Data Science – Term Project

Cricket World Cup Matches Prediction

201835528 조병근 | 201835538 최지원 | 201931889 정다희

[Full Code]

#  Data Science

#  Term project - Cricket World Cup Matches Prediction

#  Name            :  Jo Byeong Geun, Choi Ji Won, Jeong Da Hee

#  Student Number  :  201835528, 201835538, 201931889

#  Class           :  Mon 5678

#  Date            :  2021.05.26

#  Update          :  2021.06.11

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression, LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix, plot\_confusion\_matrix, classification\_report

from sklearn.model\_selection import KFold

from sklearn.ensemble import RandomForestClassifier

from sklearn import metrics

from sklearn import svm, datasets

from warnings import simplefilter

import warnings

with warnings.catch\_warnings():

    warnings.filterwarnings("ignore")

simplefilter(action='ignore', category=FutureWarning)

# Make Confusion Matirx

def make\_cf\_matrix(y\_test, y\_pred):

    # Compare y testing sets with predicted y values

    cf\_matrix = confusion\_matrix(y\_test, y\_pred)

    # Show the confusion Matrix

    print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Confusion Matrix\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print(cf\_matrix)

# Make Classification Report

def make\_cf\_report(y\_test, y\_pred):

    # Evaluate the precision and reproducibility

    cf\_report = classification\_report(y\_test, y\_pred)

    # Show the Classificatin Report

    print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Classification Report\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print(cf\_report)

# Scaling data from wicketts, runs

def scaler\_team(df\_team, df, scaler):

    value = []

    for i in df\_team.index:

        # Importing data in the 'Margin' column

        value.append(df.loc[i]['Margin'])

    # Change dimension and shape

    value = np.array(value).reshape(-1, 1)

    scaled = scaler.transform(value)

    # Scaling to mean values

    return np.mean(scaled)

# Scaling using scaler

def scaling(scaler):

    wickets\_np = df\_wickets['Margin'].to\_numpy().reshape(-1, 1)

    scaler.fit(wickets\_np)

    # Declaring a list to insert values for scaling

    wicket\_scaler = []

    wicket\_scaler.append(scaler\_team(df\_England\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_SouthAfrica\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_WestIndies\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_Pakistan\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_NewZealand\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_SriLanka\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_Afghanistan\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_Australia\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_Bangladesh\_wicket, df\_wickets, scaler))

    wicket\_scaler.append(scaler\_team(df\_India\_wicket, df\_wickets, scaler))

    runs\_np = df\_runs['Margin'].to\_numpy().reshape(-1, 1)

    scaler.fit(runs\_np)

    # Declaring a list to insert values for scaling

    runs\_scaler = []

    runs\_scaler.append(scaler\_team(df\_England\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_SouthAfrica\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_WestIndies\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_Pakistan\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_NewZealand\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_SriLanka\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_Afghanistan\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_Australia\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_Bangladesh\_runs, df\_runs, scaler))

    runs\_scaler.append(scaler\_team(df\_India\_runs, df\_runs, scaler))

    # Outputs scaling results for worldcup\_team in bar form

    plt.bar(worldcup\_teams, wicket\_scaler)

    # X axis name

    plt.xlabel('Teams')

    # Y axis name

    plt.ylabel('Scaled Wicket')

    # Show plotting result

    plt.show()

    # Outputs scaling results for worldcup\_team in bar form

    plt.bar(worldcup\_teams, runs\_scaler)

    # X axis name

    plt.xlabel('Teams')

    # Y axis name

    plt.ylabel('Scaled Runs')

    # Show plotting result

    plt.show()

    return wicket\_scaler, runs\_scaler

# Kfold- Linear Regression

def kfold\_lin(kfold, df\_teams):

    sum\_training = 0

    sum\_testing = 0

    count = 0

    for train\_index, test\_index in kfold.split(df\_teams):

        # Data for training

        df\_train = df\_teams.iloc[train\_index]

        # Data for testing

        df\_test = df\_teams.iloc[test\_index]

        model = LinearRegression()

        # Change the training set and testing set to the correct structure and store them.

        train = pd.get\_dummies(df\_train, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

        test = pd.get\_dummies(df\_test, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

        # Drop 'Winner' column of training set and put it in the X training set.

        X\_train = train.drop(['Winner'], axis=1)

        # Insert the 'Winner' column of training set into the y training set.

        y\_train = train["Winner"]

        # Drop 'Winner' column of testing set and put it in the X testing set.

        X\_test = test.drop(['Winner'], axis=1)

        # Insert the 'Winner' column of testing set into the y testing set.

        y\_test = test["Winner"]

        model.fit(X\_train, y\_train)

        # Derives the value of learning the training set to Linear Regression

        score = model.score(X\_train, y\_train)

        # Derives the value of learning the testing set to Linear Regression

        score2 = model.score(X\_test, y\_test)

        count = count + 1

        sum\_training = sum\_training + score

        sum\_testing = sum\_testing + score2

        # Accuracy output each K-fold data

        print("Training set accuracy: ", '%.3f' % (score))

        print("Test set accuracy: ", '%.3f' % (score2))

    print("Training set accuracy average: ", '%.3f' % (sum\_training / count))

    print("Test set accuracy average: ", '%.3f' % (sum\_testing / count))

    return sum\_testing/count

# Kfold - Logistic Regression

def kfold\_log(kfold,df\_teams):

    sum\_training = 0

    sum\_testing = 0

    count = 0

    for train\_index, test\_index in kfold.split(df\_teams):

