecancerprofiles.cancer.gov/incidencerates/index.php?stateFIPS=00&areatype=

[ ]:
```
# https://moonbooks.org/Articles/
↪How-to-remove-one-or-multiple-rows-in-a-pandas-DataFrame-in-python-/
# https://stackoverflow.com/questions/13682044/
↪remove-unwanted-parts-from-strings-in-a-column/22238380

# remove rows that don't contain data
data = data.drop(data.index[53:74])

# remove the number and brackets in the Stats
data['State'] = data['State'].map(lambda x: x[:-3])
```

[ ]:
```
# rename column for numbers taken form States
data = data.rename(columns = {" FIPS": "FIPS"})
```

[ ]:
```
# edit the new column FIPS
#https://www.digitalocean.com/community/tutorials/
↪update-rows-and-columns-python-pandas
#https://stackoverflow.com/questions/12604909/
↪pandas-how-to-change-all-the-values-of-a-column
data['FIPS'] = (data['FIPS']/1000).map("{:,.0f}".format)
```

[ ]:
```
data
```

[ ]:
```
# save the clean data into a CSV file
data.to_csv('clean_data.csv', index=False)
```

[7]:
```
clean_data = pd.read_csv("clean_data.csv")
clean_data
```

[7]:
```
                   State  FIPS  \
0         US (SEER+NPCR)     0
1               Kentucky    21
2                   Iowa    19
3             New Jersey    34
4          West Virginia    54
5               New York    36
6              Louisiana    22
7               Arkansas     5
8          New Hampshire    33
9           Pennsylvania    42
10                 Maine    23
11          Rhode Island    44
12           Mississippi    28
13              Delaware    10
14             Minnesota    27
15                  Ohio    39
16           Connecticut     9
17             Wisconsin    55
18        North Carolina    37
19              Nebraska    31
20               Georgia    13
21             Tennessee    47
22               Montana    30
23              Illinois    17
24               Florida    12
25                Kansas    20
26               Vermont    50
27               Indiana    18
28         Massachusetts    25
29          North Dakota    38
30              Maryland    24
31              Missouri    29
32          South Dakota    46
33               Alabama     1
34              Oklahoma    40
35                 Idaho    16
36              Michigan    26
37        South Carolina    45
38            Washington    53
39                Oregon    41
40                Alaska     3
41  District of Columbia    11
42                Hawaii    15
```

```
43              Texas    48
44           Virginia    51
45               Utah    49
46            Wyoming    56
47         California     6
48           Colorado     8
49            Arizona     4
50         New Mexico    35
51        Puerto Rico    72
52             Nevada    32

   Age-Adjusted Incidence Rate([rate note]) - cases per 100,000  \
0                                               449.4
1                                                 516
2                                               490.7
3                                               488.9
4                                               487.4
5                                               484.8
6                                               484.3
7                                               483.6
8                                               482.9
9                                               476.8
10                                              476.7
11                                              476.2
12                                                476
13                                              474.7
14                                              471.5
15                                              471.5
16                                              471.4
17                                              470.8
18                                              469.9
19                                              469.7
20                                              468.6
21                                              466.5
22                                              466.3
23                                              465.2
24                                              460.5
25                                              459.4
26                                                457
27                                              456.8
28                                              454.8
29                                              454.4
30                                              454.1
31                                              453.2
32                                              452.3
33                                              451.7
34                                              450.8
```

35	448.5
36	446.7
37	443.8
38	441.3
39	428.4
40	417
41	416.9
42	416.8
43	415.3
44	409.4
45	407.2
46	405.7
47	402.4
48	396.4
49	382.4
50	374
51	368.2
52	data not available

