# Conrad Ibanez

# Project Title- Baseball Performance Data and Relationships to Hall of Fame Selection

# Dataset- Kaggle-https://www.kaggle.com/open-source-sports/baseball-databank.

# This dataset is a collection of historical baseball data from 1871 to 2015. It contains 20 files, and the main tables include a master table for player and biographical info, batting statistics, pitching statistics, and fielding statistics. There are also tables that contain information for player salaries, awards, All-Star appearances, and Hall of Fame votes.

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.metrics import confusion\_matrix

import seaborn as sns

from scipy.spatial.distance import cdist

from scipy.spatial.distance import pdist

from sklearn.metrics import pairwise\_distances

from sklearn.cluster import KMeans

import scipy as sp

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

import plotly.graph\_objs as go

from plotly.offline import plot

def readTables():

'''

For this project, I am interested primarily on offensive metrics.

Data will be taken from Master.csv containing player data, Batting.csv containing offensive stats, and HallOfFame.csv indicating players in the Hall of Fame (HoF).

'''

try:

# Attempt to process the data from csv files.

master\_baseball\_df = pd.read\_csv("baseball/Master.csv")

master\_batting\_df = pd.read\_csv("baseball/Batting.csv")

master\_hall\_of\_fame\_df = pd.read\_csv("baseball/HallOfFame.csv")

except Exception as e:

print('Error has occurred processing file:',e)

player\_df = master\_baseball\_df[['playerID','nameFirst','nameLast']]

batting\_df = master\_batting\_df[['playerID','G','AB','R','H','2B','3B','HR','RBI']]

# Every entry should have the playerID but all other metrics can be filled with 0 since they are numbers

batting\_df = batting\_df.fillna(0)

# Each batting row corresponds to a year/season for a player

# Player can have many batting rows for each year/season played

# Need to group batting data by playerID and sum the data to get career totals to analyze for Hall of Fame (HoF) criteria

batting\_df\_groups = batting\_df.groupby('playerID',as\_index=False)

sum\_batting\_df = batting\_df\_groups.sum()

# Calculate the number of years/seasons played

count\_batting\_df = batting\_df\_groups.count()

count\_batting\_df = count\_batting\_df[['playerID','G']]

count\_batting\_df = count\_batting\_df.rename(columns={"G":"years"})

# Get the player data together

hof\_batting\_df = pd.merge(count\_batting\_df,sum\_batting\_df, on=['playerID'],how='outer')

# Get the Hall of Fame (HoF) data together

hall\_of\_fame\_df = master\_hall\_of\_fame\_df[['playerID','yearid','inducted','category']]

hall\_of\_fame\_df = hall\_of\_fame\_df[(hall\_of\_fame\_df['inducted']=='Y') & (hall\_of\_fame\_df['category']=='Player')]

# Combine player data and Hall of Fame data together to have one dataframe for analysis

# Handle missing data, clean up, and format data as necesary

data\_analysis\_df = pd.merge(player\_df,hof\_batting\_df, on=['playerID'],how='right')

data\_analysis\_df = pd.merge(data\_analysis\_df,hall\_of\_fame\_df, on=['playerID'],how='left')

data\_analysis\_df['yearid'] = pd.to\_datetime(data\_analysis\_df['yearid'], format='%Y')

hof\_fill\_values = {'yearid':pd.NaT, 'inducted': 'N', 'category': 'none'}

data\_analysis\_df = data\_analysis\_df.fillna(value=hof\_fill\_values)

data\_analysis\_df['inducted\_val'] = [1 if x =='Y' else 0 for x in data\_analysis\_df['inducted']]

#return the main data frame for analysis

return data\_analysis\_df

def createStatisticsHeatmap(data\_analysis\_df):

'''

Creating statistics heatmap to show relationships between offensive metrics data

'''

print('-'\*50, sep='')

print('Displaying statistics heatmap for some offensive metrics')

print('-'\*50, sep='')

hf\_entry\_analysis\_df = data\_analysis\_df[['years','G','AB','H','HR','RBI','R','2B','3B','inducted\_val']]

# generating correlation heatmap

sns.heatmap(hf\_entry\_analysis\_df.corr(), annot = True)

# posting correlation heatmap to output console

plt.savefig('HOF\_Statistics\_Heatmap.jpg')

plt.show()

def exploreData(data\_analysis):

'''

Exploring the data.