        # Data for training

        df\_train = df\_teams.iloc[train\_index]

        # Data for testing

        df\_test = df\_teams.iloc[test\_index]

        model = LogisticRegression()

        # Change the training set and testing set to the correct structure and store them.

        train = pd.get\_dummies(df\_train, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

        test = pd.get\_dummies(df\_test, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

        # Drop 'Winner' column of training set and put it in the X training set.

        X\_train = train.drop(['Winner'], axis=1)

        # Insert the 'Winner' column of training set into the y training set.

        y\_train = train["Winner"]

        # Drop 'Winner' column of testing set and put it in the X testing set.

        X\_test = test.drop(['Winner'], axis=1)

        # Insert the 'Winner' column of testing set into the y testing set.

        y\_test = test["Winner"]

        model.fit(X\_train, y\_train)

        # Derives the value of learning the training set to Logistic Regression

        score = model.score(X\_train, y\_train)

        # Derives the value of learning the testing set to Logistic Regression

        score2 = model.score(X\_test, y\_test)

        count = count + 1

        sum\_training = sum\_training + score

        sum\_testing = sum\_testing + score2

        # Accuracy output each K-fold data

        print("Training set accuracy: ", '%.3f' % (score))

        print("Test set accuracy: ", '%.3f' % (score2))

    print("Training set accuracy average: ", '%.3f' % (sum\_training / count))

    print("Test set accuracy average: ", '%.3f' % (sum\_testing / count))

    return sum\_testing/count

# Kfold - Random Forest

def kfold\_ran(kfold, best\_score, df\_teams):

    sum\_training=0

    sum\_testing=0

    count=0

    for train\_index, test\_index in kfold.split(df\_teams):

        # Data for training

        df\_train = df\_teams.iloc[train\_index]

        # Data for testing

        df\_test = df\_teams.iloc[test\_index]

        # Change the training set and testing set to the correct structure and store them.

        train = pd.get\_dummies(df\_train, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

        test = pd.get\_dummies(df\_test, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

        # Drop 'Winner' column of training set and put it in the X training set.

        X\_train = train.drop(['Winner'], axis=1)

        # Insert the 'Winner' column of training set into the y training set.

        y\_train = train["Winner"]

        # Drop 'Winner' column of testing set and put it in the X testing set.

        X\_test = test.drop(['Winner'], axis=1)

        # Insert the 'Winner' column of testing set into the y testing set.

        y\_test = test["Winner"]

        # Random Forest

        rf = RandomForestClassifier(n\_estimators=100, max\_depth=20, random\_state=0)

        # Learning X training set by y training

        rf.fit(X\_train, y\_train)

        # Derives the value of learning the training set to Random Forest

        score = rf.score(X\_train, y\_train)

        # Derives the value of learning the testing set to Random Forest

        score2 = rf.score(X\_test, y\_test)

        count=count+1

        sum\_training=sum\_training+score

        sum\_testing=sum\_testing+score2

        # Store the Random Forest of the best test value that have learned

        if best\_score < score2:

            best\_score = score2

            best\_rf = rf

        # Accuracy output each K-fold data

        print("Training set accuracy: ", '%.3f' % (score))

        print("Test set accuracy: ", '%.3f' % (score2))

    print("Training set accuracy average: ", '%.3f' % (sum\_training/count))

    print("Test set accuracy average: ", '%.3f' % (sum\_testing/count))

    return sum\_testing/count, best\_rf

# Label Encoding

def labelEncoding(df\_teams):

        df\_teams = df\_teams.apply(LabelEncoder().fit\_transform)

        return df\_teams

# Best accuracy of K-Fold & algorithm (Linear, Logistic and RandomForest)

def best\_kfold\_reg(df\_teams):

    #K-fold Linear Regression

    print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*Kfold of Linear Regression\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    scores = []

    kfold = KFold(n\_splits=3, shuffle=True)

    scaled\_kfold\_lin = kfold\_lin(kfold, df\_teams)

    #K-fold Logistic Regression

    print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*Kfold of Logistic Regression\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    scores = []

    kfold = KFold(n\_splits=3, shuffle=True)

    df\_teams['Winner']=df\_teams['Winner'].astype(int)

    scaled\_kfold\_log = kfold\_log(kfold,df\_teams)

    # Kfold random forest

    # Split data into three

    print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*Kfold of Random Forest \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    kfold = KFold(n\_splits=3, shuffle=True)

    # Declaring a variable to put the best value in

    best\_score = 0

    scaled\_kfold\_ran, best\_rf = kfold\_ran(kfold, best\_score, df\_teams)

    best\_acc = bestAcc(scaled\_kfold\_lin, scaled\_kfold\_log, scaled\_kfold\_ran)

    #print("\nthe best acc = ", '%0.3f' % best\_acc)

    return best\_acc

# Calculate best accuracy

def bestAcc(lin, log, ran):

    return max(lin, max(log, ran))

