BIS 634 01 / CB&B 634 01 Assignment 3

Submitted By : Mahima Kaur

Exercise 1

Implement a two-dimensional k-nearest neighbors classifier (in particular, do not use sklearn for k-nearest neighbors here): given a list of (x, y; class) data, store this data in a quad-tree (14 points). Given a new (x, y) point and a value of k (the number of nearest neighbors to examine), it should be able to identify the most common class within those k nearest neighbors (14 points). You may assume distance will be measured using the traditional euclidean distance.

Cinar and Koklu (2019) used various machine learning strategies to predict the type of rice based on image analysis. In this exercise, we'll do something similar.

Begin by going to the journal page and downloading the rice dataset (there's a dataset button on the right). Note that you can load data directly from the excel file via:

```
data = pd.read_excel('Rice_Osmancik_Cammeo_Dataset.xlsx')
```

Normalize the seven quantitative columns to a mean of 0 and standard deviation 1. (3 points)

Reduce the data to two dimensions using PCA. You can do this with e.g.:

```
from sklearn import decomposition
pca = decomposition.PCA(n_components=2)
data_reduced = pca.fit_transform(data[my_cols])
pc0 = data_reduced[:, 0]
pc1 = data_reduced[:, 1]
```

That is, pc0 can be thought of as a vector of x-values, pc1 as a vector of y-values, and data["CLASS"] as a vector of the corresponding types of rice.

Plot this on a scatterplot, color-coding by type of rice. (3 points)

Comment on what the graph suggests about the effeciveness of using k-nearest neighbors on this 2-dimensional reduction of the data to predict the type of rice. (4 points)

Using a reasonable train-test split with your k-nearest neighbors implementation, give the confusion matrix for predicting the type of rice with k=1. (4 points) Repeat for k=5. (4 points)

Provide a brief interpretation of what the confusion matrix results mean. (4 points)

Recall: All normalization must be based on the training set mean and standard deviation, not on data including the test set. Once the pca reduction has been trained, you can apply it to test data via, e.g.

```
test_data_reduced = pca.transform(test_data[my_cols])
```

(Note: we use .fit_transform with the training data and .transform with the test data.)

Hints:

- The quad tree can construct itself from all the data by dividing into children if there's enough points for that to be reasonable, each one of which constructs itself recursively. Only the leaf nodes need to keep track of the points; the parent trees can just ask for all_points recursively.It's theoretically possible that you could have repeated x,y points... don't get stuck in an infinite loop
    - If you're looking for the k nearest neighbors, to get your initial circle, stop descending if the child tree you'd go into doesn't have k points. (Alternatively, move up a level if you don't have enough points; you can do this easily if each tree knows its _parent.)
- I have a method contains(self, x, y) which determines if the box of my tree's bounds (xlo, ylo, xhi, yhi) contains the given (x,y) point... this is used by a _within_distance(self, x, y, d) that returns true or false depending on if the point is within a distance d of the box. This is used by a recursive leaves_within_distance function that finds all of the leaves (i.e. the ones with no children) within a given distance of a point.
- Once I had found the distances within my tree, to find the indices of the smallest k distances, I used the code: min_i = np.argpartition(distances, k)[:k]
- When I did this, I tested it by making a function that directly checked all points to find the k-nearest points (you can do this in about 4 lines of code) and used that to have a known truth to help debug/validate the quad-tree-based solution.

Response

In [665]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import plotnine as p9
import matplotlib.pyplot as plt
from sklearn import decomposition
import warnings
warnings.filterwarnings("ignore")
from sklearn.model_selection import KFold, train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

In [666]:

```
dataset = pd.read_excel('/Users/mahimakaur/desktop/Rice_Cammeo_Osmancik.xlsx')
```

In [667]:

```
dataset.head()
```

Out[667]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent	Class
0	15231	525.578979	229.749878	85.093788	0.928882	15617	0.572896	Cammeo
1	14656	494.311005	206.020065	91.730972	0.895405	15072	0.615436	Cammeo
2	14634	501.122009	214.106781	87.768288	0.912118	14954	0.693259	Cammeo
3	13176	458.342987	193.337387	87.448395	0.891861	13368	0.640669	Cammeo
4	14688	507.166992	211.743378	89.312454	0.906691	15262	0.646024	Cammeo

In [668]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3810 entries, 0 to 3809
Data columns (total 7 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Area               3810 non-null   int64  
 1   Perimeter          3810 non-null   float64
 2   Major_Axis_Length  3810 non-null   float64
 3   Minor_Axis_Length  3810 non-null   float64
 4   Eccentricity       3810 non-null   float64
 5   Convex_Area        3810 non-null   int64  
 6   Extent             3810 non-null   float64
dtypes: float64(5), int64(2)
memory usage: 208.5 KB
```

In [669]:

```
##Creating a dataset without the last column

data = dataset.iloc[:,0:7]
```

In [670]:

```
##creating a dataset for the class column

y = np.array(dataset.iloc[:,-1])
```

In [671]:

```
## Standardizing the dataset

std_data = (data - data.mean())/data.std()
```

In [672]:
```
std_data.describe()
```

Out[672]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent
count	3.810000e+03	3.810000e+03	3.810000e+03	3.810000e+03	3.810000e+03	3.810000e+03	3.810000e+03
mean	-2.124285e-16	-5.739940e-16	-1.181324e-16	-2.456478e-16	9.097418e-16	2.152259e-16	-4.754435e-16
std	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00	1.000000e+00
min	-2.953604e+00	-2.672668e+00	-2.493699e+00	-4.674031e+00	-5.266589e+00	-2.942926e+00	-2.130034e+00
25%	-7.488177e-01	-7.892340e-01	-8.265593e-01	-6.251605e-01	-6.950351e-01	-7.463521e-01	-8.165818e-01
50%	-1.421335e-01	-1.513238e-01	-1.699936e-01	2.109949e-02	1.046992e-01	-1.384360e-01	-2.145634e-01
75%	7.401849e-01	8.271624e-01	8.467241e-01	6.684203e-01	7.550077e-01	7.493101e-01	8.367238e-01
max	3.605050e+00	2.646475e+00	2.878973e+00	3.704952e+00	2.936761e+00	3.458976e+00	2.577922e+00

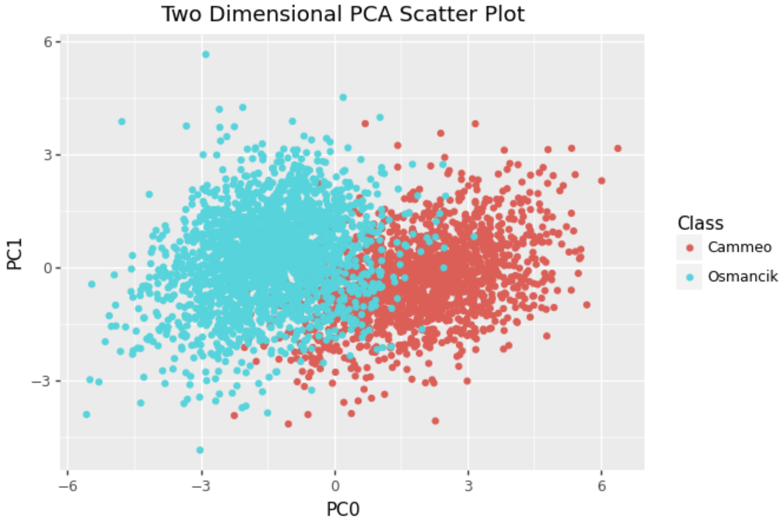
In [673]:
```
## Reducing the data to two dimensions using PCA

PCA = decomposition.PCA(n_components=2)
data_reduced = PCA.fit_transform(std_data)
PC0 = data_reduced[:, 0]
PC1 = data_reduced[:, 1]
```

In [674]:
```
data_reduced
```

Out[674]:
```
array([[ 3.81212784, -2.16504685],
       [ 2.47683257,  0.04529019],
       [ 2.63820924, -0.62153372],
       ...,
       [-0.43662669,  0.10358082],
       [-3.58746234, -0.37565233],
       [-2.55575212,  3.36079599]])
```

In [675]:
```
pc0 = PC0.tolist()
pc1 = PC1.tolist()
pca = pd.DataFrame({'PC0': pc0,'PC1': pc1,'Class': dataset['Class']})
```

In [676]:
```
print (
    p9.ggplot(data = pca, mapping = p9.aes(x='PC0', y='PC1'))
 + p9.geom_point(p9.aes(x = 'PC0', y = 'PC1', color = 'Class'))
 + p9.labs(title = "Two Dimensional PCA Scatter Plot")
)
```



Comment on what the graph suggests about the effeciveness of using k-nearest neighbors on this 2-dimensional reduction of the data to predict the type of rice.

We can somewhat see two clusters in the graphs, but there is no clear demarcation of the two clusters. The k-nearest neighbors might not an effective method to classify the data points. The region from -2 to 1 on the x-axis shows some overlap between the points of class "Cammeo" or "Osmancik" which can lead to inaccurate prediction of the type of rice.

Implementation of the KNN algorithm and Quad Tree

Citations :

- https://katherinepully.com/quadtree-python/ (https://katherinepully.com/quadtree-python/)
- https://www.geeksforgeeks.org/quad-tree/ (https://www.geeksforgeeks.org/quad-tree/)

Contructing a quadtree from a two-dimensional area using the following steps:

1.Divide the current two dimensional space into four boxes.
2.If a box contains one or more points in it, create a child object, storing in it the two dimensional space of the box.
3.If a box does not contain any points, do not create a child for it.
4.Recurse for each of the children.

In [677]:
```
indices = [i for i in range(len(PC0))]
data_knn = np.array([PC0,PC1,indices])
```

In [678]:
```
data_knn
```

Out[678]:
```
array([[ 3.81212784e+00,  2.47683257e+00,  2.63820924e+00, ...,
        -4.36626685e-01, -3.58746234e+00, -2.55575212e+00],
       [-2.16504685e+00,  4.52901938e-02, -6.21533719e-01, ...,
         1.03580824e-01, -3.75652326e-01,  3.36079599e+00],
       [ 0.00000000e+00,  1.00000000e+00,  2.00000000e+00, ...,
         3.80700000e+03,  3.80800000e+03,  3.80900000e+03]])
```

In [679]:

