Assignent 5

Question 1

Tahir Manuel D'Mello

In [1]:
```
import pandas as pd
import numpy as np
import random
import matplotlib.pyplot as plt
import seaborn as sns
import math
import itertools
import warnings

from statistics import mode


from sklearn import decomposition
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

Citation for dataset used:

CINAR, I. and KOKLU, M., (2019). “Classification of Rice Varieties Using Artificial Intelligence Methods.” International Journal of Intelligent Systems and Applications in Engineering, 7(3), 188-194.

DOI: https://doi.org/10.18201/ijisae.2019355381

UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Rice+%28Cammeo+and+Osmancik%29

In [2]:
```
data = pd.read_excel('Rice_Cammeo_Osmancik.xlsx')
```

In [3]:
```
data.head()
```

Out[3]:

	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

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

In [4]:
```
data_num = data.iloc[:,0:7]
standardized_data = (data_num - data_num.mean())/data_num.std()

standardized_data.describe()
#Standard deviation is 1 and mean is ~0
```

Out[4]:

	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

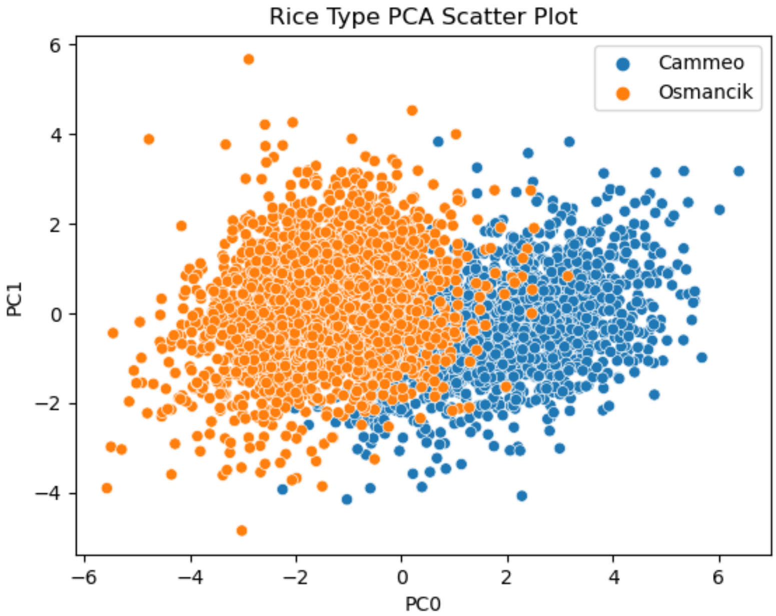
In [5]:
```
pca = decomposition.PCA(n_components=2)
data_reduced = pca.fit_transform(standardized_data)

pc0 = data_reduced[:, 0]
pc1 = data_reduced[:, 1]
```

In [6]:
```
data_reduced
```

Out[6]:
```
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]])
```

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

In [7]:
```
sns.scatterplot(pc0, pc1, hue = np.array(data.iloc[:,7]))
plt.xlabel('PC0')
plt.ylabel('PC1')
plt.title('Rice Type PCA Scatter Plot')
plt.show()
```



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)**

Two distinct clusters can be seen.
However, there is no clear boundary or separation between the two clusters.
The k-nearest neighbors method should be able to classify data in the approximate regions of (-6, -2) and (1, 6).
It might not be as effective when it comes to classifying data points in (-2, 1) as there is overlap between clusters there.

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)**

In [8]:
```
class QuadTree:
    def __init__(self, data, bounding_box=None, max_leaf_data=3):
        if bounding_box is None:
            xs, ys, conditions = zip(*data)
            self.xlo = min(xs)
            self.ylo = min(ys)
            self.xhi = max(xs)
            self.yhi = max(ys)
        else:
            self.xlo = bounding_box['xlo']
            self.xhi = bounding_box['xhi']
            self.ylo = bounding_box['ylo']
            self.yhi = bounding_box['yhi']

        if len(data) <= max_leaf_data:
            self._data = data
            self.children = []
        else:
            self._data = None
            self.children = []
            xsplit = (self.xlo + self.xhi) / 2
            ysplit = (self.ylo + self.yhi) / 2
            bbox = [
                {'xlo': self.xlo, 'xhi': xsplit, 'ylo': self.ylo, 'yhi': ysplit},
                {'xlo': self.xlo, 'xhi': xsplit, 'ylo': ysplit, 'yhi': self.yhi},
                {'xlo': xsplit, 'xhi': self.xhi, 'ylo': self.ylo, 'yhi': ysplit},
                {'xlo': xsplit, 'xhi': self.xhi, 'ylo': ysplit, 'yhi': self.yhi}
            ]
```

```python
            self.children = [
                QuadTree(get_data_in_range(data, my_bbox), my_bbox, max_leaf_data)
                for my_bbox in bbox
            ]


    def get_descendant_count(self):
        if not self.children:
            return len(self._data)
        else:
            return sum(child.get_descendant_count() for child in self.children)


    def contains(self, x, y):
        #I think this (xlo and ylo boundaries used) might help with the boundary issue but I have not verified
        #the code works beautifully and I ran out of time to check properly
        if( (self.xlo <= x) and (x < self.xhi) and (self.ylo <= y) and (y < self.yhi) ):
            return True
        return False


    def within_distance(self, d, x, y):
        #Cheatsheet Case 1
        if(self.contains(x, y)):
            return True

        #Cheatsheet Case 2
        if(self.xlo <= x <= self.xhi and (y < self.ylo or y > self.yhi) ):
            if(abs(y - self.ylo) < d or abs(y - self.yhi) < d):
                return True