# Best combination

def best\_combi(df\_teams\_minmax, df\_teams\_standard, df\_teams\_robust):

    return max(best\_kfold\_reg(df\_teams\_minmax), max(best\_kfold\_reg(df\_teams\_standard), best\_kfold\_reg(df\_teams\_robust)))

# Read the csv file

results = pd.read\_csv('dataset/results.csv')

# dataset statistical data

print("\n\*\*\*\*\*\*\*\*\*\*\*statistical data\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(results.describe())

# Feature names

print("\n\*\*\*\*\*\*\*\*\*\*\*feature names\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(results.columns.values)

# data types

print("\n\*\*\*\*\*\*\*\*\*\*\*feature data types\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(results.dtypes)

# data shape

print("\n\*\*\*\*\*\*\*\*\*\*\*data shape\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(results.shape)

# data index

print("\n\*\*\*\*\*\*\*\*\*\*\*data index\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(results.index)

# dataset columns

print("\n\*\*\*\*\*\*\*\*\*\*\*dataset columns\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(results.columns)

# Top 5 data in dataset

print("\n\*\*\*\*\*\*\*\*\*\*\*dataset\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(results.head())

# data preprocessing

# Drop missing null or ground information that is not needed

results.dropna(axis=0, inplace=True)

results.drop(['Ground'], axis=1, inplace=True)

# Limited to 10 teams participating in the World Cup

worldcup\_teams = ['England', 'South Africa', 'West Indies',

                  'Pakistan', 'New Zealand', 'Sri Lanka', 'Afghanistan',

                  'Australia', 'Bangladesh', 'India']

# Include only countries corresponding to worldcup\_teams in results dataset of column 'Team\_1'

df\_teams\_1 = results[results['Team\_1'].isin(worldcup\_teams)]

# Include only countries corresponding to worldcup\_teams in results dataset of column 'Team\_2'

df\_teams = df\_teams\_1[df\_teams\_1['Team\_2'].isin(worldcup\_teams)]

# deduplication

df\_teams = df\_teams.drop\_duplicates()

# Clean up index

df\_teams = df\_teams.sort\_index(ascending=True)

# Show df\_teams dataset

print("\n\*\*\*\*\*\*\*\*\*\*\*df\_teams dataset\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(df\_teams.head())

# data shape

print("\n\*\*\*\*\*\*\*\*\*\*\*df\_teams data shape\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(df\_teams.shape)

# If the defense team wins

# data with a value of 'Wicket' in the 'Margin' column will be separately saved.

df\_wickets = df\_teams.loc[df\_teams['Margin'].str.contains('wickets')].copy()

# To omit the word 'wickets' from the value and only insert numbers (int types)

df\_wickets['Margin'] = df\_wickets.Margin.str.extract('(\d+)')

# Create time series data using the 'data' value

df\_wickets['year'] = pd.DatetimeIndex(df\_wickets['date']).year

df\_England\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[0]]

df\_SouthAfrica\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[1]]

df\_WestIndies\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[2]]

df\_Pakistan\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[3]]

df\_NewZealand\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[4]]

df\_SriLanka\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[5]]

df\_Afghanistan\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[6]]

df\_Australia\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[7]]

df\_Bangladesh\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[8]]

df\_India\_wicket = df\_wickets[df\_wickets['Winner'] == worldcup\_teams[9]]

# If the offense team wins

# data with a value of 'runs' in the 'Margin' column will be separately saved.

df\_runs = df\_teams.loc[df\_teams['Margin'].str.contains('runs')].copy()

# To omit the word 'runs' from the value and only insert numbers (int types)

df\_runs['Margin'] = df\_runs.Margin.str.extract('(\d+)')

# Create time series data using the 'data' value

df\_runs['year'] = pd.DatetimeIndex(df\_runs['date']).year

df\_England\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[0]]

df\_SouthAfrica\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[1]]

df\_WestIndies\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[2]]

df\_Pakistan\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[3]]

df\_NewZealand\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[4]]

df\_SriLanka\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[5]]

df\_Afghanistan\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[6]]

df\_Australia\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[7]]

df\_Bangladesh\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[8]]

df\_India\_runs = df\_runs[df\_runs['Winner'] == worldcup\_teams[9]]

########### Scaling ###########

# MinMax Scaling

minmax\_scaler = MinMaxScaler()

minmax\_wicket\_scaler, minmax\_runs\_scaler = scaling(minmax\_scaler)

# Standard Scaling

standard\_scaler = StandardScaler()

standard\_wicket\_scaler, standard\_runs\_scaler = scaling(standard\_scaler)

# Robust Scaling

robust\_scaler = RobustScaler()

robust\_wicket\_scaler, robust\_runs\_scaler = scaling(robust\_scaler)

# Drop columns of 'Margin' and 'date' that has nothing to do with winning or losing.

df\_teams = df\_teams.drop(['date', 'Margin'], axis=1)

# reorder index

df\_teams = df\_teams.reset\_index(drop=True)

########## Encoding ###########

# LabelEncoding to change team name to number

df\_teams = labelEncoding(df\_teams)

# Divide the categorical properties for Team\_1 and Team\_2.

final = pd.get\_dummies(df\_teams, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