'''

print('-'\*50, sep='')

print('Exploring the data')

print('-'\*50, sep='')

hof\_member\_df = data\_analysis[data\_analysis.inducted\_val == 1]

print('-'\*35, sep='')

print('Displaying Hall of Fame data metrics')

print('-'\*35, sep='')

print('Shape of Hall of Fame dataset:', hof\_member\_df.shape)

maxValues = hof\_member\_df.max()

print('Maximum value in each column for Hall of Fame data : ')

print(maxValues)

#print(hof\_member\_df.max())

minValues = hof\_member\_df.min()

print('Minimum value in each column for Hall of Fame data : ')

print(minValues)

#print(hof\_member\_df.min())

print('Max years:\n',hof\_member\_df[hof\_member\_df.years == hof\_member\_df.years.max()])

print('Min years:\n',hof\_member\_df[hof\_member\_df.years == hof\_member\_df.years.min()])

print('Max games:\n',hof\_member\_df[hof\_member\_df.G == hof\_member\_df.G.max()])

print('Min games:\n',hof\_member\_df[hof\_member\_df.G == hof\_member\_df.G.min()])

print('Max AB:\n',hof\_member\_df[hof\_member\_df.AB == hof\_member\_df.AB.max()])

print('Min AB:\n',hof\_member\_df[hof\_member\_df.AB== hof\_member\_df.AB.min()])

print('Max H:\n',hof\_member\_df[hof\_member\_df.H == hof\_member\_df.H.max()])

print('Min H:\n',hof\_member\_df[hof\_member\_df.H== hof\_member\_df.H.min()])

print('Max HR:\n',hof\_member\_df[hof\_member\_df.HR == hof\_member\_df.HR.max()])

print('Min HR:\n',hof\_member\_df[hof\_member\_df.HR== hof\_member\_df.HR.min()])

print('Max RBI:\n',hof\_member\_df[hof\_member\_df.RBI == hof\_member\_df.RBI.max()])

print('Min RBI:\n',hof\_member\_df[hof\_member\_df.RBI== hof\_member\_df.RBI.min()])

print('-'\*35, sep='')

print('Displaying All Players data metrics')

print('-'\*35, sep='')

print('Shape of All Players data:', data\_analysis.shape)

maxValues = data\_analysis.max()

print('Maximum value in each column for All Players data : ')

print(maxValues)

print('Max years:\n',data\_analysis[data\_analysis.years == data\_analysis.years.max()])

print('Max games:\n',data\_analysis[data\_analysis.G == data\_analysis.G.max()])

print('Max AB:\n',data\_analysis[data\_analysis.AB == data\_analysis.AB.max()])

print('Max H:\n',data\_analysis[data\_analysis.H == data\_analysis.H.max()])

print('Max HR:\n',data\_analysis[data\_analysis.HR == data\_analysis.HR.max()])

print('Max RBI:\n',data\_analysis[data\_analysis.RBI == data\_analysis.RBI.max()])

print("\nSaving data\_analysis\_df to file. \n")

writer = pd.ExcelWriter("data\_analysis\_df.xlsx")

data\_analysis.to\_excel(writer, "analysis")

writer.save()

print("\nSaving hof\_member\_df to file. \n")

writer = pd.ExcelWriter("hof\_member\_df.xlsx")

hof\_member\_df.to\_excel(writer, "analysis")

writer.save()

def compute\_inertia(a, X):