```
class QTree:

    def __init__(self,bounding_box,data,height = 0,max_leaf_data = 3):


        self.xlo = min(bounding_box[0][0],bounding_box[1][0],bounding_box[2][0],bounding_box[3][0])
        self.xhi = max(bounding_box[0][0],bounding_box[1][0],bounding_box[2][0],bounding_box[3][0])
        self.ylo = min(bounding_box[0][1],bounding_box[1][1],bounding_box[2][1],bounding_box[3][1])
        self.yhi = max(bounding_box[0][1],bounding_box[1][1],bounding_box[2][1],bounding_box[3][1])
        self.leafNode = False
        self.height = height
        self.points = []
        self.add(data)
        self.numPoints = len(self.points[0])

        if(self.numPoints<= max_leaf_data):
            self.leafNode = True
            return


        self.x_median = np.median(self.points[0])
        self.y_median = np.median(self.points[1])


        bounding_box_ul = [bounding_box[0],(self.xlo,self.y_median),(self.x_median,self.yhi),(self.x_median,self.y_median)]
        bounding_box_lr = [(self.x_median,self.y_median),(self.x_median,self.ylo),(self.xhi,self.y_median),bounding_box[3]]
        bounding_box_ll = [(self.xlo,self.y_median),bounding_box[1],(self.x_median,self.y_median),(self.x_median,self.ylo)]
        bounding_box_ur = [(self.x_median,self.yhi),(self.x_median,self.y_median),bounding_box[2],(self.xhi,self.y_median)]
        self.URChild = QTree(bounding_box_ur,self.points,height = self.height + 1,max_leaf_data = max_leaf_data)
        self.LRChild = QTree(bounding_box_lr,self.points,height = self.height + 1,max_leaf_data = max_leaf_data)
        self.ULChild = QTree(bounding_box_ul,self.points,height = self.height + 1,max_leaf_data = max_leaf_data)
        self.LLChild = QTree(bounding_box_ll,self.points,height = self.height + 1,max_leaf_data = max_leaf_data)
        self.children = [self.ULChild,self.LLChild,self.URChild,self.LRChild]

        if(self.leafNode == False):
            del self.points

    def contains(self,x,y):
        if((self.xlo <= x) and (x < self.xhi) and (self.ylo <= y) and (y < self.yhi)):
            return True
        return False

    def add(self,data):
        points_x = []
        points_y = []
        points_indices = []
        for i in range(0,len(data[0])):

            if(self.contains(data[0][i],data[1][i])):
                points_x.append(data[0][i])
                points_y.append(data[1][i])
                points_indices.append(data[2][i])

        self.points = np.array([points_x,points_y,points_indices],dtype = float)

    def getPoints(self):
        if(self.leafNode):
            return self.points

        return np.concatenate((self.ULChild.getPoints(),self.LLChild.getPoints(),self.URChild.getPoints(),self.LRChild.getPoints()),axis = 1)

    def Distance(self,x,y,xp,yp):
        return np.linalg.norm((x-xp,y-yp))

    def getHeight(self):
        if(self.leafNode):
            return self.height
        return max(self.ULChild.getHeight(),self.LLChild.getHeight(),self.URChild.getHeight(),self.LRChild.getHeight())


    def BBox(self,d,x,y):
        if(self.contains(x,y)):
            return True
        if(x<=self.xlo):
            if(y<=self.ylo):
                d_first = self.Distance(x,y,self.xlo,self.ylo)
                if(d_first<=d):
                    return True
            elif(y>=self.yhi):
                d_first = self.Distance(x,y,self.xlo,self.yhi)
                if(d_first<=d):
                    return True
            else:
                d_first = self.Distance(x,y,self.xlo,y)
                if(d_first<=d):
                    return True

        elif(x>self.xhi):
            if(y<=self.ylo):
                d_first = self.Distance(x,y,self.xhi,self.ylo)
                if(d_first<=d):
                    return True
            elif(y>=self.yhi):
                d_first = self.Distance(x,y,self.xhi,self.yhi)
                if(d_first<=d):
                    return True
            else:
                d_first = self.Distance(x,y,self.xhi,y)
                if(d_first<=d):
                    return True

        else:
            if(y<=self.ylo):
                d_first = self.Distance(x,y,x,self.ylo)
                if(d_first<=d):
                    return True
            elif(y>=self.ylo):
                d_first = self.Distance(x,y,x,self.yhi)
                if(d_first<=d):
                    return True

        return False


    def within_distance(self,d,x,y):
        if(self.leafNode):
            if(self.BBox(d,x,y)):
                return self.points
            else:
                return np.array([[],[],[]])
        return np.concatenate((self.ULChild.within_distance(d,x,y),self.LLChild.within_distance(d,x,y),self.URChild.within_distance(d,x,y),self.LRChild.within_distance(d,x,y)),axis = 1)

    def get_data_in_range(self,k,x,y):
        if(self.leafNode):
            return self


        for child in self.children:
            if(child.contains(x,y)):
                if(child.numPoints>=k):
                    return child.get_data_in_range(k,x,y)
                else:
                    return self
```

```
In [680]: def getQuadTree(data_knn,max_leaf_data = 3):
              xlo = min(data_knn[0]) - 2
              xhi= max(data_knn[0]) + 2
              ylo = min(data_knn[1]) - 2
              yhi = max(data_knn[1]) + 2
              bounding_box = [(xlo,yhi),(xlo,ylo),(xhi,yhi),(xhi,ylo)]
              q1 = QTree(bounding_box,data_knn,height = 0,max_leaf_data=max_leaf_data)
              return q1
```

Implementing k nearest neighbors using QuadTree

- Step 1 : Determine parameter k = number of nearest neighbors
- Step 2 : Calculate the distance between the query-instance and all the training samples (The most commonly used method to calculate distance is Euclidean)
- Step 3 : Sort the distance and determine the nearest neighbors based on the kth minimum distance
- Step 4 : Gather the category Y of the nearest neighbors
- Step 5 : Use majority voting of the category of the nearest neighnors as the prediction value.

Reference :

- https://serokell.io/blog/knn-algorithm-in-ml (https://serokell.io/blog/knn-algorithm-in-ml)
- https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html (https://people.revoledu.com/kardi/tutorial/KNN/KNN_Numerical-example.html)
- https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm (https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_knn_algorithm_finding_nearest_neighbors.htm)

numpy.argpartition() function is used to create a indirect partitioned copy of input array with its elements rearranged in such a way that the value of the element in k-th position is in the position it would be in a sorted array. All elements smaller than the k-th element are moved before this element and all equal or greater are moved behind it (https://www.geeksforgeeks.org/numpy-argpartition-in-python/ (https://www.geeksforgeeks.org/numpy-argpartition-in-python/))

```
In [681]: def knn_quad(k,x,y,quadtree):
              BoundingBox = quadtree.get_data_in_range(k,x,y)
              points =  BoundingBox.getPoints()
              Euclideandistance = []

              for i in range(len(points[0])):
                  dist = BoundingBox.Distance(x,y,points[0][i],points[1][i])
                  Euclideandistance.append(dist)
              k_smallest_indices = np.argpartition(Euclideandistance, k-1)[:k]


              kth_dist = Euclideandistance[k_smallest_indices[-1]]
              points2 = quadtree.within_distance(kth_dist,x,y)
              Euclideandistance2 = []

              for j in range(len(points2[0])):
                  dist = BoundingBox.Distance(x,y,points2[0][j],points2[1][j])
                  Euclideandistance2.append(dist)
              k_smallest_indices_2 = np.argpartition(Euclideandistance2, k-1)[:k]
              value = [points2[:,i] for i in k_smallest_indices_2]
              return value
```

```
In [682]: def get_Indices(arr_points):
              indices_arr = []
              for i in range(len(arr_points)):
                  indices_arr.append(int(arr_points[i][2]))
              return np.array(indices_arr,dtype = int)
```

```
In [683]: def knn_points(k,known_neighbors,x,y):
              test_neighbors = np.array([[x,y]])
              Euclideandistance = []

              for i in range(len(known_neighbors[0])):
                  neighbors = known_neighbors[:,i]

                  Euclideandistance.append(np.linalg.norm(test_neighbors - neighbors))


              k_smallest_indices = np.argpartition(Euclideandistance, k)[:k]
              return k_smallest_indices,Euclideandistance
```

Creating a function to get majority vote as For classification: A class label assigned to the majority of K Nearest Neighbors from the training dataset is considered as a predicted class for the new data point.

Reference : https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/ (https://www.analyticsvidhya.com/blog/2021/04/simple-understanding-and-implementation-of-knn-algorithm/)