# Drop 'Winner' column from final dataset and put it in X

X = final.drop(['Winner'], axis=1)

# Store only 'Winner' column in y in final dataset

y = final["Winner"]

# Divide training and testing set by 7:3 each.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.30, random\_state=42)

# Random Forest

rf = RandomForestClassifier(n\_estimators=100, max\_depth=20, random\_state=0)

# Learning X training set by y training

rf.fit(X\_train, y\_train)

# Predicting an X testing set with learning results

y\_pred = rf.predict(X\_test)

# Derives the value of learning the training set to Random Forest

score = rf.score(X\_train, y\_train)

# Derives the value of learning the testing set to Random Forest

score2 = rf.score(X\_test, y\_test)

# Show Random Forest

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Random Forest\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred))

print('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))

# Accuracy output

print("Training set accuracy: ", '%.3f' % (score))

print("Test set accuracy: ", '%.3f' % (score2))

# K neareast neighbor

k\_list = range(1,10)

training\_accuracy = []

test\_accuracy = []

for k in k\_list:

  classifier = KNeighborsClassifier(n\_neighbors = k)

  # Learning X training set by y training

  classifier.fit(X\_train, y\_train)

  # Store accuracy of training set

  training\_accuracy.append(classifier.score(X\_train, y\_train))

  # Store accuracy of testing set

  test\_accuracy.append(classifier.score(X\_test, y\_test))

# Outputs K neareast neighbor result

plt.plot(k\_list, training\_accuracy)

# X axis name

plt.xlabel("n\_neighbors")

# Y axis name

plt.ylabel("Accuracy")

# Show plotting result

plt.show()

# Prediction of testing set

clf = KNeighborsClassifier(n\_neighbors = 7)

# Learning X training set by y training

clf.fit(X\_train, y\_train)

# Show the prediction and accuracy of testing set

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*KNN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print("Test Prediction: {}".format(clf.predict(X\_test)))

print("Test Accuracy: {:.2f}".format(clf.score(X\_test, y\_test)))

# Linear Regression

linreg = LinearRegression()

# Learning X training set by y training

linreg.fit(X\_train, y\_train)

# Predicting an X testing set with learning results

y\_pred = linreg.predict(X\_test)

# Derives the value of learning the training set to Linear regression

score = linreg.score(X\_train, y\_train)

# Derives the value of learning the testing set to Linear regression

score2 = linreg.score(X\_test, y\_test)

# Show Linear Regression

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Linear Regression\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred))

print('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))

# Accuracy output

print("Training set accuracy: ", '%.3f' % (score))

print("Test set accuracy: ", '%.3f' % (score2))

# Logistic Regression

logreg = LogisticRegression()

# Learning X training set by y training

logreg.fit(X\_train, y\_train)

# Predicting an X testing set with learning results

y\_pred = logreg.predict(X\_test)

# Derives the value of learning the training set to Logistic regression

score = logreg.score(X\_train, y\_train)

# Derives the value of learning the testing set to Logistic regression

score2 = logreg.score(X\_test, y\_test)

# Show Logistic Regression

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Logistic Regression\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print('Mean Absolute Error:', metrics.mean\_absolute\_error(y\_test, y\_pred))

print('Mean Squared Error:', metrics.mean\_squared\_error(y\_test, y\_pred))

print('Root Mean Squared Error:', np.sqrt(metrics.mean\_squared\_error(y\_test, y\_pred)))

# Accuracy output

print("Training set accuracy: ", '%.3f' % (score))

print("Test set accuracy: ", '%.3f' % (score2))

# Show Confusion Matrix and Classification Report

make\_cf\_matrix(y\_test, y\_pred)

make\_cf\_report( y\_test, y\_pred)

# Confusion matrix Visualization

cf\_matrix = confusion\_matrix(y\_test, y\_pred)

# Support Vector classifier

clf = svm.SVC(random\_state=0)

# Learning X training set by y training

clf.fit(X\_train, y\_train)

svm.SVC(random\_state=0)

# Decide how to show the matrix

matrix = plot\_confusion\_matrix(clf, X\_test, y\_test, cmap=plt.cm.Blues, normalize='true')

# Title of matrix table

plt.title('Confusion matrix for our classifier')

# Show the matrix table

plt.show()

Scaler = MinMaxScaler()

df\_teams\_minmax = pd.DataFrame(Scaler.fit\_transform(df\_teams), columns=['Team\_1', 'Team\_2', 'Winner'])

print(df\_teams\_minmax)

Scaler = StandardScaler()

df\_teams\_standard = pd.DataFrame(Scaler.fit\_transform(df\_teams), columns=['Team\_1', 'Team\_2', 'Winner'])

print(df\_teams\_standard)

Scaler = RobustScaler()

df\_teams\_robust = pd.DataFrame(Scaler.fit\_transform(df\_teams), columns=['Team\_1', 'Team\_2', 'Winner'])

print(df\_teams\_robust)

best\_combi\_acc = best\_combi(df\_teams\_minmax, df\_teams\_standard, df\_teams\_robust)

print("\nbest combination accuracy : ", best\_combi\_acc)

# Expected qualifier rank

# Read the csv file

ranking = pd.read\_csv('dataset/icc\_rankings.csv')

fixtures = pd.read\_csv('dataset/fixtures.csv')