W = [np.mean(pairwise\_distances(X[a == c, :])) for c in np.unique(a)]

return np.mean(W)

def compute\_gap(clustering, data, k\_max=5, n\_references=5):

if len(data.shape) == 1:

data = data.reshape(-1, 1)

reference = np.random.rand(\*data.shape)

reference\_inertia = []

for k in range(1, k\_max+1):

local\_inertia = []

for \_ in range(n\_references):

clustering.n\_clusters = k

assignments = clustering.fit\_predict(reference)

local\_inertia.append(compute\_inertia(assignments, reference))

reference\_inertia.append(np.mean(local\_inertia))

ondata\_inertia = []

for k in range(1, k\_max+1):

clustering.n\_clusters = k

assignments = clustering.fit\_predict(data)

ondata\_inertia.append(compute\_inertia(assignments, data))

gap = np.log(reference\_inertia)-np.log(ondata\_inertia)

return gap, np.log(reference\_inertia), np.log(ondata\_inertia)

#Functions provided by the professor

def compute\_ssq(data, k, kmeans):

dist = np.min(cdist(data, kmeans.cluster\_centers\_, 'euclidean'), axis=1)

tot\_withinss = sum(dist\*\*2) # Total within-cluster sum of squares

totss = sum(pdist(data)\*\*2) / data.shape[0] # The total sum of squares

betweenss = totss - tot\_withinss # The between-cluster sum of squares

return betweenss/totss\*100

#Given a data (as nxm matrix) and an array of ks, this returns the SSQ (sum of squared distances)

#SSQ is also called as SSD or SSE

def ssq\_statistics(data, ks, ssq\_norm=True):

ssqs = sp.zeros((len(ks),)) # array for SSQs (length ks)

for (i,k) in enumerate(ks): # iterate over the range of k values

kmeans = KMeans(n\_clusters=k, random\_state=1234).fit(data)

if ssq\_norm:

ssqs[i] = compute\_ssq(data, k, kmeans)

else:

# The sum of squared error (SSQ) for k

ssqs[i] = kmeans.inertia\_

return ssqs

#Method to plot the gap

def plot\_gap(gap, k\_max):

plt.clf() #clear the plot

plt.plot(range(1, k\_max+1), gap, '-o')

plt.ylabel('gap')

plt.xlabel('k')

plt.savefig('Plot-Gap.pdf')

plt.show()

#Method to plot the inertia

def plot\_inertia(reference\_inertia, ondata\_inertia, k\_max):

plt.clf()#clear the plot

plt.plot(range(1, k\_max+1), reference\_inertia,

'-o', label='reference')

plt.plot(range(1, k\_max+1), ondata\_inertia,

'-o', label='data')

plt.xlabel('k')

plt.ylabel('log(inertia)')

plt.savefig('Plot-Inertia.pdf')

plt.show()

#Method to plot the SSQ

def plot\_ssq(k\_values, ssqs):

plt.clf()#clear the plot

plt.figure()

plt.plot(k\_values, ssqs)

plt.xlabel("Number of cluster")

plt.ylabel("SSQ")

plt.show()

def evaluate\_kmeans(data\_analysis):

data\_analysis\_df = data\_analysis[['AB','H','HR','RBI','inducted\_val']]

hof\_entry\_array = data\_analysis\_df.to\_numpy()

#hof\_entry\_target = hof\_entry\_array[:, -1] # for last column

#print('target--', hof\_entry\_target)

# Get all columns except for last

data\_analysis\_array = hof\_entry\_array[:,:-1]

#print('array', hof\_entry\_array)

k\_values = [x for x in range(1,10)]

ssqs=ssq\_statistics(data\_analysis\_array, k\_values)

print('\n\nSSQ Plot for k= 1 to 10\n','-'\*40, sep='')

plot\_ssq(k\_values,ssqs)

k\_max = max(k\_values)

gap, reference\_inertia, ondata\_inertia = compute\_gap(KMeans(), data\_analysis\_array, k\_max)

print('\n\nGap plot for k= 10\n','-'\*40, sep='')

plot\_gap(gap, k\_max)

print('\nInertia plot for k= 10\n','-'\*40, sep='')

plot\_inertia(reference\_inertia, ondata\_inertia, k\_max)

def perform\_reduction\_analysis(data\_analysis, target\_names):