```
In [684]: def majority_vote(y):
              values,counts = np.unique(y,return_counts=True)
              indices = np.argmax(counts)
              return values[indices]
```

Reasonable train-test split

If the dataset is relatively small (n < 10,000), 70:30 would be a suitable choice.

However, for smaller datasets (n < 1,000), each observation is extremely valuable, and we can’t spare any for validation. In this case, k-fold cross-validation is a better choice than the holdout method. This method is computationally expensive. Yet it offers an effective model evaluation over a small dataset by training multiple models.

Reference : https://www.baeldung.com/cs/train-test-datasets-ratio (https://www.baeldung.com/cs/train-test-datasets-ratio)

Confusion Matrix when k = 1 or k = 5 using k-fold cross-validation

Cross Validation is the procedure which has a single parameter called k that refers to the number of groups that a given data sample is to be split into. The value for k is chosen such that each train/test group of data samples is large enough to be statistically representative of the broader dataset. The choice of k is usually 5 or 10, but there is no formal rule. Cross-validation gives the model an opportunity to test on multiple splits so we can get a better idea on how the model will perform on unseen data.

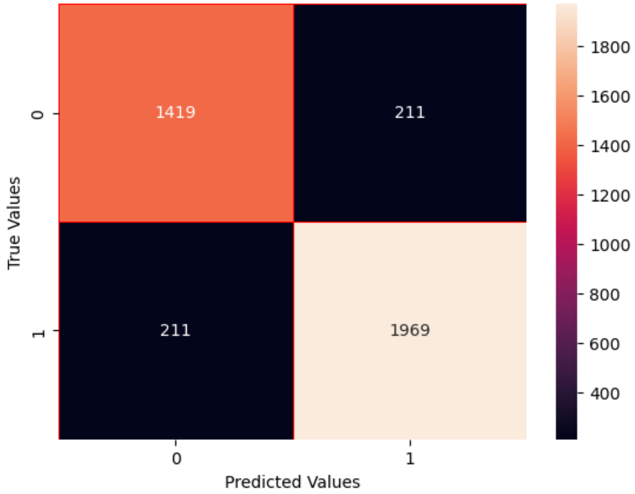
Cross Validation Citation : https://towardsdatascience.com/cross-validation-using-knn-6babb6e619c8 (https://towardsdatascience.com/cross-validation-using-knn-6babb6e619c8)

Confusion Matrix when k = 1 using k-fold cross-validation

In [685]:

```
pca = decomposition.PCA(n_components=2)
kfold = KFold(10,shuffle = True,random_state = 1)
scaler = StandardScaler()
counter = 0

Value_Pred = []
Value_True = []
First_Value = []

for train,test in (kfold.split(data_knn[2])):
    X_train_raw = dataset.iloc[train,0:7]
    X_test_raw = dataset.iloc[test,0:7]
    scaler.fit(X_train_raw)
    X_train_std = scaler.transform(X_train_raw)
    X_test_std = scaler.transform(X_test_raw)
    train_data_reduced = pca.fit_transform(X_train_std)
    test_data_reduced = pca.transform(X_test_std)
    train_data = np.array([train_data_reduced[:,0],train_data_reduced[:,1],train])
    test_data = np.array([test_data_reduced[:,0],test_data_reduced[:,1],test])

    quad_train = getQuadTree(train_data)

    for i in test:

        value_1 = knn_quad(1,test_data[0,test_data[2] == i],test_data[1,test_data[2] == i],quad_train)
        First_Value.append(value_1)
        k_smallest_indices = get_Indices(value_1)
        y_points = y[k_smallest_indices]
        value_pred = majority_vote(y_points)
        value_true = y[i]
        Value_Pred.append(value_pred)
        Value_True.append(value_true)


    counter+=1

print("The confusion matrix when k =1 is:")
cm = confusion_matrix(Value_True,Value_Pred)
sns.heatmap(cm,annot=True, linewidths= 0.5, linecolor="red", fmt=".0f")
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()
print("Classification Report:")
print(classification_report(Value_True,Value_Pred))
print("Accuracy Score:")
print(accuracy_score(Value_True,Value_Pred))
```

```
The confusion matrix when k =1 is:
```



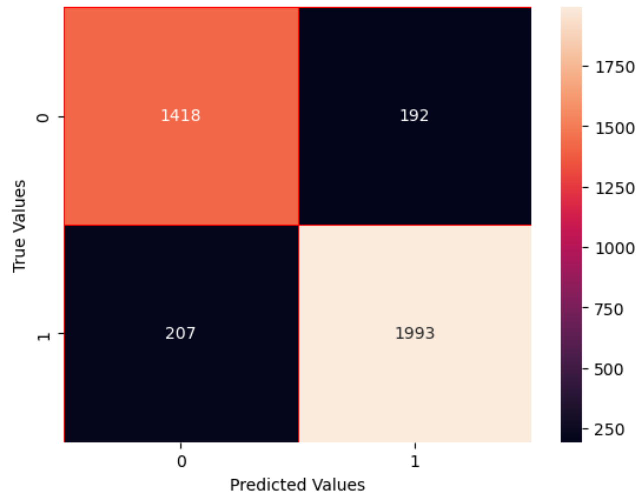
```
Classification Report:
              precision    recall  f1-score   support

      Cammeo       0.87      0.87      0.87      1630
    Osmancik       0.90      0.90      0.90      2180

    accuracy                           0.89      3810
   macro avg       0.89      0.89      0.89      3810
weighted avg       0.89      0.89      0.89      3810

Accuracy Score:
0.8892388451443569
```

Confusion Matrix when k = 5 using k-fold cross-validation

In [686]:

```
pca = decomposition.PCA(n_components=2)
kfold = KFold(10,shuffle = True,random_state = 1)
scaler = StandardScaler()
counter = 0

Value_Pred = []
Value_True = []
First_Value = []

for train_index,test_index in (kfold.split(data_knn[2])):
    X_train_raw = dataset.iloc[train_index,0:7]
    X_test_raw = dataset.iloc[test_index,0:7]
    scaler.fit(X_train_raw)
    X_train_std = scaler.transform(X_train_raw)
    X_test_std = scaler.transform(X_test_raw)
    train_data_reduced = pca.fit_transform(X_train_std)
    test_data_reduced = pca.transform(X_test_std)
    train_data = np.array([train_data_reduced[:,0],train_data_reduced[:,1],train])
    test_data = np.array([test_data_reduced[:,0],test_data_reduced[:,1],test])

    quad_train = getQuadTree(train_data)

    for i in test:

        value_5 = knn_quad(5,test_data[0,test_data[2] == i],test_data[1,test_data[2] == i],quad_train)
        First_Value.append(value_5)
        k_smallest_indices = get_Indices(value_5)
        y_points = y[k_smallest_indices]
        value_pred = majority_vote(y_points)
        value_true = y[i]
        Value_Pred.append(value_pred)
        Value_True.append(value_true)


    counter+=1

print("The confusion matrix when k = 5 is:")
cm = confusion_matrix(Value_True,Value_Pred)
sns.heatmap(cm,annot=True, linewidths= 0.5, linecolor="red", fmt=".0f")
plt.xlabel("Predicted Values")
plt.ylabel("True Values")
plt.show()
print("Classification Report:")
print(classification_report(Value_True,Value_Pred))
print("Accuracy Score:")
print(accuracy_score(Value_True,Value_Pred))
```

```
The confusion matrix when k = 5 is:
```



```
Classification Report:
              precision    recall  f1-score   support

      Cammeo       0.87      0.88      0.88      1610
    Osmancik       0.91      0.91      0.91      2200

    accuracy                           0.90      3810
   macro avg       0.89      0.89      0.89      3810
weighted avg       0.90      0.90      0.90      3810

Accuracy Score:
0.8952755905511811
```

Brief interpretation of what the confusion matrix results mean.

Confusion Matrix is a method of summarizing a classification algorithm’s performance. It is simply a summarized table of the number of correct and incorrect predictions. Calculating a confusion matrix can give us a better idea about the miss-classified classes. It is a square matrix where the column reflects actual values and the row represents the model’s predicted value or vice versa. rows show actual values and columns indicate predicted values.

- TP = A true positive is an outcome where the model correctly predicts the positive class.
- FP = The model predicted True and it is false.
- TN = The model predicted false and it is false.
- FN = The model predicted false and it is true.

When k = 1, the confusion matrix is (1419 211 211 1969). We can see that 1630 out of 1419 true classes were classified correctly i.e the model classified the type of rice as the data itself (TP). At the same time, 211 of them were placed incorrectly (FP), and 2180 out of 1969 false classes were classified correctly (TN), while 211 of them were misclassified (FN). The accuracy score of the model is 88.9%.

When k = 5, the confusion matrix is (1418 192 207 1993). We can see that 1610 out of 1418 true classes were classified correctly i.e the model classified the type of rice as the data itself (TP). At the same time, 192 of them were placed incorrectly (FP), and 2200 out of 1993 false classes were classified correctly (TN), while 207 of them were misclassified (FN). The accuracy score of the model is 89.5%.

Insights :

- As k increases the the accuracy of the model increases.
- The precision of clasiifying Rice Type : Osmancik is more than Cammeo in both the models.

Confusion matrix for predicting the type of rice with k=1 using train-test split - Using pre-existing library

```
In [687]: X = dataset.iloc[:,0:7]
```

```
In [688]: y = dataset['Class']
```

In [689]:

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train1 = scaler.transform(X_train)
X_test1 = scaler.transform(X_test)
```

In [690]:

```
PCA = decomposition.PCA(n_components=2)
```

In [691]:

```
X_train = PCA.fit_transform(X_train1)
```

In [692]:

```
X_test = PCA.transform(X_test1)
```

In [693]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

In [694]:

```
classifier = KNeighborsClassifier(n_neighbors = 1)
classifier.fit(X_train, y_train)
```

Out[694]:

```
KNeighborsClassifier(n_neighbors=1)
```

In [695]:

```
y_pred = classifier.predict(X_test)
```

In [696]:

```
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[ 658  123]
 [ 113 1011]]
Classification Report:
              precision    recall  f1-score   support

      Cammeo       0.85      0.84      0.85       781
    Osmancik       0.89      0.90      0.90      1124

    accuracy                           0.88      1905
   macro avg       0.87      0.87      0.87      1905
weighted avg       0.88      0.88      0.88      1905

Accuracy: 0.8761154855643044
```

Confusion matrix for predicting the type of rice with k=5 using train-test split.