# Show Top 5 data of fixtures dataset

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*icc\_rankings dataset\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(ranking.head())

# Show last 5 data of fixtures dataset

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Fixtures dataset\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(fixtures.tail())

# Ranked using icc\_ranking dataset

fixtures.insert(1, 'first\_position', fixtures['Team\_1'].map(ranking.set\_index('Team')['Position']))

fixtures.insert(2, 'second\_position', fixtures['Team\_2'].map(ranking.set\_index('Team')['Position']))

# Find the value through the integral position

fixtures = fixtures.iloc[:45, :]

# Show last 5 data of fixtures dataset with ranking

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*fixtures dataset with ranking\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(fixtures.tail())

# Declaring a variable to put the predicted value

pred\_ranking\_set = []

# The first variable is indexed, and the second variable is accessed one by one in the row in the column.

for idx, row in fixtures.iterrows():

    # Change the value of team\_1 or team\_2 according to rank

    if row['first\_position'] < row['second\_position']:

        pred\_ranking\_set.append({'Team\_1': row['Team\_1'], 'Team\_2': row['Team\_2'], 'Winner': None})

    else:

        pred\_ranking\_set.append({'Team\_1': row['Team\_2'], 'Team\_2': row['Team\_1'], 'Winner': None})

# Convert DataFrame type

pred\_ranking\_set = pd.DataFrame(pred\_ranking\_set)

# Top 5 data in dataset

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Predicted dataset\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(pred\_ranking\_set.head())

# Declares variables for backing up data in 'df\_teams'

backup\_df\_teams = df\_teams

# Divide the categorical properties for Team\_1 and Team\_2.

backup\_final = pd.get\_dummies(backup\_df\_teams, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

# Declares variables for backing up data in 'pred\_set'

backup\_pred\_set = pred\_ranking\_set

# Divide the categorical properties for Team\_1 and Team\_2.

pred\_ranking\_set = pd.get\_dummies(pred\_ranking\_set, prefix=['Team\_1', 'Team\_2'], columns=['Team\_1', 'Team\_2'])

# Store the differences between existing dataset and predicted dataset

missing\_cols = set(backup\_final.columns) - set(pred\_ranking\_set.columns)

# Store missing value as 0 in predicted dataset

for c in missing\_cols:

    pred\_ranking\_set[c] = 0

# Insert data of the column in 'backup\_final' into the predicted dataset

pred\_ranking\_set = pred\_ranking\_set[backup\_final.columns]

# Drop the 'Winner' column

pred\_ranking\_set = pred\_ranking\_set.drop(['Winner'], axis=1)

# Put the dataset to experiment into the random forest to derive the result value

best\_rf = RandomForestClassifier(n\_estimators=100, max\_depth=20, random\_state=0)

best\_rf.fit(X\_train, y\_train)

predict\_result = best\_rf.predict(pred\_ranking\_set)

# Show expected qualifier rank

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*Qualifier\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

for i in range(fixtures.shape[0]):

    # Show the country to compete with

    print(backup\_pred\_set.iloc[i, 1] + " and " + backup\_pred\_set.iloc[i, 0])

    if predict\_result[i] == 1:

        # The second written country wins

        print("Winner: " + backup\_pred\_set.iloc[i, 1])

    else:

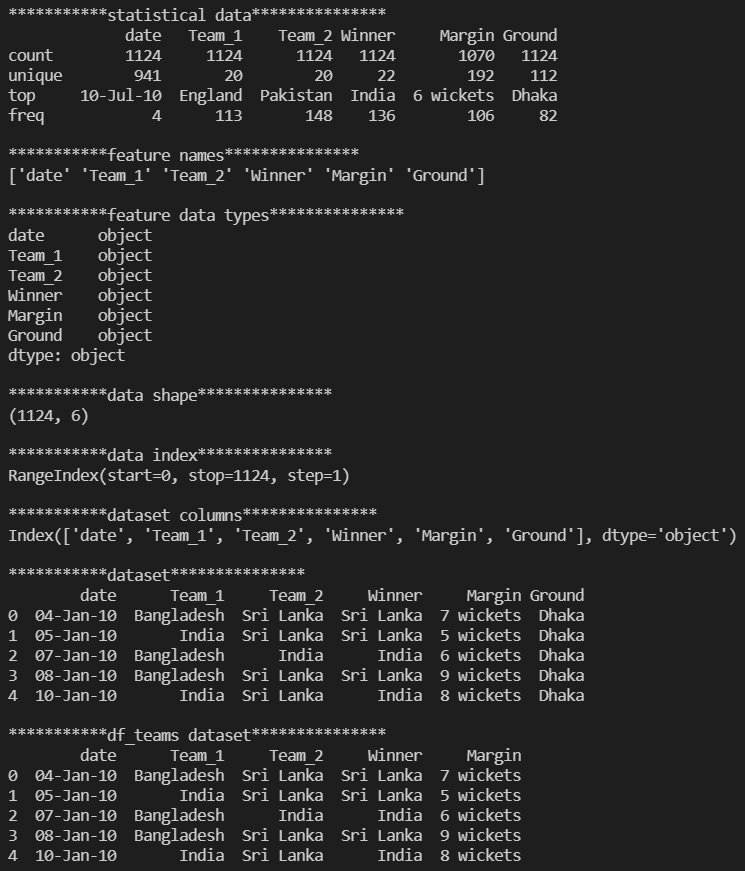
        # The first written country wins

        print("Winner: " + backup\_pred\_set.iloc[i, 0])

    print("")