# Reference code- https://scikit-learn.org/stable/auto\_examples/decomposition/plot\_pca\_vs\_lda.html#sphx-glr-auto-examples-decomposition-plot-pca-vs-lda-py

data\_analysis\_df = data\_analysis[['AB','H','HR','RBI','inducted\_val']]

data\_analysis\_array = data\_analysis\_df.to\_numpy()

analysis\_target = data\_analysis\_array[:, -1] # for last column

# Get all columns except and last

data\_analysis\_array = data\_analysis\_array[:,:-1]

X = data\_analysis\_array

y = analysis\_target

y = y.astype('int')

pca = PCA(n\_components=2)

#X\_r = pca.fit(X).transform(X)

X\_r = pca.fit\_transform(X)

print('X\_r', X\_r)

data\_analysis['pc\_1'] = X\_r[:,0] # sets this first principal component

data\_analysis['pc\_2'] = X\_r[:,1] # sets this second principal component

hof\_low\_pc = data\_analysis[(data\_analysis['inducted\_val']==1) & (data\_analysis['pc\_1']<2000)]

print('hof\_low\_pc:', hof\_low\_pc.shape)

non\_hof\_high\_pc = data\_analysis[(data\_analysis['inducted\_val']==0) & ((data\_analysis['pc\_1']> 9000) | (data\_analysis['pc\_2'] > 750) | (data\_analysis['pc\_2'] < -5000))]

print('non\_hof\_high\_pc:', non\_hof\_high\_pc.shape)

print("\nSaving PCA Analysis data pcaAnalysisData.xlsx \n")

writer = pd.ExcelWriter("pcaAnalysisData.xlsx")

data\_analysis.to\_excel(writer, "analysis")

writer.save()

print("\nSaving PCA Hall of Fame Outliers pcaHallOfFameOutliers.xlsx \n")

writer = pd.ExcelWriter("pcaHallOfFameOutliers.xlsx")

hof\_low\_pc.to\_excel(writer, "analysis")

writer.save()

print("\nSaving PCA Non-Hall of Fame Outliers pcaNonHallOfFameOutliers.xlsx \n")

writer = pd.ExcelWriter("pcaNonHallOfFameOutliers.xlsx")

non\_hof\_high\_pc.to\_excel(writer, "analysis")

writer.save()

#lda = LinearDiscriminantAnalysis()

#X\_r2 = lda.fit(X, y).transform(X)

# Percentage of variance explained for each components

print('explained variance ratio (first two components): %s'

% str(pca.explained\_variance\_ratio\_))

plt.figure()

colors = ['navy', 'turquoise']

lw = 2

for color, i, target\_name in zip(colors, [0, 1], target\_names):

plt.scatter(X\_r[y == i, 0], X\_r[y == i, 1], color=color, alpha=.8, lw=lw,

label=target\_name)

plt.legend(loc='best', shadow=False, scatterpoints=1)

plt.title('PCA of dataset')

plt.show()

return hof\_low\_pc, non\_hof\_high\_pc

def get\_kmeans\_clusters(X, number\_clusters):

# Reference https://www.kaggle.com/minc33/visualizing-high-dimensional-clusters

# Assume data\_analysis contains only numerical variables

#Initialize our scaler

scaler = StandardScaler()

#Scale each column in numer

X = pd.DataFrame(scaler.fit\_transform(X))

#Initialize our model

kmeans = KMeans(n\_clusters = number\_clusters)

#Fit our model

kmeans.fit(X)

#Find which cluster each data-point belongs to

clusters = kmeans.predict(X)

return clusters

def perform\_kmeans(X):

# Reference https://www.kaggle.com/minc33/visualizing-high-dimensional-clusters

# Assume data\_analysis contains only numerical variables

number\_clusters = 2

#Find which cluster each data-point belongs to

clusters = get\_kmeans\_clusters(X, number\_clusters)

#Add the cluster vector to our DataFrame, X

X['Cluster'] = clusters

# Map the cluster values to match target values

X['Cluster'] = X['Cluster'].map({0:1,1:0})