	Lower 95% Confidence Interval	Upper 95% Confidence Interval	\
0	449.1	449.7	
1	513.2	518.8	
2	487.5	494	
3	487	490.8	
4	483.3	491.4	
5	483.6	486.1	
6	481.7	487	
7	480.4	486.9	
8	478.2	487.7	
9	475.3	478.3	
10	472.2	481.3	
11	470.8	481.6	
12	472.7	479.3	
13	469.1	480.3	
14	469.2	474	
15	469.8	473.1	
16	468.5	474.3	
17	468.5	473.1	
18	468.2	471.7	
19	465.6	473.8	
20	466.8	470.4	
21	464.4	468.7	
22	461	471.7	
23	463.6	466.8	
24	459.4	461.6	
25	456.1	462.7	
26	450.3	463.8	

27	454.6	458.9
28	452.7	456.8
29	447.8	461.1
30	451.9	456.4
31	451	455.4
32	446.4	458.3
33	449.3	454.2
34	448	453.6
35	444.2	452.8
36	445	448.4
37	441.4	446.2
38	439.2	443.3
39	425.8	431
40	410	424.1
41	410	424
42	412.5	421.2
43	414.3	416.4
44	407.6	411.2
45	403.8	410.6
46	398.9	412.7
47	401.6	403.3
48	394.1	398.6
49	380.6	384.3
50	370.6	377.5
51	365.4	370.9
52	data not available	data not available

	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank) \
0	N/A	N/A	N/A
1	1	1	1
2	2	2	5
3	3	2	5
4	4	2	8
5	5	4	8
6	6	3	9
7	7	3	9
8	8	2	11
9	9	9	13
10	10	7	18
11	11	7	20
12	12	8	16
13	13	7	21
14	14	11	21
15	15	12	20
16	16	11	21
17	17	12	21
18	18	13	21

19	19	11	23
20	20	15	22
21	21	18	23
22	22	13	26
23	23	20	24
24	24	23	27
25	25	23	30
26	26	21	35
27	27	25	31
28	28	26	33
29	29	23	36
30	30	26	34
31	31	27	34
32	32	24	37
33	33	28	35
34	34	28	36
35	35	28	37
36	36	34	37
37	37	35	38
38	38	37	38
39	39	39	40
40	40	40	46
41	41	40	45
42	42	40	44
43	43	40	43
44	44	43	46
45	45	43	47
46	46	42	48
47	47	46	47
48	48	47	48
49	49	49	49
50	50	50	50
51	N/A	N/A	N/A
52	N/A	N/A	N/A

	Average Annual Count	Recent Trend \
0	1728431	stable
1	27998	falling
2	19110	rising
3	53473	falling
4	12216	falling
5	116044	falling
6	26426	stable
7	17906	stable
8	8695	falling
9	80256	falling
10	9189	stable

11	6407	falling
12	16924	*
13	6008	falling
14	31253	stable
15	68972	stable
16	21622	stable
17	34173	falling
18	58411	falling
19	10457	stable
20	53463	falling
21	38326	falling
22	6455	falling
23	70185	falling
24	134730	falling
25	15621	falling
26	3903	falling
27	35999	falling
28	38547	stable
29	3894	falling
30	32515	falling
31	34317	falling
32	4749	falling
33	27407	falling
34	20705	stable
35	8879	stable
36	56208	falling
37	28333	falling
38	37988	falling
39	22327	falling
40	3022	falling
41	2855	stable
42	7537	falling
43	118438	stable
44	40801	falling
45	11212	falling
46	2846	falling
47	174350	falling
48	24436	stable
49	33179	falling
50	9627	falling
51	14806	stable
52	data not available	data not available

	Recent 5-Year Trend ([trend note]) in Incidence Rates \
0	-0.9
1	-0.9
2	0.8

3	-0.6
4	-0.2
5	-0.6
6	0.5
7	0.4
8	-0.7
9	-1.6
10	-0.2
11	-0.8
12	*
13	-1.3
14	-0.2
15	0.3
16	-0.5
17	-0.2
18	-0.6
19	0.5
20	-0.2
21	-0.5
22	-0.5
23	-0.8
24	-1.9
25	-0.6
26	-0.8
27	-3.2
28	-2.0
29	-0.3
30	-0.5
31	-0.7
32	-0.5
33	-0.7
34	-0.2
35	-0.6
36	-1.1
37	-2.3
38	-1.0
39	-0.9
40	-1.4
41	-0.5
42	-0.5
43	-0.1
44	-0.9
45	-0.4
46	-0.8
47	-0.8
48	-0.6
49	-2.2