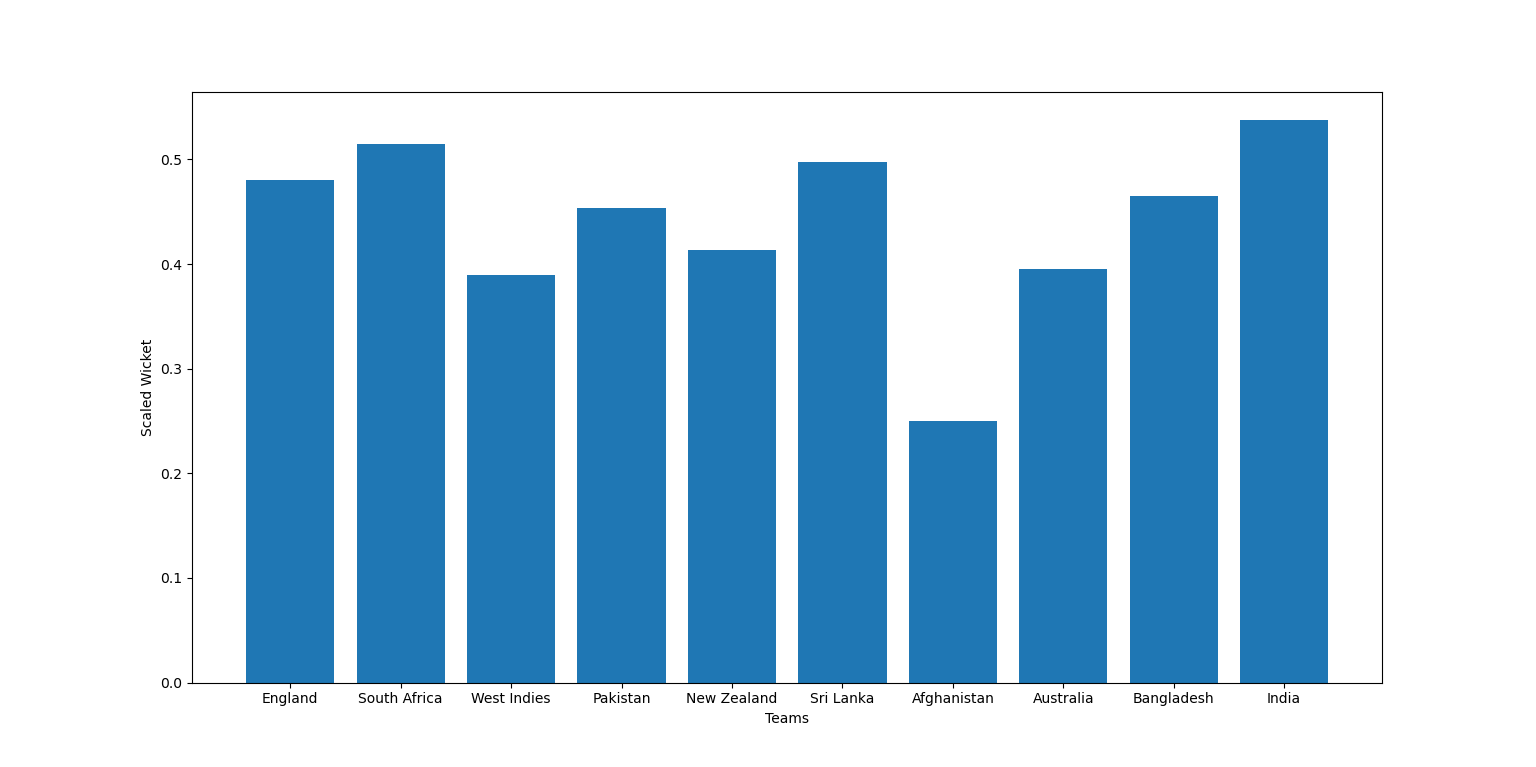
[result]