        #Cheatsheet Case 3
        if(self.ylo <= y <= self.yhi and (x < self.xlo or x > self.xhi)):
            if(abs(x - self.xlo) < d or abs(x - self.xhi) < d):
                return True

        #Cheatsheet Case 4
        if min(math.dist([x, y], [self.xlo, self.ylo]),
               math.dist([x, y], [self.xlo, self.yhi]),
               math.dist([x, y], [self.xhi, self.yhi]),
               math.dist([x, y], [self.xhi, self.ylo])) < d:
            return True

        return False


    def leaves_within_distance(self, d, x, y):
        result = []

        if not self.children:
            if(self.within_distance(d, x, y) and self._data):
                return self._data
            else:
                return None

        for child in self.children:
            temp = child.leaves_within_distance(d,x,y)
            if temp: #Drops None return cases
                for item in temp:
                    result.append(item)

        return result


    def quadtree_diag(self):
        return math.dist( [self.xhi, self.yhi], [self.xlo, self.ylo] )


    def __repr__(self):
        return f'<QuadTree xlo={self.xlo} ylo={self.ylo} xhi={self.xhi} yhi={self.yhi} #desc={self.get_descendant_count()}>'
```

In [9]:

```python
def get_data_in_range(data, bbox):
    result = []
    for x, y, condition in data:
        if bbox['xlo'] <= x <= bbox['xhi'] and bbox['ylo'] <= y <= bbox['yhi']:
            result.append([x, y, condition])
    return result


def small_containing_quadtree(quad_in, k, x, y):
    parent = quad_in
    current_quad = quad_in

    while (current_quad.get_descendant_count() > k):
        for child in current_quad.children:
            #print(child)
            if child.contains(x,y):
                parent = current_quad
                current_quad = child

    if current_quad.get_descendant_count() < k:
```

```
        return parent

    return current_quad
```

In [10]:
```
def get_fake_data(N=1_000):
    data = []
    for _ in range(N):
        data.append([
            random.random(),
            random.random(),
            random.choice(["healthy", "sick"])])
    return data
```

In [11]:
```
def knn_quad(quad_in, k, x, y):

    smallest_quad = small_containing_quadtree(quad_in, k, x, y)

    radius = smallest_quad.quadtree_diag()

    first_radius = radius

    radius_points = pd.DataFrame( quad_in.leaves_within_distance(radius, x, y) )

    #To ensure capture of atleast k points, rarely used
    while len(radius_points) < k:
        radius = radius*2
        radius_points = pd.DataFrame( quad_in.leaves_within_distance(radius, x, y) )

    radius_points['distance'] = np.sqrt( np.square(radius_points[0] - x) + np.square(radius_points[1] - y))

    out_table = radius_points.sort_values(by=['distance']).reset_index(drop=True).iloc[0:20,:]

    result = mode(out_table[2]) #Picks most common tag
    #print(result)
    return out_table, result
```

Testing out the quad tree knn implementation.

In [12]:
```
fake_data = get_fake_data()
#fake_data
```

In [13]:
```
x = 0.5
y = 0.5
k = 20
quad_in = QuadTree(fake_data)
```

In [14]:
```
table, tag = knn_quad(quad_in, k, x, y)
print(tag)
table
```

```
sick
```

Out[14]:

	0	1	2	distance
0	0.521285	0.519752	healthy	0.029038
1	0.478134	0.524390	sick	0.032757
2	0.465740	0.502601	healthy	0.034359
3	0.538338	0.510178	sick	0.039666
4	0.527561	0.469578	sick	0.041050
5	0.479519	0.460662	sick	0.044350
6	0.544418	0.513268	healthy	0.046358
7	0.460375	0.474059	healthy	0.047361
8	0.463632	0.537944	sick	0.052559
9	0.553843	0.492209	healthy	0.054404
10	0.553363	0.485942	sick	0.055184
11	0.446860	0.517334	sick	0.055896
12	0.519599	0.443671	healthy	0.059641
13	0.509805	0.565452	healthy	0.066182
14	0.560723	0.472156	sick	0.066802
15	0.545454	0.450188	healthy	0.067433
16	0.444173	0.545361	healthy	0.071932
17	0.507673	0.427893	sick	0.072514
18	0.425924	0.519746	sick	0.076662
19	0.423048	0.491528	sick	0.077417

In [15]:
```
fake_data_test = pd.DataFrame(fake_data)

fake_data_test['distance'] = np.sqrt( np.square(fake_data_test[0] - x) + np.square(fake_data_test[1] - y))
fake_data_test = fake_data_test.sort_values(by=['distance']).reset_index(drop=True).iloc[0:20,:]