50	-1.1
51	-0.1
52	data not available

	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
0	-2.0	0.2
1	-1.8	-0.1
2	0.4	1.2
3	-0.7	-0.5
4	-0.4	-0.1
5	-0.8	-0.4
6	-0.3	1.3
7	-0.5	1.4
8	-0.9	-0.6
9	-3.2	-0.1
10	-0.6	0.1
11	-1.0	-0.6
12	*	*
13	-1.6	-0.9
14	-0.5	0.0
15	-0.2	0.8
16	-1.0	0.0
17	-0.3	-0.1
18	-0.7	-0.4
19	-0.7	1.7
20	-0.3	-0.1
21	-0.8	-0.3
22	-0.7	-0.2
23	-1.1	-0.5
24	-3.5	-0.3
25	-0.8	-0.4
26	-1.0	-0.5
27	-4.6	-1.7
28	-4.1	0.1
29	-0.5	-0.1
30	-0.7	-0.3
31	-0.9	-0.5
32	-0.8	-0.2
33	-1.0	-0.3
34	-0.6	0.2
35	-1.7	0.6
36	-1.3	-0.9
37	-3.4	-1.1
38	-1.2	-0.9
39	-1.1	-0.7
40	-1.6	-1.2
41	-4.4	3.6

```
42                     -0.8                  -0.2
43                     -0.5                   0.4
44                     -1.3                  -0.6
45                     -0.5                  -0.2
46                     -1.1                  -0.5
47                     -1.3                  -0.3
48                     -1.2                   0.0
49                     -3.9                  -0.6
50                     -1.3                  -0.9
51                     -1.3                   1.2
52       data not available    data not available
```