In [697]:

```
X = dataset.iloc[:,0:7]
```

In [698]:

```
y = np.array(dataset.iloc[:,-1])
```

In [699]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
```

In [700]:

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [701]:

```
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(X_train, y_train)
```

Out[701]:

```
KNeighborsClassifier()
```

In [702]:

```
y_pred = classifier.predict(X_test)
```

In [703]:

```
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[588  61]
 [ 68 807]]
Classification Report:
              precision    recall  f1-score   support

      Cammeo       0.90      0.91      0.90       649
    Osmancik       0.93      0.92      0.93       875

    accuracy                           0.92      1524
   macro avg       0.91      0.91      0.91      1524
weighted avg       0.92      0.92      0.92      1524

Accuracy: 0.9153543307086615
```

Exercise 2

Describe your plan for your final project. I won't hold you to this, but it's good to have a plan. What's your data? (4 points) What analyses do you want to run and why are they interesting? (4 points) Which ones will be interactive, where the user can provide a parameter? (4 points; there must be at least one). What graphs will you make? (4 points) Describe how you want your website to work. (4 points) What do you see as your biggest challenge to completing this, and how do you expect to overcome this challenge? (5 points)

Response

Q1.What's your data?

I have decided to work on Diabetes Health Indictor BRFSS2015 dataset. It is a clean dataset of 253,680 survey responses to the CDC's BRFSS2015. This dataset has 21 feature variables.It is a subset of the original data. It was cleaned and consolidated dataset created from the BRFSS 2015 dataset already on Kaggle.The dataset was released by the CDC. The original dataset is provided by CDC from Kaggle public data repositories, named Behavioral Risk Factor Surveillance System The metadata of the dataset is available and licensed on Kaggle. Alex Teboul created the subset of the data and made it available on Kaggle for public use. The Dataset had 70692 rows and 22 columns.The variables include Diabetes status, HighBP, HighChol, CholCheck, BMI, Smoking status, Stroke Status, HeartDiseaseorAttack, PhysActivity, Fruits intake, Veggies intake,HvyAlcoholConsump, AnyHealthcare, NoDocbcCost, General health, Mental Health, Physical Activity, Difficulty in Walking, Gender, Income, Education Level and Age.

Q2. What analyses do you want to run and why are they interesting?

I plan to do descriptive and predictive analysis. Through the descriptve analysis we will able to find the which age group, gender, eductaion level etc is more prone to be diabetic. It is interesting to see such analysis as they will give an insight to develop targetted and evidence based intervention to lower the prevalence of diabetes. Additionally, I will do correlation analysis between diabetes and other variables in the dataset. I also aim to find the risk factors most predictive of diabetes which will help to classify the people whether they will have diabetes or not. I am planning to perfrom logostic regression, decision tree and random forest algorithm. In the end, I will see which algorithm had the highest accuracy to classify the patients. This is quite interesting to see as it can help individuals modify the variable leading to an increase in the risk of being diabetic.

Q3. Which ones will be interactive, where the user can provide a parameter?

The random forest algorithm will be interactive. The Web application build using flask would help the users the give the parameters and based on those attributes the model will predict whether the person can be diabetetic or not. Additionally, I plan to make some graphs intercative in the sense that it will show the text when clicked on certain points.

Q4. What graphs will you make?

I plan to make the following graphs :
a. Frequency distribution graphs of all the 22 variables.
b. Stratified graphs of diabetic and non-dabetic individuals v/s all the other variables.
c. Correlation Matrix.
d. Confusion Matrix of the algorithms.

Q5. Describe how you want your website to work.

Firstly the website will display some insights and prevalence of diabetes and other risk factors in US. There would be different tabs for each parameter such as Home, about the dataset,summary, descriptive analysis, and Predictive analysis. The Web application build using flask would help the users the give the parameters and based on those attributes the model will predict whether the person can be diabetetic or not.

Q6. What do you see as your biggest challenge to completing this, and how do you expect to overcome this challenge?

The first challenge which I faced was to find a good dataset. Secondly, it was challenging to to come to a conlusion about the type of analysis to be done on the dataset and whether to remove any outliers from the dataset. Thirdly, it is my first project in which I would be deploying and building a Web application using flask.Even though it is a challenge right now, I think it would be a great skill to learn and I am excited about it!

I explored a lot of dataset to find the one which is interesting to me. I decided not to remove any outliers from the data as I find those points would be helpful in determining the predictors of diabetes.For the third challenge, I believe that the resources shared,hands-on experience in class regarding flask and the assignment question would be helpful to overcome the challenge.

Exercise 3

Go to https://statecancerprofiles.cancer.gov/incidencerates/index.php (https://statecancerprofiles.cancer.gov/incidencerates/index.php) from the National Cancer Institute, search across the entire "area" of the United States at the "area type" resolution of By State, for all cancer sites, all races, all sexes, and all ages. Export the data (this will give you a CSV file).

Perform any necessary data cleaning (e.g. you'll probably want to get rid of the numbers in e.g. "Connecticut(7)" which refer to data source information as well as remove lines that aren't part of the table). Include the cleaned CSV file in your homework submission, and make sure your readme includes a citation of where the original data came from and how you changed the csv file. (5 points)

Using Flask, implement a server that provides three routes (5 points each):

@app.route("/") the index/homepage written in HTML which prompts the user to enter a state and provides a button, which passes this information to /info @app.route("/state/string:name") an API that returns JSON-encoded data containing the name of the state and the age-adjusted incidence rate (cases per 100k) NOTE: in this case, the name of the state is part of the URL itself; it is not passed in as a GET or POST argument @app.route("/info", methods=["GET"]) a web page that takes the name of the state as a GET argument and (1) if the state name is valid, displays the same information as the API above, but does so by putting it in an HTML page, or (2) displays an error if the state name is invalid (discuss what counts as valid and invalid in your readme; e.g. does capitalization matter?) either way, include a link back to the homepage / You've now completed most of this course, so you're now qualified to choose the next step. Take this exercise one step beyond that which is described above in a way that you think is appropriate, and discuss your extension in your readme. (e.g. you might show maps, or provide more data, or use CSS/JS to make the page prettier or more interactive, or use a database, or...) (5 points).

We will introduce Flask briefly at the beginning of the November 29th lecture. You might also want to check out CS50's lecture on Flask for more in-depth introduction: https://cs50.yale.edu/2022/fall/weeks/9/ (https://cs50.yale.edu/2022/fall/weeks/9/)

If you wish, you may start with the code at https://github.com/ramcdougal/flask-example/ (https://github.com/ramcdougal/flask-example/)

Since you're using GET, you'll want to use request.args.get("name") and not request.form.

Response

Citation :

- National Cancer Institute. State Cancer Profile (2022). https://statecancerprofiles.cancer.gov/incidencerates/index.php (https://statecancerprofiles.cancer.gov/incidencerates/index.php). Accessed on 3rd December 2022.
- Incidence data are provided by the National Program of Cancer RegistriesExternal Web Site Policy Cancer Surveillance System (NPCR-CSS), Centers for Disease Control and Prevention and by the National Cancer Institute's Surveillance, Epidemiology, and End Results (SEER)
- Population counts for denominators are based on Census populations as modified by NCI.
- Rates are calculated using SEER

In [11]:
```
import pandas as pd
```

In [12]:
```
##loading the raw dataset to do the cleaning

data = pd.read_csv("/Users/mahimakaur/Desktop/incd.csv", skiprows = 8)
```

```
In [13]: data
```

Out[13]:

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
0	US (SEER+NPCR)(1)	0.0	449.4	449.1	449.7	N/A	N/A	N/A	1728431	stable	-0.9	-2.0	0.2
1	Kentucky(7)	21000.0	516	513.2	518.8	1	1	1	27998	falling	-0.9	-1.8	-0.1
2	Iowa(7)	19000.0	490.7	487.5	494	2	2	5	19110	rising	0.8	0.4	1.2
3	New Jersey(7)	34000.0	488.9	487	490.8	3	2	5	53473	falling	-0.6	-0.7	-0.5
4	West Virginia(6)	54000.0	487.4	483.3	491.4	4	2	8	12216	falling	-0.2	-0.4	-0.1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
69	8 Source: Incidence data provided by the SEER ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
70	Interpret Rankings provides insight into inter...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
71	Data not available [http://statecancerprofiles...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
72	Data for the United States does not include da...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
73	Data for the United States does not include Pu...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

74 rows × 13 columns

```
In [14]: data.head(5)
```

Out[14]:

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
0	US (SEER+NPCR)(1)	0.0	449.4	449.1	449.7	N/A	N/A	N/A	1728431	stable	-0.9	-2.0	0.2
1	Kentucky(7)	21000.0	516	513.2	518.8	1	1	1	27998	falling	-0.9	-1.8	-0.1
2	Iowa(7)	19000.0	490.7	487.5	494	2	2	5	19110	rising	0.8	0.4	1.2
3	New Jersey(7)	34000.0	488.9	487	490.8	3	2	5	53473	falling	-0.6	-0.7	-0.5
4	West Virginia(6)	54000.0	487.4	483.3	491.4	4	2	8	12216	falling	-0.2	-0.4	-0.1