print(mode(fake_data_test[2]))
fake_data_test
```

```
sick
```

Out[15]:

	0	1	2	distance
0	0.521285	0.519752	healthy	0.029038
1	0.478134	0.524390	sick	0.032757
2	0.465740	0.502601	healthy	0.034359
3	0.538338	0.510178	sick	0.039666
4	0.527561	0.469578	sick	0.041050
5	0.479519	0.460662	sick	0.044350
6	0.544418	0.513268	healthy	0.046358
7	0.460375	0.474059	healthy	0.047361
8	0.463632	0.537944	sick	0.052559
9	0.553843	0.492209	healthy	0.054404
10	0.553363	0.485942	sick	0.055184
11	0.446860	0.517334	sick	0.055896
12	0.519599	0.443671	healthy	0.059641
13	0.509805	0.565452	healthy	0.066182
14	0.560723	0.472156	sick	0.066802
15	0.545454	0.450188	healthy	0.067433
16	0.444173	0.545361	healthy	0.071932
17	0.507673	0.427893	sick	0.072514
18	0.425924	0.519746	sick	0.076662
19	0.423048	0.491528	sick	0.077417

As seen above, both the manual and quad tree-knn returned the same predicion label and 20 closest neighbours

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)**

In [16]:
```
working_data = data.sample(frac=1, random_state=42).reset_index(drop=True) #shuffle data because of incoming Class sorting
working_data
```

Out[16]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent	Class
0	12442	459.535004	187.508850	87.187302	0.885323	12941	0.587580	Cammeo
1	12408	437.014008	179.741165	88.829605	0.869343	12598	0.636928	Osmancik
2	12867	449.079987	181.700562	91.341064	0.864460	13152	0.649062	Osmancik
3	13090	472.945007	202.601578	83.230179	0.911722	13331	0.775290	Cammeo
4	10359	409.510986	173.337967	76.875809	0.896273	10510	0.573588	Osmancik
...	...	...	...	...	...	...	...	...
3805	16625	535.989014	229.793594	93.089622	0.914272	16951	0.654141	Cammeo
3806	13901	478.848999	200.441910	89.341988	0.895170	14232	0.568548	Cammeo
3807	16291	523.192993	223.252335	93.604156	0.907859	16595	0.581157	Cammeo
3808	10847	417.924011	170.366791	82.473007	0.875018	11107	0.746319	Osmancik
3809	13154	451.562012	179.953598	94.313812	0.851656	13428	0.650222	Osmancik

3810 rows × 8 columns

In [17]:
```
train_data = working_data[:int((len(working_data)+1)*.85)] #85% train
test_data = working_data[int((len(working_data)+1)*.85):].reset_index(drop=True) #15% test

print(len(train_data), len(test_data))
train_data
```

```
3239 571
```

Out[17]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent	Class
0	12442	459.535004	187.508850	87.187302	0.885323	12941	0.587580	Cammeo
1	12408	437.014008	179.741165	88.829605	0.869343	12598	0.636928	Osmancik
2	12867	449.079987	181.700562	91.341064	0.864460	13152	0.649062	Osmancik
3	13090	472.945007	202.601578	83.230179	0.911722	13331	0.775290	Cammeo
4	10359	409.510986	173.337967	76.875809	0.896273	10510	0.573588	Osmancik
...	...	...	...	...	...	...	...	...
3234	12312	447.717010	178.960739	88.517525	0.869109	12593	0.618321	Osmancik
3235	15568	519.455994	222.228088	90.817528	0.912683	15984	0.782705	Cammeo
3236	11183	414.348999	170.649384	83.881950	0.870852	11321	0.616993	Osmancik
3237	14693	484.562988	199.619995	94.807610	0.880018	15034	0.609163	Cammeo
3238	13057	445.177002	177.807816	94.507072	0.847050	13348	0.643360	Osmancik

3239 rows × 8 columns

In [18]:
```
test_data
```

Out[18]:

	Area	Perimeter	Major_Axis_Length	Minor_Axis_Length	Eccentricity	Convex_Area	Extent	Class
0	12339	435.937012	173.027420	92.289864	0.845874	12646	0.737493	Osmancik
1	10286	418.289001	175.148590	75.853813	0.901354	10548	0.641592	Osmancik
2	12767	463.842010	196.193039	83.973564	0.903772	13104	0.606220	Cammeo
3	12016	443.222992	189.281067	81.122215	0.903503	12186	0.746567	Osmancik
4	15080	500.006012	209.589279	92.796661	0.896643	15614	0.732323	Cammeo
...	...	...	...	...	...	...	...	...
566	16625	535.989014	229.793594	93.089622	0.914272	16951	0.654141	Cammeo
567	13901	478.848999	200.441910	89.341988	0.895170	14232	0.568548	Cammeo
568	16291	523.192993	223.252335	93.604156	0.907859	16595	0.581157	Cammeo
569	10847	417.924011	170.366791	82.473007	0.875018	11107	0.746319	Osmancik
570	13154	451.562012	179.953598	94.313812	0.851656	13428	0.650222	Osmancik

571 rows × 8 columns

In [19]:
```
train_data_raw = train_data.iloc[:,0:7]
test_data_raw = test_data.iloc[:,0:7]

scaler = StandardScaler()
st_train_data = scaler.fit_transform( train_data_raw )
st_test_data = scaler.transform( test_data_raw )

pca = decomposition.PCA(n_components=2)
train_data_reduced = pca.fit_transform( st_train_data )
test_data_reduced = pca.transform( st_test_data )
```

In [20]:
```
train_data_in = []
count = 0

for item in train_data_reduced:
    train_data_in.append([ item[0], item[1], train_data['Class'][count] ])
    count = count + 1

#train_data_in
```

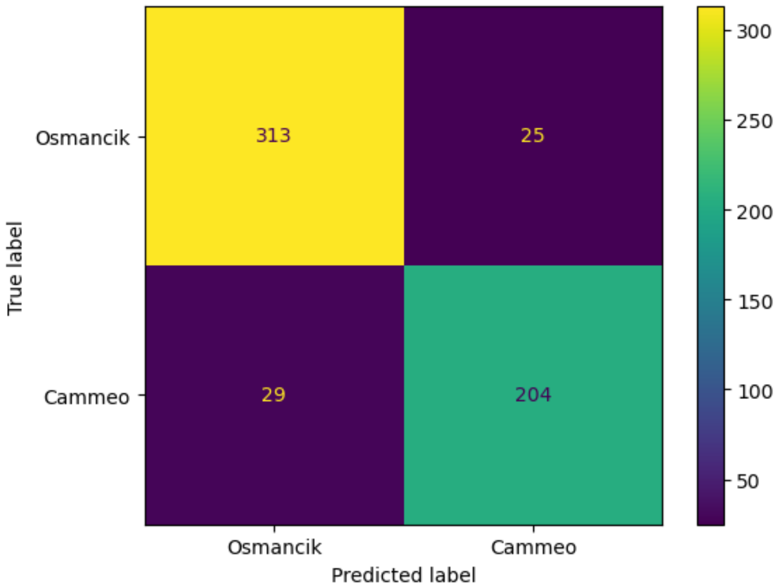
In [21]:
```
test_data_in = []
count = 0

for item in test_data_reduced:
    test_data_in.append([ item[0], item[1], test_data['Class'][count] ])
    count = count + 1