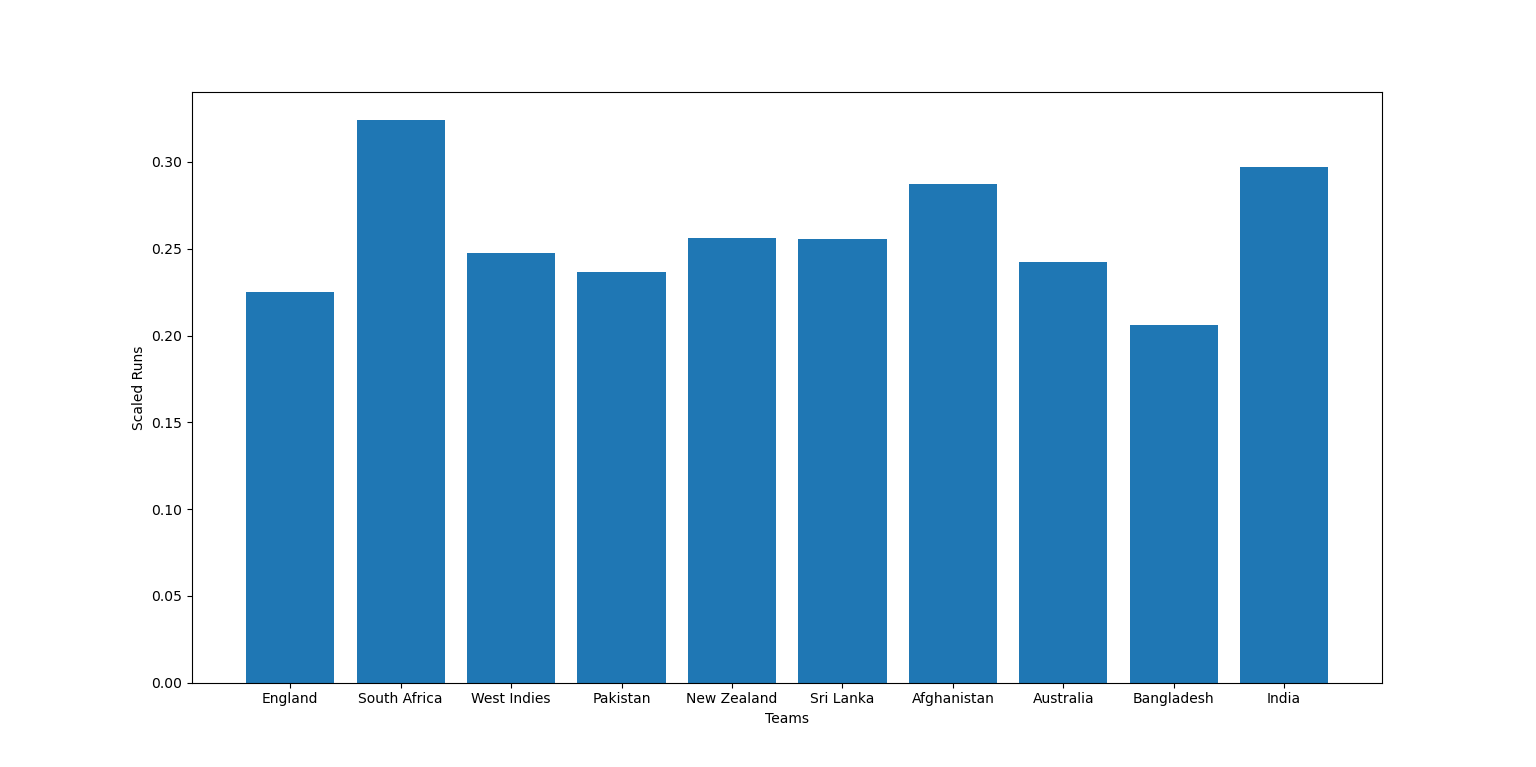
<Describe ‘results’ dataset>



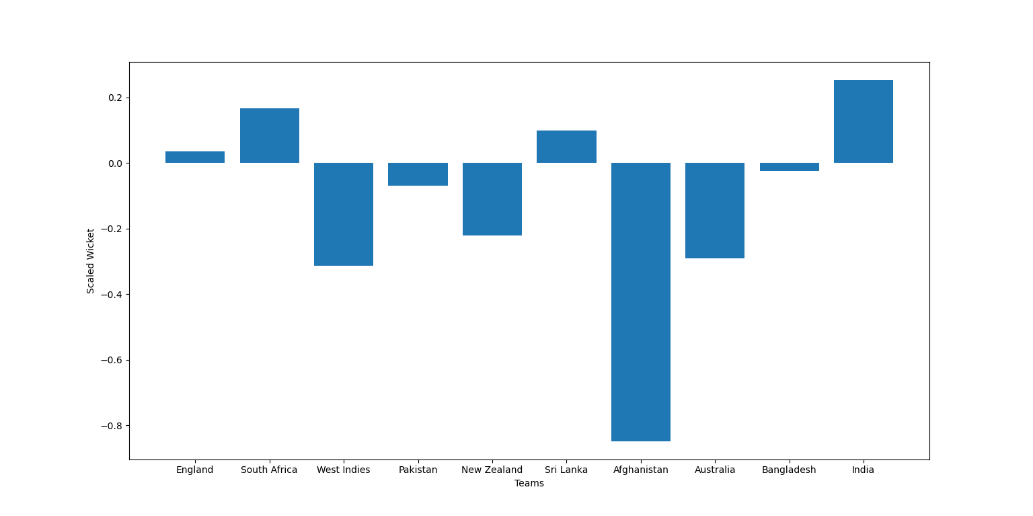


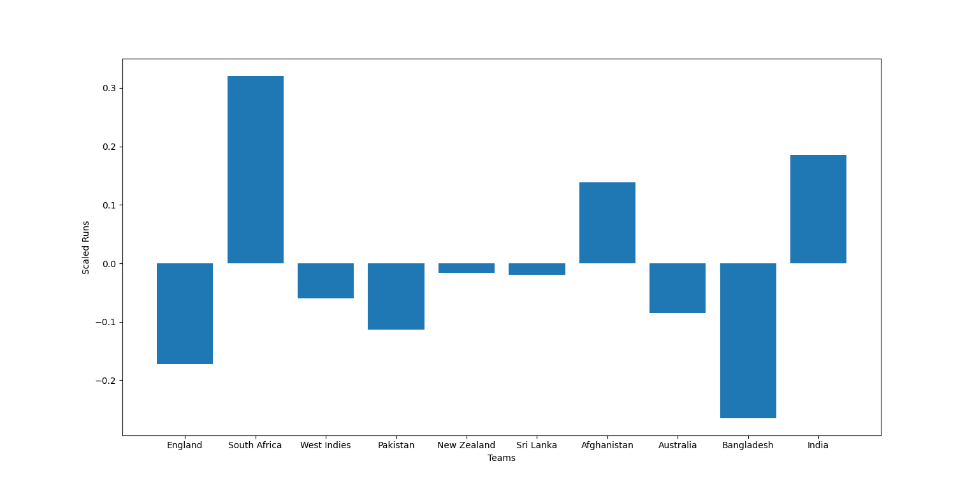
<Standard Scaling – Wicket and Runs>



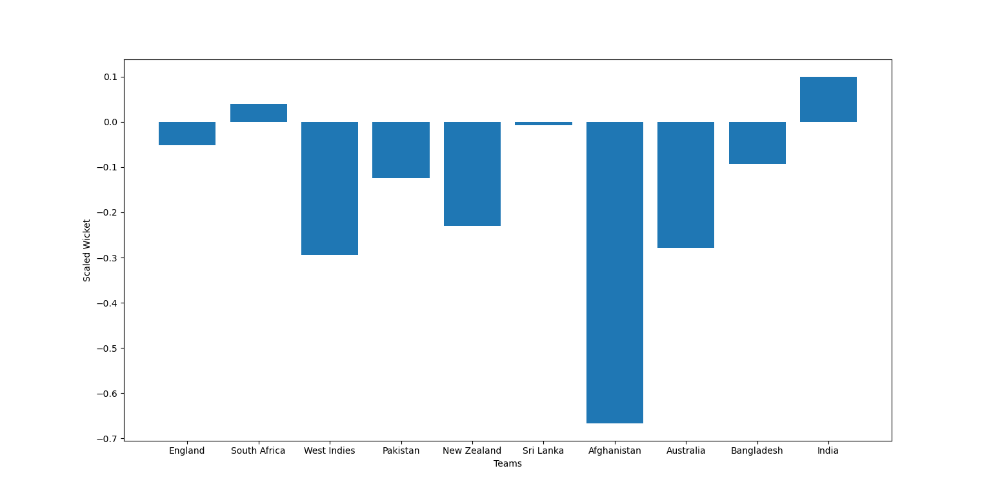