```
In [15]: data.tail(25)
```

Out[15]:

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
49	Arizona(6)	4000.0	382.4	380.6	384.3	49	49	49	33179	falling	-2.2	-3.9	-0.6
50	New Mexico(7)	35000.0	374	370.6	377.5	50	50	50	9627	falling	-1.1	-1.3	-0.9
51	Puerto Rico(6)	72001.0	368.2	365.4	370.9	N/A	N/A	N/A	14806	stable	-0.1	-1.3	1.2
52	Nevada(6)	32000.0	data not available	data not available	data not available	N/A	N/A	N/A	data not available	data not available	data not available	data not available	data not available
53	Created by statecancerprofiles.cancer.gov on 1...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
54	State Cancer Registries may provide more curre...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
55	Trend	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
56	Rising when 95% confidence interval of aver...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
57	Stable when 95% confidence interval of aver...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
58	Falling when 95% confidence interval of ave...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
59	[rate note] Incidence rates (cases per 100,000...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
60	[trend note] Incidence data come from differen...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
61	Rates and trends are computed using different ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
62	^ All Stages refers to any stage in the Survei...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
63	[rank note]Results presented with the CI*Rank ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
64	* Data has been suppressed to ensure confident...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
65	Data not available [http://statecancerprofiles...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
66	1 Source: National Program of Cancer Registrie...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
67	6 Source: National Program of Cancer Registrie...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
68	7 Source: SEER November 2021 submission.	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
69	8 Source: Incidence data provided by the SEER ...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
70	Interpret Rankings provides insight into inter...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
71	Data not available [http://statecancerprofiles...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
72	Data for the United States does not include da...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
73	Data for the United States does not include Pu...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

```
In [16]: ## Dropping the row from 53 to 73 as it contains no relevant data

         data = data.drop(data.index[53:74])
```

In [17]:

```
data
```

Out[17]:

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
0	US (SEER+NPCR)(1)	0.0	449.4	449.1	449.7	N/A	N/A	N/A	1728431	stable	-0.9	-2.0	0.2
1	Kentucky(7)	21000.0	516	513.2	518.8	1	1	1	27998	falling	-0.9	-1.8	-0.1
2	Iowa(7)	19000.0	490.7	487.5	494	2	2	5	19110	rising	0.8	0.4	1.2
3	New Jersey(7)	34000.0	488.9	487	490.8	3	2	5	53473	falling	-0.6	-0.7	-0.5
4	West Virginia(6)	54000.0	487.4	483.3	491.4	4	2	8	12216	falling	-0.2	-0.4	-0.1
5	New York(7)	36000.0	484.8	483.6	486.1	5	4	8	116044	falling	-0.6	-0.8	-0.4
6	Louisiana(7)	22000.0	484.3	481.7	487	6	3	9	26426	stable	0.5	-0.3	1.3
7	Arkansas(6)	5000.0	483.6	480.4	486.9	7	3	9	17906	stable	0.4	-0.5	1.4
8	New Hampshire(6)	33000.0	482.9	478.2	487.7	8	2	11	8695	falling	-0.7	-0.9	-0.6
9	Pennsylvania(6)	42000.0	476.8	475.3	478.3	9	9	13	80256	falling	-1.6	-3.2	-0.1
10	Maine(6)	23000.0	476.7	472.2	481.3	10	7	18	9189	stable	-0.2	-0.6	0.1
11	Rhode Island(6)	44000.0	476.2	470.8	481.6	11	7	20	6407	falling	-0.8	-1.0	-0.6
12	Mississippi(6)	28000.0	476	472.7	479.3	12	8	16	16924	*	*	*	*
13	Delaware(6)	10000.0	474.7	469.1	480.3	13	7	21	6008	falling	-1.3	-1.6	-0.9
14	Minnesota(6)	27000.0	471.5	469.2	474	14	11	21	31253	stable	-0.2	-0.5	0.0
15	Ohio(6)	39000.0	471.5	469.8	473.1	15	12	20	68972	stable	0.3	-0.2	0.8
16	Connecticut(7)	9000.0	471.4	468.5	474.3	16	11	21	21622	stable	-0.5	-1.0	0.0
17	Wisconsin(6)	55000.0	470.8	468.5	473.1	17	12	21	34173	falling	-0.2	-0.3	-0.1
18	North Carolina(6)	37000.0	469.9	468.2	471.7	18	13	21	58411	falling	-0.6	-0.7	-0.4
19	Nebraska(6)	31000.0	469.7	465.6	473.8	19	11	23	10457	stable	0.5	-0.7	1.7
20	Georgia(7)	13000.0	468.6	466.8	470.4	20	15	22	53463	falling	-0.2	-0.3	-0.1
21	Tennessee(6)	47000.0	466.5	464.4	468.7	21	18	23	38326	falling	-0.5	-0.8	-0.3
22	Montana(6)	30000.0	466.3	461	471.7	22	13	26	6455	falling	-0.5	-0.7	-0.2
23	Illinois(7)	17000.0	465.2	463.6	466.8	23	20	24	70185	falling	-0.8	-1.1	-0.5
24	Florida(6)	12000.0	460.5	459.4	461.6	24	23	27	134730	falling	-1.9	-3.5	-0.3
25	Kansas(6)	20000.0	459.4	456.1	462.7	25	23	30	15621	falling	-0.6	-0.8	-0.4
26	Vermont(6)	50000.0	457	450.3	463.8	26	21	35	3903	falling	-0.8	-1.0	-0.5
27	Indiana(6)	18000.0	456.8	454.6	458.9	27	25	31	35999	falling	-3.2	-4.6	-1.7
28	Massachusetts(7)	25000.0	454.8	452.7	456.8	28	26	33	38547	stable	-2.0	-4.1	0.1
29	North Dakota(6)	38000.0	454.4	447.8	461.1	29	23	36	3894	falling	-0.3	-0.5	-0.1
30	Maryland(6)	24000.0	454.1	451.9	456.4	30	26	34	32515	falling	-0.5	-0.7	-0.3
31	Missouri(6)	29000.0	453.2	451	455.4	31	27	34	34317	falling	-0.7	-0.9	-0.5
32	South Dakota(6)	46000.0	452.3	446.4	458.3	32	24	37	4749	falling	-0.5	-0.8	-0.2
33	Alabama(6)	1000.0	451.7	449.3	454.2	33	28	35	27407	falling	-0.7	-1.0	-0.3
34	Oklahoma(6)	40000.0	450.8	448	453.6	34	28	36	20705	stable	-0.2	-0.6	0.2
35	Idaho(7)	16000.0	448.5	444.2	452.8	35	28	37	8879	stable	-0.6	-1.7	0.6
36	Michigan(6)	26000.0	446.7	445	448.4	36	34	37	56208	falling	-1.1	-1.3	-0.9
37	South Carolina(6)	45000.0	443.8	441.4	446.2	37	35	38	28333	falling	-2.3	-3.4	-1.1
38	Washington(1)	53000.0	441.3	439.2	443.3	38	37	38	37988	falling	-1.0	-1.2	-0.9
39	Oregon(6)	41000.0	428.4	425.8	431	39	39	40	22327	falling	-0.9	-1.1	-0.7
40	Alaska(6)	2900.0	417	410	424.1	40	40	46	3022	falling	-1.4	-1.6	-1.2
41	District of Columbia(6)	11001.0	416.9	410	424	41	40	45	2855	stable	-0.5	-4.4	3.6
42	Hawaii(7)	15000.0	416.8	412.5	421.2	42	40	44	7537	falling	-0.5	-0.8	-0.2

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
43	Texas(7)	48000.0	415.3	414.3	416.4	43	40	43	118438	stable	-0.1	-0.5	0.4
44	Virginia(6)	51000.0	409.4	407.6	411.2	44	43	46	40801	falling	-0.9	-1.3	-0.6
45	Utah(7)	49000.0	407.2	403.8	410.6	45	43	47	11212	falling	-0.4	-0.5	-0.2
46	Wyoming(6)	56000.0	405.7	398.9	412.7	46	42	48	2846	falling	-0.8	-1.1	-0.5
47	California(7)	6000.0	402.4	401.6	403.3	47	46	47	174350	falling	-0.8	-1.3	-0.3
48	Colorado(6)	8000.0	396.4	394.1	398.6	48	47	48	24436	stable	-0.6	-1.2	0.0
49	Arizona(6)	4000.0	382.4	380.6	384.3	49	49	49	33179	falling	-2.2	-3.9	-0.6
50	New Mexico(7)	35000.0	374	370.6	377.5	50	50	50	9627	falling	-1.1	-1.3	-0.9
51	Puerto Rico(6)	72001.0	368.2	365.4	370.9	N/A	N/A	N/A	14806	stable	-0.1	-1.3	1.2
52	Nevada(6)	32000.0	data not available	data not available	data not available	N/A	N/A	N/A	data not available	data not available	data not available	data not available	data not available

In [18]:

```
## to remove the numbers from the states

data['State'] = data['State'].str.replace("\(\d\)","",regex = True)
```

In [20]:

```
data = data.rename(columns = {" FIPS": "FIPS"})
```

In [21]:

```
## Standarizing the FIPS Codes

data['FIPS'] = (data['FIPS']/1000).map("{:,.0f}".format)
```

In [23]:

```
# Replace the (*) from Mississippi as Data Suppressed (as mentioned on the website)

data.iloc[:,-4:] = data.iloc[:,-4:].replace("*", "Data Suppressed")
```

```
In [24]: data
```

Out[24]:

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
0	US (SEER+NPCR)	0	449.4	449.1	449.7	N/A	N/A	N/A	1728431	stable	-0.9	-2.0	0.2
1	Kentucky	21	516	513.2	518.8	1	1	1	27998	falling	-0.9	-1.8	-0.1
2	Iowa	19	490.7	487.5	494	2	2	5	19110	rising	0.8	0.4	1.2
3	New Jersey	34	488.9	487	490.8	3	2	5	53473	falling	-0.6	-0.7	-0.5
4	West Virginia	54	487.4	483.3	491.4	4	2	8	12216	falling	-0.2	-0.4	-0.1
5	New York	36	484.8	483.6	486.1	5	4	8	116044	falling	-0.6	-0.8	-0.4
6	Louisiana	22	484.3	481.7	487	6	3	9	26426	stable	0.5	-0.3	1.3
7	Arkansas	5	483.6	480.4	486.9	7	3	9	17906	stable	0.4	-0.5	1.4
8	New Hampshire	33	482.9	478.2	487.7	8	2	11	8695	falling	-0.7	-0.9	-0.6
9	Pennsylvania	42	476.8	475.3	478.3	9	9	13	80256	falling	-1.6	-3.2	-0.1
10	Maine	23	476.7	472.2	481.3	10	7	18	9189	stable	-0.2	-0.6	0.1
11	Rhode Island	44	476.2	470.8	481.6	11	7	20	6407	falling	-0.8	-1.0	-0.6
12	Mississippi	28	476	472.7	479.3	12	8	16	16924	Data Suppressed	Data Suppressed	Data Suppressed	Data Suppressed
13	Delaware	10	474.7	469.1	480.3	13	7	21	6008	falling	-1.3	-1.6	-0.9
14	Minnesota	27	471.5	469.2	474	14	11	21	31253	stable	-0.2	-0.5	0.0
15	Ohio	39	471.5	469.8	473.1	15	12	20	68972	stable	0.3	-0.2	0.8
16	Connecticut	9	471.4	468.5	474.3	16	11	21	21622	stable	-0.5	-1.0	0.0
17	Wisconsin	55	470.8	468.5	473.1	17	12	21	34173	falling	-0.2	-0.3	-0.1
18	North Carolina	37	469.9	468.2	471.7	18	13	21	58411	falling	-0.6	-0.7	-0.4
19	Nebraska	31	469.7	465.6	473.8	19	11	23	10457	stable	0.5	-0.7	1.7
20	Georgia	13	468.6	466.8	470.4	20	15	22	53463	falling	-0.2	-0.3	-0.1
21	Tennessee	47	466.5	464.4	468.7	21	18	23	38326	falling	-0.5	-0.8	-0.3
22	Montana	30	466.3	461	471.7	22	13	26	6455	falling	-0.5	-0.7	-0.2
23	Illinois	17	465.2	463.6	466.8	23	20	24	70185	falling	-0.8	-1.1	-0.5
24	Florida	12	460.5	459.4	461.6	24	23	27	134730	falling	-1.9	-3.5	-0.3
25	Kansas	20	459.4	456.1	462.7	25	23	30	15621	falling	-0.6	-0.8	-0.4
26	Vermont	50	457	450.3	463.8	26	21	35	3903	falling	-0.8	-1.0	-0.5
27	Indiana	18	456.8	454.6	458.9	27	25	31	35999	falling	-3.2	-4.6	-1.7
28	Massachusetts	25	454.8	452.7	456.8	28	26	33	38547	stable	-2.0	-4.1	0.1
29	North Dakota	38	454.4	447.8	461.1	29	23	36	3894	falling	-0.3	-0.5	-0.1
30	Maryland	24	454.1	451.9	456.4	30	26	34	32515	falling	-0.5	-0.7	-0.3
31	Missouri	29	453.2	451	455.4	31	27	34	34317	falling	-0.7	-0.9	-0.5
32	South Dakota	46	452.3	446.4	458.3	32	24	37	4749	falling	-0.5	-0.8	-0.2
33	Alabama	1	451.7	449.3	454.2	33	28	35	27407	falling	-0.7	-1.0	-0.3
34	Oklahoma	40	450.8	448	453.6	34	28	36	20705	stable	-0.2	-0.6	0.2
35	Idaho	16	448.5	444.2	452.8	35	28	37	8879	stable	-0.6	-1.7	0.6
36	Michigan	26	446.7	445	448.4	36	34	37	56208	falling	-1.1	-1.3	-0.9
37	South Carolina	45	443.8	441.4	446.2	37	35	38	28333	falling	-2.3	-3.4	-1.1
38	Washington	53	441.3	439.2	443.3	38	37	38	37988	falling	-1.0	-1.2	-0.9
39	Oregon	41	428.4	425.8	431	39	39	40	22327	falling	-0.9	-1.1	-0.7
40	Alaska	3	417	410	424.1	40	40	46	3022	falling	-1.4	-1.6	-1.2
41	District of Columbia	11	416.9	410	424	41	40	45	2855	stable	-0.5	-4.4	3.6

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
42	Hawaii	15	416.8	412.5	421.2	42	40	44	7537	falling	-0.5	-0.8	-0.2
43	Texas	48	415.3	414.3	416.4	43	40	43	118438	stable	-0.1	-0.5	0.4
44	Virginia	51	409.4	407.6	411.2	44	43	46	40801	falling	-0.9	-1.3	-0.6
45	Utah	49	407.2	403.8	410.6	45	43	47	11212	falling	-0.4	-0.5	-0.2
46	Wyoming	56	405.7	398.9	412.7	46	42	48	2846	falling	-0.8	-1.1	-0.5
47	California	6	402.4	401.6	403.3	47	46	47	174350	falling	-0.8	-1.3	-0.3
48	Colorado	8	396.4	394.1	398.6	48	47	48	24436	stable	-0.6	-1.2	0.0
49	Arizona	4	382.4	380.6	384.3	49	49	49	33179	falling	-2.2	-3.9	-0.6
50	New Mexico	35	374	370.6	377.5	50	50	50	9627	falling	-1.1	-1.3	-0.9
51	Puerto Rico	72	368.2	365.4	370.9	N/A	N/A	N/A	14806	stable	-0.1	-1.3	1.2
52	Nevada	32	data not available	data not available	data not available	N/A	N/A	N/A	data not available	data not available	data not available	data not available	data not available

```
In [26]: ## We can also drop the row for the state Nevada since no data is available