#test_data_in
```

In [22]:
```
k = 1
k1_results = []
count = 0
quad_in = QuadTree(train_data_in, max_leaf_data = 1)

for item in test_data_in:
    #print(count)
    count = count + 1
    x = item[0]
    y = item[1]
```

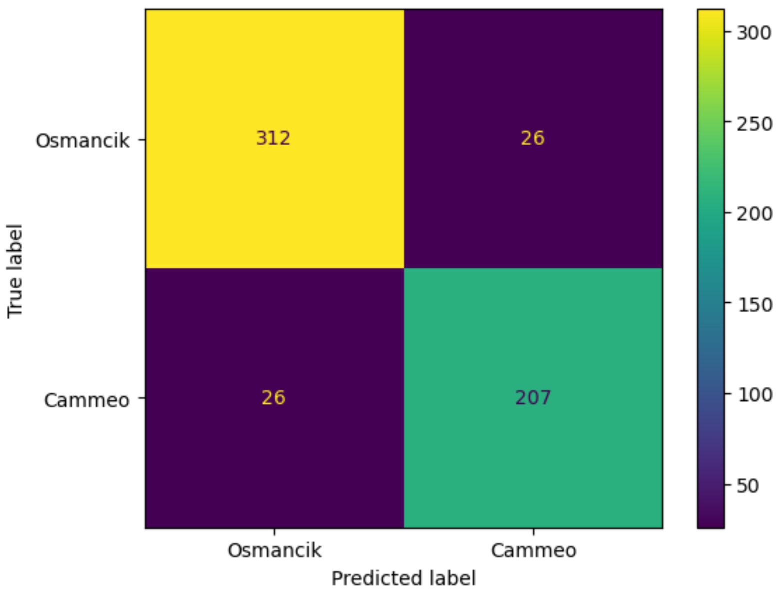
```
    table, tag = knn_quad(quad_in, k, x, y)

    k1_results.append(tag)
```

In [23]:
```
k = 5
k5_results = []
count = 0
quad_in = QuadTree(train_data_in, max_leaf_data = 1)

for item in test_data_in:

    x = item[0]
    y = item[1]
    table, tag = knn_quad(quad_in, k, x, y)

    k5_results.append(tag)
```

In [24]:
```
confusion_matrix_1 = confusion_matrix(test_data['Class'], k1_results,  labels=test_data['Class'].unique())
disp_1 = ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_1,  display_labels = test_data['Class'].unique())
disp_1.plot()
plt.show()
```



In [26]:
```
confusion_matrix_5 = confusion_matrix(test_data['Class'], k5_results,  labels=test_data['Class'].unique())
disp_5 = ConfusionMatrixDisplay(confusion_matrix = confusion_matrix_5,  display_labels = test_data['Class'].unique())
disp_5.plot()
plt.show()
```



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

This confusion matrix provides information about how many labels were predicted correctly and vice versa.
Precision checks for how many values predicted are actually what they are - Precision = TP/(TP+FP).
Recall checks for how many true values the model was able to choose correctly from all true values present - Recall = TP/ (TP + FN).

For k = 1:

313 'Osmancik' labels were actually 'Osmancik' and 204 'Cammeo' were actually 'Cammeo'.

Precision for 'Osmancik' = 313/(313+29) = 0.915 and Precision for 'Cammeo' = 204/(204 + 25) = 0.890.

Recall for 'Osmancik' = 313/(313+25) = 0.926 and Precision for 'Cammeo' = 204/(204 + 29) = 0.875.

For k = 5:

312 'Osmancik' labels were actually 'Osmancik' and 207 'Cammeo' were actually 'Cammeo'.

Precision for 'Osmancik' = 312/(312+26) = 0.923 and Precision for 'Cammeo' = 207/(207 + 26) = 0.888.

Recall for 'Osmancik' = 312/(312+26) = 0.923 and Precision for 'Cammeo' = 2047/(207 + 26) = 0.888.

Assignent 5

Question 2

Tahir Manuel D'Mello

What's your data? **(4 points)**

My data is from is a dataset titled 'FIFA 22 complete player dataset' compiled by STEFANO LEONE on Kaggle. https://www.kaggle.com/datasets/stefanoleone992/fifa-22-complete-player-dataset?select=players_22.csv

It contains data about all soccer/football players on the FIFA 2022 Football game published by EA Sports FC. The data was scraped from the publicly available website sofifa.com by Stefano Leone and uploaded to Kaggle.

It contains 110 columns of information about 19239 professional players.
It consists of player statistics and attributes in various areas - physical, attacking, defending, psychological, etc.
It also has information about player nationality, position, age, club status and monetary contracts.

What analyses do you want to run and why are they interesting? **(4 points)**

The overall goal of my project is to buld a decision tool that will allow someone with no football knowledge to assess the player quality and financial (wage) status of a team. The analyses will be done on a user-selected team.

I am going to perform two main analyses for this:

```
1. Principal component analysis on player stats and attributes
followed by k-means clustering. This will allow me to visually
demonstrate which players are similar in playing profile without
using any football-specific attributes.
2. A Random Forest Regressor will be trained to predict the wages
from playing attributes of all players on the dataset except in
the team chosen. Then, the regressor is used to predict the wages
of the players on the team to determine if they are under/over
paid.
```

Which ones will be interactive, where the user can provide a parameter? **(4 points)**
What graphs will you make? **(4 points)**

The user provides the team name as a parameter at the start for both analyses.

Both analyses will create graphs that will be interactive.

The first one will be a 3D plot that the user can move around to look at the data. There might even be hover information for each player. The user can specify how many clusters (k)

to make of the data.
The second one will be a hover scatter plot of predicted vs actual wages. Each point will hover to show information about the player.

Describe how you want your website to work. **(4 points)**

The landing page will have a user-entry field.
The user will choose what team the analyses will be carried out on.

Then maybe, basic EDA summary statistics of the team will be displayed.

The user will be able to scroll (or maybe click) to the PCA-KNN graph and manipulate that as desired.
Then he/she will similarly be able to move to the wage scatter plot.

What do you see as your biggest challenge to completing this, and how do you expect to overcome this challenge? **(5 points)**

I have actually already completed my major analyses and generated all my graphs.
I do want to still add additional features to my graphs if I can.

I need to still build my website. This is the main challenge left for me.
I have not worked a lot with website building tools beyond this assignemnt.

My plan is to start with a basic framework and build my way up from there until I have a full website with all the features I need.

Assignent 5

Question 3

Tahir Manuel D'Mello

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

Perform any necessary data cleaning . 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)**

All data cleaning steps and changes to the csv have been descibed step-by-step in this notebook.

Summary:

```
1. Dropped all metadata above and below table.
2. Replaced all non-standard missing values with standard Python
NaN.
3. Remove number reference of state in each state name string.
```

CITATION:
This data was from the National Cancer Institute.
It is the 'Latest 5-year average' numbers 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.

Link: https://statecancerprofiles.cancer.gov/incidencerates/index.php

1 Source: National Program of Cancer Registries [https://www.cdc.gov/cancer/npcr/index.htm] and Surveillance, Epidemiology, and End Results [http://seer.cancer.gov] SEER*Stat Database (2001-2019) - United States Department of Health and Human Services, Centers for Disease Control and Prevention and National Cancer Institute. Based on the 2021 submission.*
*6 Source: National Program of Cancer Registries SEER*Stat Database (2001-2019) - United States Department of Health and Human Services, Centers for Disease Control and Prevention (based on the 2021 submission).[https://www.cdc.gov/cancer/npcr/index.htm]
7 Source: SEER November 2021 submission.
8 Source: Incidence data provided by the SEER Program. (http://seer.cancer.gov) AAPCs are calculated by the Joinpoint Regression Program (https://surveillance.cancer.gov/joinpoint/) and are based on APCs. Data are age-adjusted to the 2000 US standard population (http://www.seer.cancer.gov/stdpopulations/single_age.html) (19 age groups: <1, 1-4, 5-9, ... , 80-84,85+). Rates are for invasive cancer only (except for bladder cancer which is invasive and in situ) or unless otherwise specified. Population counts for denominators are based on Census populations as modifed by NCI. The US Population Data (http://seer.cancer.gov/popdata/) File is used with SEER November 2021 data.

In [2]:
```
#Drop metadata above the needed table by choosing header at entry row 5.
data = pd.read_csv('incd.csv', header = 4)
data
```

Out[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*Ran
0	US (SEER+NPCR)(1)	0.0	449.4	449.1	449.7	N/A	N,
1	Kentucky(7)	21000.0	516	513.2	518.8	1	
2	Iowa(7)	19000.0	490.7	487.5	494	2	
3	New Jersey(7)	34000.0	488.9	487	490.8	3	
4	West Virginia(6)	54000.0	487.4	483.3	491.4	4	
...	...	...	...	...	...	...	
69	8 Source: Incidence data provided by the SEER ...	NaN	NaN	NaN	NaN	NaN	Na
70	Interpret Rankings provides insight into inter...	NaN	NaN	NaN	NaN	NaN	Na
71	Data not available [http://statecancerprofiles...	NaN	NaN	NaN	NaN	NaN	Na
72	Data for the United States does not include da...	NaN	NaN	NaN	NaN	NaN	Na
73	Data for the United States does not include Pu...	NaN	NaN	NaN	NaN	NaN	Na

74 rows × 13 columns

In [3]:

```
#Drop all extra metadata rows from below by dropping NA values in FIPS column
data = data[data[' FIPS'].notna()]
data
```

Out[3]:

	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)	Uppe (CI*Ra
0	US (SEER+NPCR) (1)	0.0	449.4	449.1	449.7	N/A	N/A	
1	Kentucky(7)	21000.0	516	513.2	518.8	1	1	
2	Iowa(7)	19000.0	490.7	487.5	494	2	2	
3	New Jersey(7)	34000.0	488.9	487	490.8	3	2	
4	West Virginia(6)	54000.0	487.4	483.3	491.4	4	2	
5	New York(7)	36000.0	484.8	483.6	486.1	5	4	
6	Louisiana(7)	22000.0	484.3	481.7	487	6	3	
7	Arkansas(6)	5000.0	483.6	480.4	486.9	7	3	
8	New Hampshire(6)	33000.0	482.9	478.2	487.7	8	2	
9	Pennsylvania(6)	42000.0	476.8	475.3	478.3	9	9	
10	Maine(6)	23000.0	476.7	472.2	481.3	10	7	
11	Rhode Island(6)	44000.0	476.2	470.8	481.6	11	7	
12	Mississippi(6)	28000.0	476	472.7	479.3	12	8	
13	Delaware(6)	10000.0	474.7	469.1	480.3	13	7	
14	Minnesota(6)	27000.0	471.5	469.2	474	14	11	
15	Ohio(6)	39000.0	471.5	469.8	473.1	15	12	
16	Connecticut(7)	9000.0	471.4	468.5	474.3	16	11	
17	Wisconsin(6)	55000.0	470.8	468.5	473.1	17	12	
18	North Carolina(6)	37000.0	469.9	468.2	471.7	18	13	
19	Nebraska(6)	31000.0	469.7	465.6	473.8	19	11	
20	Georgia(7)	13000.0	468.6	466.8	470.4	20	15	
21	Tennessee(6)	47000.0	466.5	464.4	468.7	21	18	
22	Montana(6)	30000.0	466.3	461	471.7	22	13	
23	Illinois(7)	17000.0	465.2	463.6	466.8	23	20	
24	Florida(6)	12000.0	460.5	459.4	461.6	24	23	
25	Kansas(6)	20000.0	459.4	456.1	462.7	25	23	
26	Vermont(6)	50000.0	457	450.3	463.8	26	21	
27	Indiana(6)	18000.0	456.8	454.6	458.9	27	25	
28	Massachusetts(7)	25000.0	454.8	452.7	456.8	28	26	
29	North Dakota(6)	38000.0	454.4	447.8	461.1	29	23	
30	Maryland(6)	24000.0	454.1	451.9	456.4	30	26	

	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)	Uppe (CI*Ra
31	Missouri(6)	29000.0	453.2	451	455.4	31	27	
32	South Dakota(6)	46000.0	452.3	446.4	458.3	32	24	
33	Alabama(6)	1000.0	451.7	449.3	454.2	33	28	
34	Oklahoma(6)	40000.0	450.8	448	453.6	34	28	
35	Idaho(7)	16000.0	448.5	444.2	452.8	35	28	
36	Michigan(6)	26000.0	446.7	445	448.4	36	34	
37	South Carolina(6)	45000.0	443.8	441.4	446.2	37	35	
38	Washington(1)	53000.0	441.3	439.2	443.3	38	37	
39	Oregon(6)	41000.0	428.4	425.8	431	39	39	
40	Alaska(6)	2900.0	417	410	424.1	40	40	
41	District of Columbia(6)	11001.0	416.9	410	424	41	40	
42	Hawaii(7)	15000.0	416.8	412.5	421.2	42	40	
43	Texas(7)	48000.0	415.3	414.3	416.4	43	40	
44	Virginia(6)	51000.0	409.4	407.6	411.2	44	43	
45	Utah(7)	49000.0	407.2	403.8	410.6	45	43	
46	Wyoming(6)	56000.0	405.7	398.9	412.7	46	42	
47	California(7)	6000.0	402.4	401.6	403.3	47	46	
48	Colorado(6)	8000.0	396.4	394.1	398.6	48	47	
49	Arizona(6)	4000.0	382.4	380.6	384.3	49	49	
50	New Mexico(7)	35000.0	374	370.6	377.5	50	50	
51	Puerto Rico(6)	72001.0	368.2	365.4	370.9	N/A	N/A	
52	Nevada(6)	32000.0	data not	data not	data not	N/A	N/A	

In [4]:

```
#Replace all non-standard missing values with standard Python NaN

data = data.replace(['data not available', 'data not available ', ' data not avail
                    np.nan)
data
```

Out[4]:

	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)	Uppe (CI*Ra
0	US (SEER+NPCR) (1)	0.0	449.4	449.1	449.7	NaN	NaN	
1	Kentucky(7)	21000.0	516	513.2	518.8	1	1	
2	Iowa(7)	19000.0	490.7	487.5	494	2	2	
3	New Jersey(7)	34000.0	488.9	487	490.8	3	2	
4	West Virginia(6)	54000.0	487.4	483.3	491.4	4	2	
5	New York(7)	36000.0	484.8	483.6	486.1	5	4	
6	Louisiana(7)	22000.0	484.3	481.7	487	6	3	
7	Arkansas(6)	5000.0	483.6	480.4	486.9	7	3	
8	New Hampshire(6)	33000.0	482.9	478.2	487.7	8	2	
9	Pennsylvania(6)	42000.0	476.8	475.3	478.3	9	9	
10	Maine(6)	23000.0	476.7	472.2	481.3	10	7	
11	Rhode Island(6)	44000.0	476.2	470.8	481.6	11	7	
12	Mississippi(6)	28000.0	476	472.7	479.3	12	8	
13	Delaware(6)	10000.0	474.7	469.1	480.3	13	7	
14	Minnesota(6)	27000.0	471.5	469.2	474	14	11	
15	Ohio(6)	39000.0	471.5	469.8	473.1	15	12	
16	Connecticut(7)	9000.0	471.4	468.5	474.3	16	11	
17	Wisconsin(6)	55000.0	470.8	468.5	473.1	17	12	
18	North Carolina(6)	37000.0	469.9	468.2	471.7	18	13	
19	Nebraska(6)	31000.0	469.7	465.6	473.8	19	11	
20	Georgia(7)	13000.0	468.6	466.8	470.4	20	15	
21	Tennessee(6)	47000.0	466.5	464.4	468.7	21	18	
22	Montana(6)	30000.0	466.3	461	471.7	22	13	
23	Illinois(7)	17000.0	465.2	463.6	466.8	23	20	
24	Florida(6)	12000.0	460.5	459.4	461.6	24	23	
25	Kansas(6)	20000.0	459.4	456.1	462.7	25	23	
26	Vermont(6)	50000.0	457	450.3	463.8	26	21	
27	Indiana(6)	18000.0	456.8	454.6	458.9	27	25	
28	Massachusetts(7)	25000.0	454.8	452.7	456.8	28	26	
29	North Dakota(6)	38000.0	454.4	447.8	461.1	29	23	
30	Maryland(6)	24000.0	454.1	451.9	456.4	30	26	

	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)	Uppe (CI*Ra
31	Missouri(6)	29000.0	453.2	451	455.4	31	27	
32	South Dakota(6)	46000.0	452.3	446.4	458.3	32	24	
33	Alabama(6)	1000.0	451.7	449.3	454.2	33	28	
34	Oklahoma(6)	40000.0	450.8	448	453.6	34	28	
35	Idaho(7)	16000.0	448.5	444.2	452.8	35	28	
36	Michigan(6)	26000.0	446.7	445	448.4	36	34	
37	South Carolina(6)	45000.0	443.8	441.4	446.2	37	35	
38	Washington(1)	53000.0	441.3	439.2	443.3	38	37	
39	Oregon(6)	41000.0	428.4	425.8	431	39	39	
40	Alaska(6)	2900.0	417	410	424.1	40	40	
41	District of Columbia(6)	11001.0	416.9	410	424	41	40	
42	Hawaii(7)	15000.0	416.8	412.5	421.2	42	40	
43	Texas(7)	48000.0	415.3	414.3	416.4	43	40	
44	Virginia(6)	51000.0	409.4	407.6	411.2	44	43	
45	Utah(7)	49000.0	407.2	403.8	410.6	45	43	
46	Wyoming(6)	56000.0	405.7	398.9	412.7	46	42	
47	California(7)	6000.0	402.4	401.6	403.3	47	46	
48	Colorado(6)	8000.0	396.4	394.1	398.6	48	47	
49	Arizona(6)	4000.0	382.4	380.6	384.3	49	49	
50	New Mexico(7)	35000.0	374	370.6	377.5	50	50	
51	Puerto Rico(6)	72001.0	368.2	365.4	370.9	NaN	NaN	
52	Nevada(6)	32000.0	NaN	NaN	NaN	NaN	NaN	

In [5]:

```
#Remove number tag of state in each state name in the State column

data['State'] = data['State'].str[:-3]
data.to_csv('incd_cleaned.csv', index=False)
data
```

Out[5]:

	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*Ranl
0	US (SEER+NPCR)	0.0	449.4	449.1	449.7	NaN	NaN	Na
1	Kentucky	21000.0	516	513.2	518.8	1	1	
2	Iowa	19000.0	490.7	487.5	494	2	2	
3	New Jersey	34000.0	488.9	487	490.8	3	2	
4	West Virginia	54000.0	487.4	483.3	491.4	4	2	
5	New York	36000.0	484.8	483.6	486.1	5	4	
6	Louisiana	22000.0	484.3	481.7	487	6	3	
7	Arkansas	5000.0	483.6	480.4	486.9	7	3	
8	New Hampshire	33000.0	482.9	478.2	487.7	8	2	1
9	Pennsylvania	42000.0	476.8	475.3	478.3	9	9	1
10	Maine	23000.0	476.7	472.2	481.3	10	7	1
11	Rhode Island	44000.0	476.2	470.8	481.6	11	7	2
12	Mississippi	28000.0	476	472.7	479.3	12	8	1
13	Delaware	10000.0	474.7	469.1	480.3	13	7	2
14	Minnesota	27000.0	471.5	469.2	474	14	11	2
15	Ohio	39000.0	471.5	469.8	473.1	15	12	2
16	Connecticut	9000.0	471.4	468.5	474.3	16	11	2
17	Wisconsin	55000.0	470.8	468.5	473.1	17	12	2
18	North Carolina	37000.0	469.9	468.2	471.7	18	13	2
19	Nebraska	31000.0	469.7	465.6	473.8	19	11	2
20	Georgia	13000.0	468.6	466.8	470.4	20	15	2
21	Tennessee	47000.0	466.5	464.4	468.7	21	18	2
22	Montana	30000.0	466.3	461	471.7	22	13	2
23	Illinois	17000.0	465.2	463.6	466.8	23	20	2
24	Florida	12000.0	460.5	459.4	461.6	24	23	2
25	Kansas	20000.0	459.4	456.1	462.7	25	23	3
26	Vermont	50000.0	457	450.3	463.8	26	21	3
27	Indiana	18000.0	456.8	454.6	458.9	27	25	3
28	Massachusetts	25000.0	454.8	452.7	456.8	28	26	3
29	North Dakota	38000.0	454.4	447.8	461.1	29	23	3
30	Maryland	24000.0	454.1	451.9	456.4	30	26	3

	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*Ranl
31	Missouri	29000.0	453.2	451	455.4	31	27	3
32	South Dakota	46000.0	452.3	446.4	458.3	32	24	3
33	Alabama	1000.0	451.7	449.3	454.2	33	28	3
34	Oklahoma	40000.0	450.8	448	453.6	34	28	3
35	Idaho	16000.0	448.5	444.2	452.8	35	28	3
36	Michigan	26000.0	446.7	445	448.4	36	34	3
37	South Carolina	45000.0	443.8	441.4	446.2	37	35	3
38	Washington	53000.0	441.3	439.2	443.3	38	37	3
39	Oregon	41000.0	428.4	425.8	431	39	39	4
40	Alaska	2900.0	417	410	424.1	40	40	4
41	District of Columbia	11001.0	416.9	410	424	41	40	4
42	Hawaii	15000.0	416.8	412.5	421.2	42	40	4
43	Texas	48000.0	415.3	414.3	416.4	43	40	4
44	Virginia	51000.0	409.4	407.6	411.2	44	43	4
45	Utah	49000.0	407.2	403.8	410.6	45	43	4
46	Wyoming	56000.0	405.7	398.9	412.7	46	42	4
47	California	6000.0	402.4	401.6	403.3	47	46	4
48	Colorado	8000.0	396.4	394.1	398.6	48	47	4
49	Arizona	4000.0	382.4	380.6	384.3	49	49	4
50	New Mexico	35000.0	374	370.6	377.5	50	50	5
51	Puerto Rico	72001.0	368.2	365.4	370.9	NaN	NaN	Na
52	Nevada	32000.0	NaN	NaN	NaN	NaN	NaN	Na

Assignment 5

Question 3

Tahir D'Mello

A5Q3.py