[ ]:
```
pip install werkzeug==2.2.2
```

[ ]:
```
pip install flask-lambda --upgrade
```

[ ]:
```
pip install jinja2==3.0.3
```

[ ]:
```
pip install Werkzeug==2.0.3
```

[ ]:
```
%conda install -c conda-forge mysqlclient -y
```

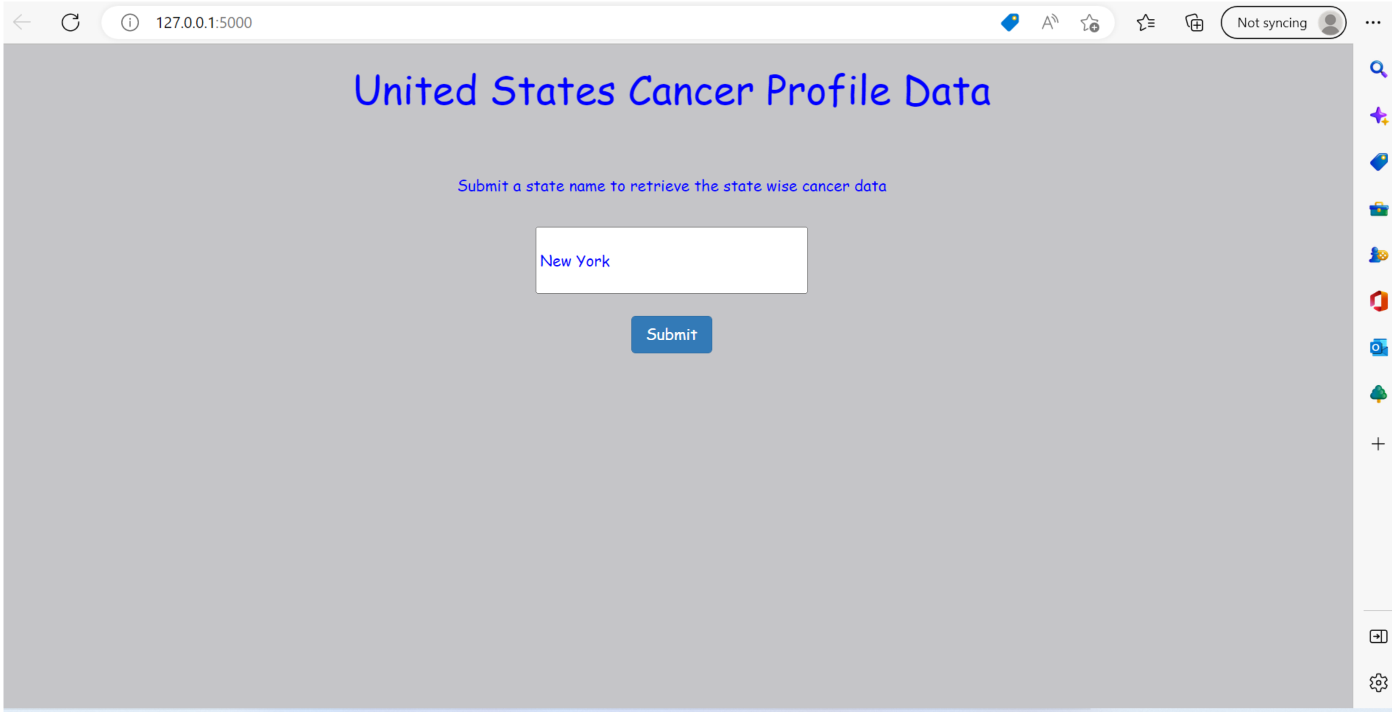
[ ]:
```
import pandas as pd
import json

data = pd.read_csv("clean_data.csv")

age_adjusted_IR = data['Age-Adjusted Incidence Rate([rate note]) - cases per␣
 ↪100,000'].tolist()
state_name1 = data['State'].tolist()
state_name = []
for state in state_name1:
    state_name.append(state.lower())

from flask import Flask, render_template, request, url_for
app = Flask(__name__)

@app.route("/")
def homepage():
    return render_template("index.html")

@app.route("/state/<string:name>")
def get_info():
    dics = {}
    for item in state_name:
        dics[item] = age_adjusted_IR[state_name.index(item)]
    json_object = json.dumps(dics)
```