         data = data.drop(data.index[52])
```

```
In [27]: data
```

Out[27]:

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
0	US (SEER+NPCR)	0	449.4	449.1	449.7	N/A	N/A	N/A	1728431	stable	-0.9	-2.0	0.2
1	Kentucky	21	516	513.2	518.8	1	1	1	27998	falling	-0.9	-1.8	-0.1
2	Iowa	19	490.7	487.5	494	2	2	5	19110	rising	0.8	0.4	1.2
3	New Jersey	34	488.9	487	490.8	3	2	5	53473	falling	-0.6	-0.7	-0.5
4	West Virginia	54	487.4	483.3	491.4	4	2	8	12216	falling	-0.2	-0.4	-0.1
5	New York	36	484.8	483.6	486.1	5	4	8	116044	falling	-0.6	-0.8	-0.4
6	Louisiana	22	484.3	481.7	487	6	3	9	26426	stable	0.5	-0.3	1.3
7	Arkansas	5	483.6	480.4	486.9	7	3	9	17906	stable	0.4	-0.5	1.4
8	New Hampshire	33	482.9	478.2	487.7	8	2	11	8695	falling	-0.7	-0.9	-0.6
9	Pennsylvania	42	476.8	475.3	478.3	9	9	13	80256	falling	-1.6	-3.2	-0.1
10	Maine	23	476.7	472.2	481.3	10	7	18	9189	stable	-0.2	-0.6	0.1
11	Rhode Island	44	476.2	470.8	481.6	11	7	20	6407	falling	-0.8	-1.0	-0.6
12	Mississippi	28	476	472.7	479.3	12	8	16	16924	Data Suppressed	Data Suppressed	Data Suppressed	Data Suppressed
13	Delaware	10	474.7	469.1	480.3	13	7	21	6008	falling	-1.3	-1.6	-0.9
14	Minnesota	27	471.5	469.2	474	14	11	21	31253	stable	-0.2	-0.5	0.0
15	Ohio	39	471.5	469.8	473.1	15	12	20	68972	stable	0.3	-0.2	0.8
16	Connecticut	9	471.4	468.5	474.3	16	11	21	21622	stable	-0.5	-1.0	0.0
17	Wisconsin	55	470.8	468.5	473.1	17	12	21	34173	falling	-0.2	-0.3	-0.1
18	North Carolina	37	469.9	468.2	471.7	18	13	21	58411	falling	-0.6	-0.7	-0.4
19	Nebraska	31	469.7	465.6	473.8	19	11	23	10457	stable	0.5	-0.7	1.7
20	Georgia	13	468.6	466.8	470.4	20	15	22	53463	falling	-0.2	-0.3	-0.1
21	Tennessee	47	466.5	464.4	468.7	21	18	23	38326	falling	-0.5	-0.8	-0.3
22	Montana	30	466.3	461	471.7	22	13	26	6455	falling	-0.5	-0.7	-0.2
23	Illinois	17	465.2	463.6	466.8	23	20	24	70185	falling	-0.8	-1.1	-0.5
24	Florida	12	460.5	459.4	461.6	24	23	27	134730	falling	-1.9	-3.5	-0.3
25	Kansas	20	459.4	456.1	462.7	25	23	30	15621	falling	-0.6	-0.8	-0.4
26	Vermont	50	457	450.3	463.8	26	21	35	3903	falling	-0.8	-1.0	-0.5
27	Indiana	18	456.8	454.6	458.9	27	25	31	35999	falling	-3.2	-4.6	-1.7
28	Massachusetts	25	454.8	452.7	456.8	28	26	33	38547	stable	-2.0	-4.1	0.1
29	North Dakota	38	454.4	447.8	461.1	29	23	36	3894	falling	-0.3	-0.5	-0.1
30	Maryland	24	454.1	451.9	456.4	30	26	34	32515	falling	-0.5	-0.7	-0.3
31	Missouri	29	453.2	451	455.4	31	27	34	34317	falling	-0.7	-0.9	-0.5
32	South Dakota	46	452.3	446.4	458.3	32	24	37	4749	falling	-0.5	-0.8	-0.2
33	Alabama	1	451.7	449.3	454.2	33	28	35	27407	falling	-0.7	-1.0	-0.3
34	Oklahoma	40	450.8	448	453.6	34	28	36	20705	stable	-0.2	-0.6	0.2
35	Idaho	16	448.5	444.2	452.8	35	28	37	8879	stable	-0.6	-1.7	0.6
36	Michigan	26	446.7	445	448.4	36	34	37	56208	falling	-1.1	-1.3	-0.9
37	South Carolina	45	443.8	441.4	446.2	37	35	38	28333	falling	-2.3	-3.4	-1.1
38	Washington	53	441.3	439.2	443.3	38	37	38	37988	falling	-1.0	-1.2	-0.9
39	Oregon	41	428.4	425.8	431	39	39	40	22327	falling	-0.9	-1.1	-0.7
40	Alaska	3	417	410	424.1	40	40	46	3022	falling	-1.4	-1.6	-1.2
41	District of Columbia	11	416.9	410	424	41	40	45	2855	stable	-0.5	-4.4	3.6

	State	FIPS	Age-Adjusted Incidence Rate([rate note]) - cases per 100,000	Lower 95% Confidence Interval	Upper 95% Confidence Interval	CI*Rank([rank note])	Lower CI (CI*Rank)	Upper CI (CI*Rank)	Average Annual Count	Recent Trend	Recent 5-Year Trend ([trend note]) in Incidence Rates	Lower 95% Confidence Interval.1	Upper 95% Confidence Interval.1
42	Hawaii	15	416.8	412.5	421.2	42	40	44	7537	falling	-0.5	-0.8	-0.2
43	Texas	48	415.3	414.3	416.4	43	40	43	118438	stable	-0.1	-0.5	0.4
44	Virginia	51	409.4	407.6	411.2	44	43	46	40801	falling	-0.9	-1.3	-0.6
45	Utah	49	407.2	403.8	410.6	45	43	47	11212	falling	-0.4	-0.5	-0.2
46	Wyoming	56	405.7	398.9	412.7	46	42	48	2846	falling	-0.8	-1.1	-0.5
47	California	6	402.4	401.6	403.3	47	46	47	174350	falling	-0.8	-1.3	-0.3
48	Colorado	8	396.4	394.1	398.6	48	47	48	24436	stable	-0.6	-1.2	0.0
49	Arizona	4	382.4	380.6	384.3	49	49	49	33179	falling	-2.2	-3.9	-0.6
50	New Mexico	35	374	370.6	377.5	50	50	50	9627	falling	-1.1	-1.3	-0.9
51	Puerto Rico	72	368.2	365.4	370.9	N/A	N/A	N/A	14806	stable	-0.1	-1.3	1.2

In [28]:

```
## saving the cleaned dataset

data.to_csv('cleaned_cancer_data.csv', index= True)
```

Flask Implementation

Note : The code is uploaded to the github as exercise3.py

```
import pandas as pd
import json
```

```
data = pd.read_csv("cleaned_cancer_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())
```

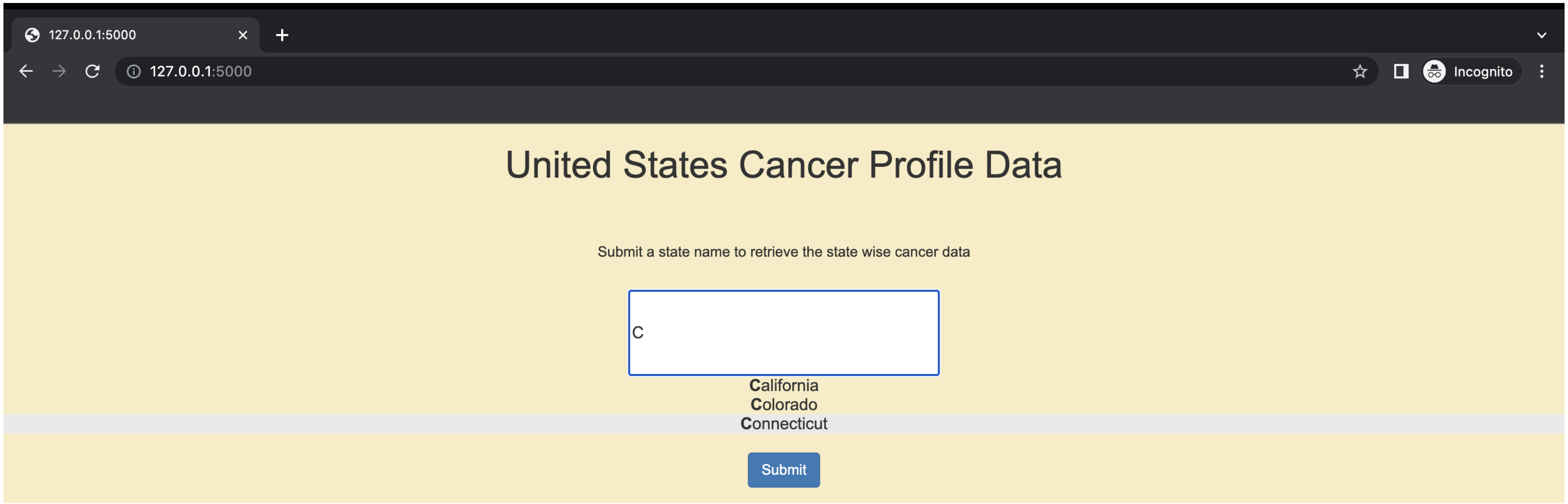
```
from flask import Flask, render_template, request, url_for
app = Flask(__name__)

@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)
    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,6])
                rec_trend = data.iloc[i,7]
            i+=1
        result += f"The state name is {UserState}, the age-adjusted incidence rate(cases per 100k) is {IR}.\n"
        print(result)
        return render_template("info.html", analysis = result, FIPS = FIPS, IR = IR, CI = CI, AAC = AAC, rec_trend = rec_trend, usertext = UserState)
    else:
        result += f"Error: the state name {UserState} is invalid.\n"
        return render_template("error.html", analysis = result, usertext = UserState)

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

In [2]:
```
from PIL import Image
```

In [4]:
```
try:
    HomePage = Image.open("/Users/mahimakaur/Desktop/Computional Methods for Health Informatics/StateName.png")
except IOError:
    pass
```

In [5]:
```
HomePage
```

Out[5]:



United States Cancer Profile Data

Submit a state name to retrieve the state wise cancer data

C

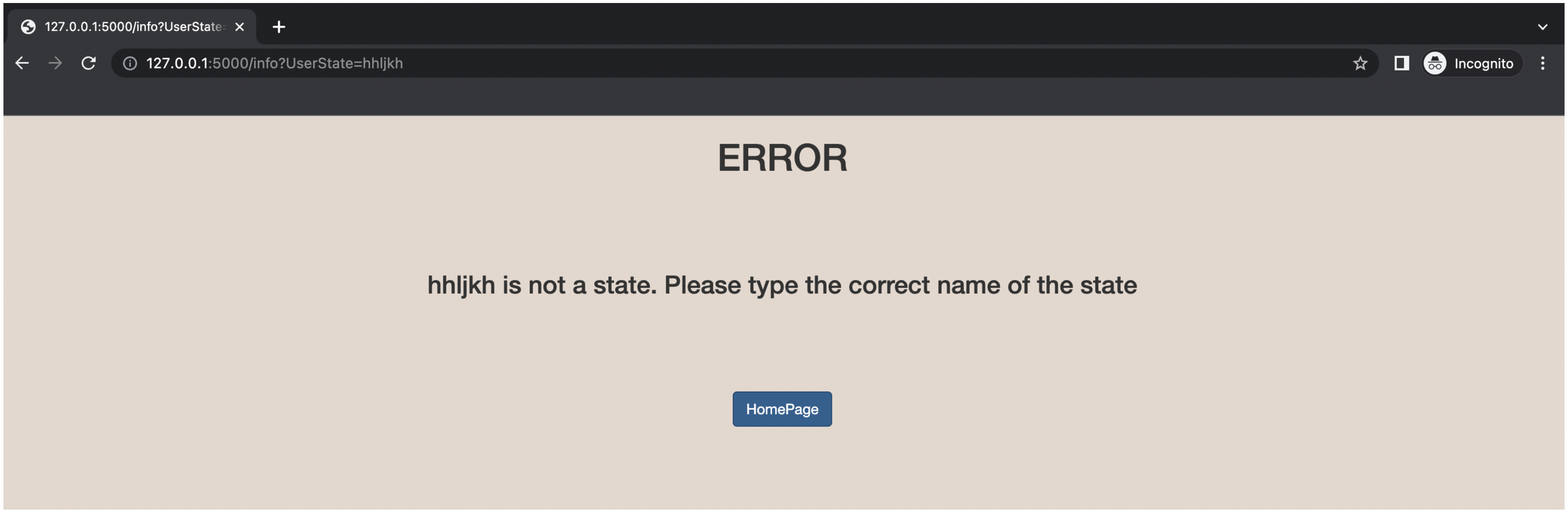
California
Colorado
Connecticut

Submit

In [6]:
```
try:
    ErrorPage = Image.open("/Users/mahimakaur/Desktop/Computional Methods for Health Informatics/ErrorPage.png")
except IOError:
    pass
```

```
In [7]: ErrorPage
```