```
from flask import Flask, render_template, request, jsonify
import pandas as pd
import json
import plotly
import plotly.express as px

data = pd.read_csv('incd_cleaned.csv')

code = {'Alabama': 'AL',
        'Alaska': 'AK',
        'Arizona': 'AZ',
        'Arkansas': 'AR',
        'California': 'CA',
        'Colorado': 'CO',
        'Connecticut': 'CT',
        'Delaware': 'DE',
        'District of Columbia': 'DC',
        'Florida': 'FL',
        'Georgia': 'GA',
        'Hawaii': 'HI',
        'Idaho': 'ID',
        'Illinois': 'IL',
        'Indiana': 'IN',
        'Iowa': 'IA',
        'Kansas': 'KS',
        'Kentucky': 'KY',
        'Louisiana': 'LA',
        'Maine': 'ME',
        'Maryland': 'MD',
        'Massachusetts': 'MA',
        'Michigan': 'MI',
        'Minnesota': 'MN',
        'Mississippi': 'MS',
        'Missouri': 'MO',
        'Montana': 'MT',
        'Nebraska': 'NE',
        'Nevada': 'NV',
        'New Hampshire': 'NH',
        'New Jersey': 'NJ',
        'New Mexico': 'NM',
        'New York': 'NY',
        'North Carolina': 'NC',
```

```
        'North Dakota': 'ND',
        'Ohio': 'OH',
        'Oklahoma': 'OK',
        'Oregon': 'OR',
        'Pennsylvania': 'PA',
        'Rhode Island': 'RI',
        'South Carolina': 'SC',
        'South Dakota': 'SD',
        'Tennessee': 'TN',
        'Texas': 'TX',
        'Utah': 'UT',
        'Vermont': 'VT',
        'Virginia': 'VA',
        'Washington': 'WA',
        'West Virginia': 'WV',
        'Wisconsin': 'WI',
        'Wyoming': 'WY'}

data['Code'] = data['State'].map(code)

data.rename(columns={'Age-Adjusted Incidence Rate([rate note]) - cases per 100,000':'Cases per 100k'},
           inplace=True)

app = Flask(__name__)

#Using Flask, implement the 3 routes (15 points total):

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


@app.route('/state/<string:name>')
def returnJSON(name):
  usertext = name

  if sum(data['State'].str.lower() == usertext.lower()) == True:

    result = data[data['State'].str.lower() == usertext.lower()].iloc[0,2]

    data_json = {"State" : usertext, "Cases" : result}

  else:
    data_json = "ERROR - Check spelling or spaces"


  return jsonify(data_json)
```

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

   if sum(data['State'].str.lower() == usertext.lower()) == True:
      result = data[data['State'].str.lower() == usertext.lower()].iloc[0,2]
   else:
      result = "ERROR  - Check spelling or spaces"

   return render_template("info.html", analysis=result, usertext=usertext)


#Extra work (5 points)
@app.route("/map")
def map():
   fig = px.choropleth(data,
                  locations='Code',
                  color='Cases per 100k',
                  color_continuous_scale='spectral_r',
                  hover_name='State',
                  locationmode='USA-states',
                  scope='usa')

   graphJSON = json.dumps(fig, cls=plotly.utils.PlotlyJSONEncoder)

   return render_template("map.html", graphJSON=graphJSON)


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

index.html

```
<html>
  <body>
                <h1>US State Cancer Statistics</h1>
      Type a name of a state here to retrieve the respective state cancer incidince rate per 100k
people:<br>
      <form action="info" method="GET">
        <textarea style="width:50%; height: 5em" name="usertext"></textarea>
        <br>
        <input type="submit" value="Cases">
      </form>
                OR <br> <br>
                <a href="{{ url_for('map') }}">Visualize state cancer incidince rate per 100k people
for all states on interactive map</a>
  </body>
</html>
```

info.html

```
<html>
  <body>
      The state you entered:
      <pre style="background-color: lightgray; margin-left: 5em">{{ usertext }}</pre>
      Cancer cases per 100k population are:
      <pre style="background-color: lightgray; margin-left: 5em">{{ analysis }}</pre>
                <a href="{{ url_for('index') }}">Go back</a>
  </body>
</html>
```

map.html

```
<html>
        <body>
                <h1>Cases per 100k Population for all US States</h1>
                <div id='chart' class='chart'”></div>
        </body>

        <script src='https://cdn.plot.ly/plotly-latest.min.js'></script>
        <script type='text/javascript'>
          var graphs = {{graphJSON | safe}};
          Plotly.plot('chart',graphs,{});
        </script>
        <a href="{{ url_for('index') }}">Go back</a>
</html>
```