#plotX is a DataFrame containing 5000 values sampled randomly from X

plotX = pd.DataFrame(np.array(X))

#Rename plotX's columns since it was briefly converted to an np.array above

plotX.columns = X.columns

#PCA with one principal component

pca\_1d = PCA(n\_components=1)

#PCA with two principal components

pca\_2d = PCA(n\_components=2)

#This DataFrame holds that single principal component mentioned above

PCs\_1d = pd.DataFrame(pca\_1d.fit\_transform(plotX.drop(["Cluster"], axis=1)))

#This DataFrame contains the two principal components that will be used

#for the 2-D visualization mentioned above

PCs\_2d = pd.DataFrame(pca\_2d.fit\_transform(plotX.drop(["Cluster"], axis=1)))

PCs\_1d.columns = ["PC1\_1d"]

#"PC1\_2d" means: 'The first principal component of the components created for 2-D visualization, by PCA.'

#And "PC2\_2d" means: 'The second principal component of the components created for 2-D visualization, by PCA.'

PCs\_2d.columns = ["PC1\_2d", "PC2\_2d"]

plotX = pd.concat([plotX,PCs\_1d,PCs\_2d], axis=1, join='inner')

plotX["dummy"] = 0

#Note that all of the DataFrames below are sub-DataFrames of 'plotX'.

#This is because we intend to plot the values contained within each of these DataFrames.

cluster0 = plotX[plotX["Cluster"] == 0]

cluster1 = plotX[plotX["Cluster"] == 1]

#cluster2 = plotX[plotX["Cluster"] == 2]

#Instructions for building the 1-D plot

#trace1 is for 'Cluster 0'

trace1 = go.Scatter(

x = cluster0["PC1\_1d"],

y = cluster0["dummy"],

mode = "markers",

name = "Cluster 0",

marker = dict(color = 'rgba(255, 128, 255, 0.8)'),

text = None)

#trace2 is for 'Cluster 1'

trace2 = go.Scatter(

x = cluster1["PC1\_1d"],

y = cluster1["dummy"],

mode = "markers",

name = "Cluster 1",

marker = dict(color = 'rgba(255, 128, 2, 0.8)'),

text = None)

data = [trace1, trace2]

title = "Visualizing Clusters in One Dimension Using PCA"

layout = dict(title = title,

xaxis= dict(title= 'PC1',ticklen= 5,zeroline= False),

yaxis= dict(title= '',ticklen= 5,zeroline= False)

)

fig = dict(data = data, layout = layout)

plot(fig, filename='1D-PCA.html')

#Instructions for building the 2-D plot

#trace1 is for 'Cluster 0'

trace1 = go.Scatter(

x = cluster0["PC1\_2d"],

y = cluster0["PC2\_2d"],

mode = "markers",

name = "Cluster 0",

marker = dict(color = 'rgba(255, 128, 255, 0.8)'),

text = None)

#trace2 is for 'Cluster 1'

trace2 = go.Scatter(

x = cluster1["PC1\_2d"],

y = cluster1["PC2\_2d"],

mode = "markers",

name = "Cluster 1",

marker = dict(color = 'rgba(255, 128, 2, 0.8)'),

text = None)

data = [trace1, trace2]

title = "Visualizing Clusters in Two Dimensions Using PCA"

layout = dict(title = title,

xaxis= dict(title= 'PC1',ticklen= 5,zeroline= False),

yaxis= dict(title= 'PC2',ticklen= 5,zeroline= False)

)

fig = dict(data = data, layout = layout)

plot(fig, filename='2D-PCA.html')

return clusters

def analyze\_kmeans(data\_analysis, hof\_pc\_outliers):

data\_analysis\_df = data\_analysis[['AB','H','HR','RBI']]

clusters = perform\_kmeans(data\_analysis\_df)

data\_analysis['cluster\_k2'] = clusters

# Map the cluster values to match target values

data\_analysis['cluster\_k2'] = data\_analysis['cluster\_k2'].map({0:1,1:0})