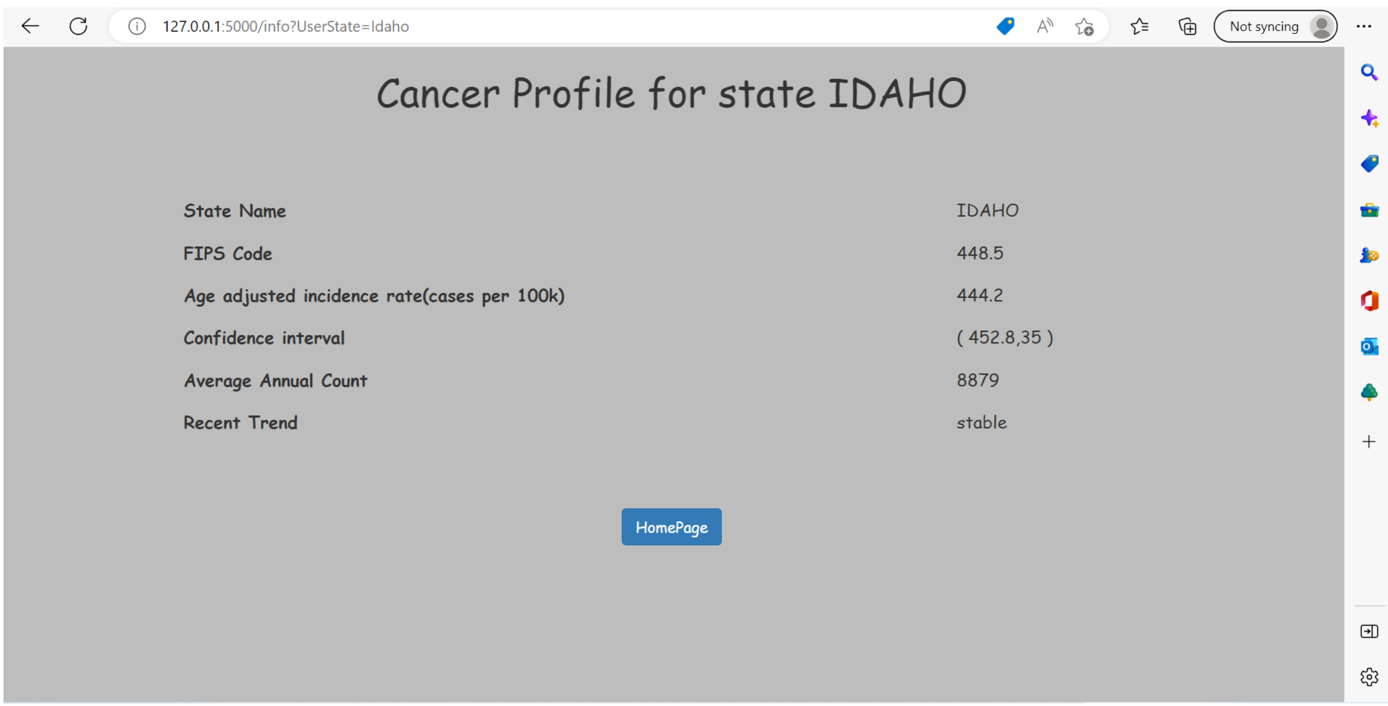
```python
    return json_object

@app.route("/info", methods=["GET"])
def info():
    UserState = request.args.get("UserState")
    result = ""
    IR = 0
    FIPS = 0
    CI = ""
    AAC = 0
    rec_trend = ""
    if UserState.lower().strip() in str(data['State']).lower():
        i=0
        for item in data['State']:
            print(item)
            if UserState.lower().strip() == item.lower():
                FIPS = data.iloc[i,2]
                print(FIPS)
                IR = data.iloc[i,3]
                print(IR)
                CI = "(" + str(data.iloc[i,4]) + "," + str(data.iloc[i,5]) + ")"
                AAC = int(data.iloc[i,8])
                rec_trend = data.iloc[i,9]
            i+=1
        result += f"State name: {UserState}, Age-Adjusted Incidence Rate - cases per 100,000  is {IR}.\n"
        return render_template("info.html", analysis = result, FIPS = FIPS, IR = IR, CI = CI, AAC = AAC, rec_trend = rec_trend, usertext = UserState)
    else:
        return f"Error: name {UserState} is invalid.\n"


if __name__ == "__main__":
    app.run()
```

[ ]:

[ ]:

```
    UserState = request.args.get("UserState")
    result = ""
    IR = 0
    FIPS = 0
    CI = ""
    AAC = 0
    rec_trend = ""
    if UserState.lower().strip() in str(data['State']).lower():
        i=0
        for item in data['State']:
            print(item)
            if UserState.lower().strip() == item.lower():
                FIPS = data.iloc[i,2]
                print(FIPS)
                IR = data.iloc[i,3]
                print(IR)
                CI = "(" + str(data.iloc[i,4]) + "," + str(data.iloc[i,5]) + '
                AAC = int(data.iloc[i,8])
                rec_trend = data.iloc[i,9]
            i+=1
        result += f"State name: {UserState}, Age-Adjusted Incidence Rate - ca
        return render_template("info.html", analysis = result, FIPS = FIPS, IR
    else:
        return f"Error: name {UserState} is invalid.\n"


if __name__ == "__main__":
    app.run()
```


127.0.0.1:5000
Not syncing
United States Cancer Profile Data
Submit a state name to retrieve the state wise cancer data
New York
Submit


127.0.0.1:5000/info?UserState=Idaho
Not syncing
Cancer Profile for state IDAHO
State Name IDAHO
FIPS Code 448.5
Age adjusted incidence rate(cases per 100k) 444.2
Confidence interval (452.8,35)
Average Annual Count 8879
Recent Trend stable
HomePage