Out[7]:



127.0.0.1:5000/info?UserState
127.0.0.1:5000/info?UserState=hhljkh
Incognito

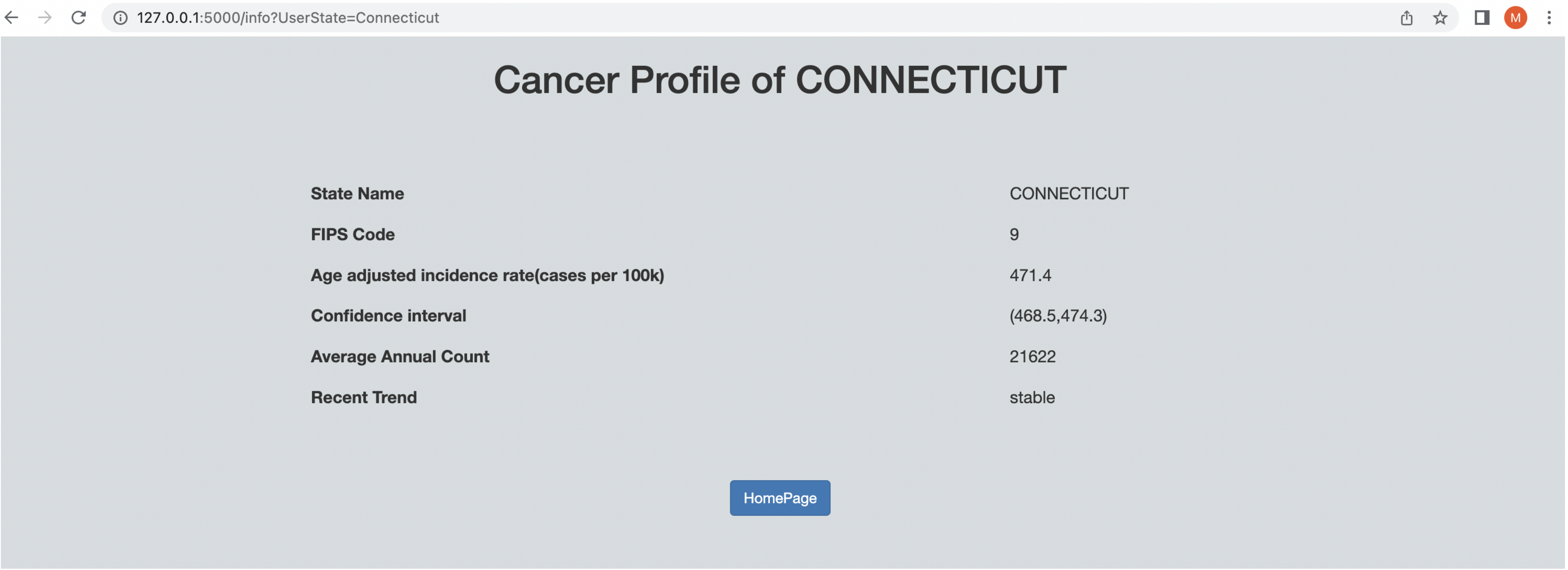
ERROR

hhljkh is not a state. Please type the correct name of the state

```
In [3]: try:
            StateData = Image.open("/Users/mahimakaur/Desktop/Computional Methods for Health Informatics/StateData.png")
        except IOError:
            pass
```

In [30]:
```
StateData
```

Out[30]:



127.0.0.1:5000/info?UserState=Connecticut

Cancer Profile of CONNECTICUT

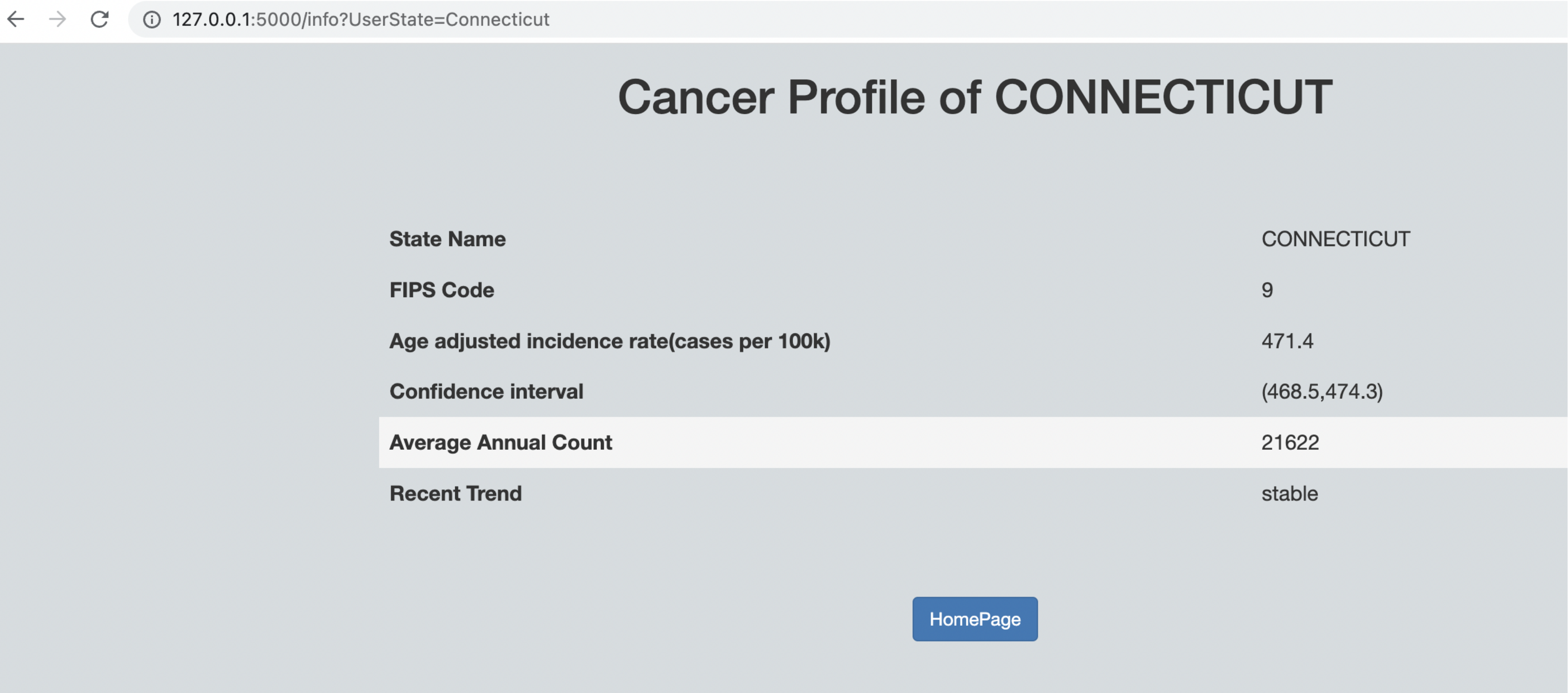
State Name	CONNECTICUT
FIPS Code	9
Age adjusted incidence rate(cases per 100k)	471.4
Confidence interval	(468.5,474.3)
Average Annual Count	21622
Recent Trend	stable

In [4]:
```
try:
    Highlight = Image.open("/Users/mahimakaur/Desktop/Computional Methods for Health Informatics/Highlight.png")
except IOError:
    pass
```

In [5]:

```
Highlight
```

Out[5]:



127.0.0.1:5000/info?UserState=Connecticut

Cancer Profile of CONNECTICUT

State Name	CONNECTICUT
FIPS Code	9
Age adjusted incidence rate(cases per 100k)	471.4
Confidence interval	(468.5,474.3)
Average Annual Count	21622
Recent Trend	stable

Answers

a. To get the cleaned dataset, I dropped the first 8 rows and the last 24 rows that do not contain relevant data. Then I removed the number and brackets in the each column State. I replaced (*) in Mississippi to Data Suppressed for clearly interpretation of data in the table. I also standardized the FIPS Codes. The cleaned dataset was saved into csv file cleaned_cancer_data.csv (data.to_csv('cleaned_cancer_data.csv', index= True))

b. I had changed the state names into lowercase. Therefore, when someone searches for a particular state, the capitalization won't impact the results. However, if the user puts an incorrect name or state not in the US or misspells the name the website will show can error. The user can go back from the error page by clicking the HomePage button (as shown in the ErrorPage image).

Advanced interactions:

- On the homepage I have included a type of autocomplete list of the US names i.e when the user starts to enter the state name, suggestions of the state will pop up. It was included to reduce any spelling errors from the user side and enhance the user experience. The same is shown in the image above eg. the user entered 'c' so all the US state name with 'c' will come and the user can select the state of interest.
- On the information page I have included more data. It also displays FIPS code, age adjusted incidence rate(cases per 100k), confidence interval, average annual count and recent trend in a tabular format. I removed the borders from the table and we can see the parameter highlighted as and when the user selected the particular parameter. This page also has the HomePage button.