# Set clusters to new mapping

clusters = data\_analysis['cluster\_k2']

print('-'\*50, sep='')

print('Analyzing kmeans results when k = 2')

print('-'\*50, sep='')

confusion\_data = confusion\_matrix(data\_analysis['inducted\_val'].tolist(), clusters)

# Used as reference: https://stats.stackexchange.com/questions/95731/how-to-calculate-purity

print('confusion\_matrix',confusion\_data)

max\_values = 0

for i in range(2):

max\_values += max(confusion\_data[i])

purity\_score = max\_values/data\_analysis.shape[0]

print("Purity score = ", purity\_score)

hof\_outliers = hof\_pc\_outliers[['playerID','nameFirst','nameLast']]

data\_analysis\_false\_neg = data\_analysis[(data\_analysis['inducted\_val']==1) & (data\_analysis['cluster\_k2']==0)]

data\_analysis\_false\_neg = data\_analysis\_false\_neg[['playerID','nameFirst','nameLast']]

print('Shape of Hall of Fame outliers (Players in HoF but do not have impressive numbers.):', hof\_outliers.shape)

print('Shape of dataframe containing Hall of Fame Players but in Cluster indicated not in HOF:', data\_analysis\_false\_neg.shape)

diff\_df = pd.concat([hof\_outliers,data\_analysis\_false\_neg]).drop\_duplicates(keep=False)

print('Difference between the two sets:', diff\_df.to\_string())

'''

print("\nSaving data to hof\_outliers.xlsx \n")

writer = pd.ExcelWriter("hof\_outliers.xlsx")

hof\_outliers.to\_excel(writer, "analysis")

writer.save()

print("\nSaving data to data\_analysis\_false\_neg.xlsx \n")

writer = pd.ExcelWriter("data\_analysis\_false\_neg.xlsx")

data\_analysis\_false\_neg.to\_excel(writer, "analysis")

writer.save()

'''

# Analyze other values when k=3

print('-'\*50, sep='')

print('Analyzing kmeans results when k = 3 and combining two clusters into one to get two clusters only')

print('-'\*50, sep='')

number\_clusters = 3

cluster\_k3 = get\_kmeans\_clusters(data\_analysis\_df, number\_clusters)

data\_analysis['cluster\_k3'] = cluster\_k3

data\_analysis['cluster\_k3'] = data\_analysis['cluster\_k3'].map({0:1,1:0,2:2})

# Map the cluster values to match target values

data\_analysis['cluster\_k3\_revised'] = data\_analysis['cluster\_k3'].map({0:1,1:0,2:0})

cluster\_k3 = data\_analysis['cluster\_k3\_revised']

confusion\_data = confusion\_matrix(data\_analysis['inducted\_val'].tolist(), cluster\_k3)

# Used as reference: https://stats.stackexchange.com/questions/95731/how-to-calculate-purity

print('confusion\_matrix',confusion\_data)

max\_values = 0

for i in range(2):

max\_values += max(confusion\_data[i])

purity\_score = max\_values/data\_analysis.shape[0]

print("Purity score = ", purity\_score)

print("\nSaving data to k\_Means\_Analysis.xlsx \n")

writer = pd.ExcelWriter("k\_Means\_Analysis.xlsx")

data\_analysis.to\_excel(writer, "kmeans\_Analysis")

writer.save()

def main():

np.random.seed(20) # Initiate the random seed

target\_names = ['Not Hall of Famer', 'Hall of Famer']

data\_analysis\_df = readTables()

createStatisticsHeatmap(data\_analysis\_df)

exploreData(data\_analysis\_df)

hof\_pc\_outliers, non\_hof\_pc\_outliers = perform\_reduction\_analysis(data\_analysis\_df, target\_names)

analyze\_kmeans(data\_analysis\_df, hof\_pc\_outliers)

#evaluate\_kmeans(data\_analysis\_df)

if \_\_name\_\_ == '\_\_main\_\_':

main()