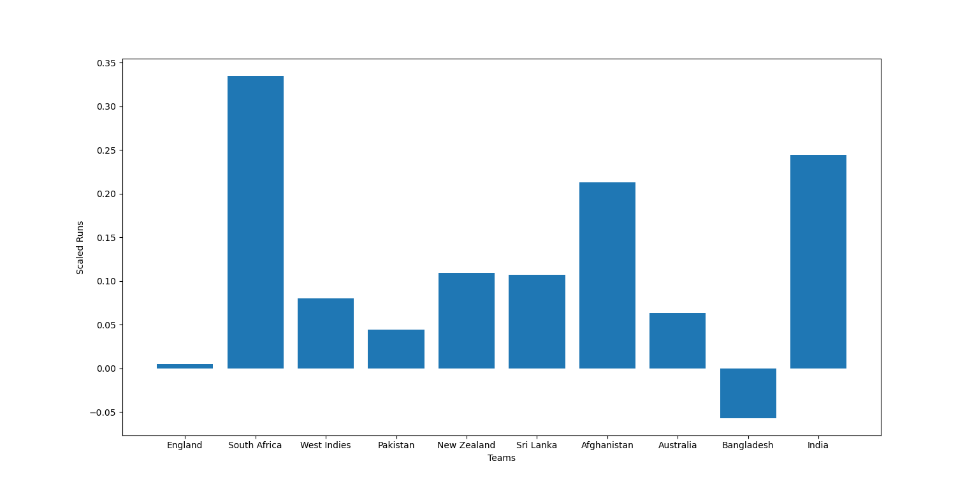
<MinMax Scaling – Wicket and Runs>



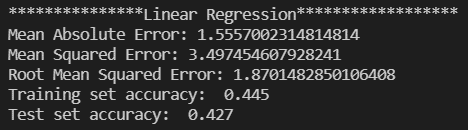


<Robust Scaling – Wicket and Runs>

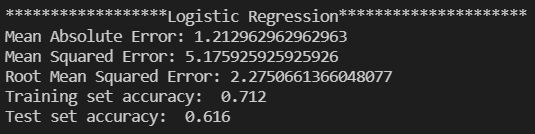




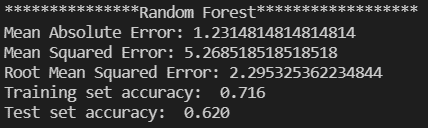
<Linear Regression>



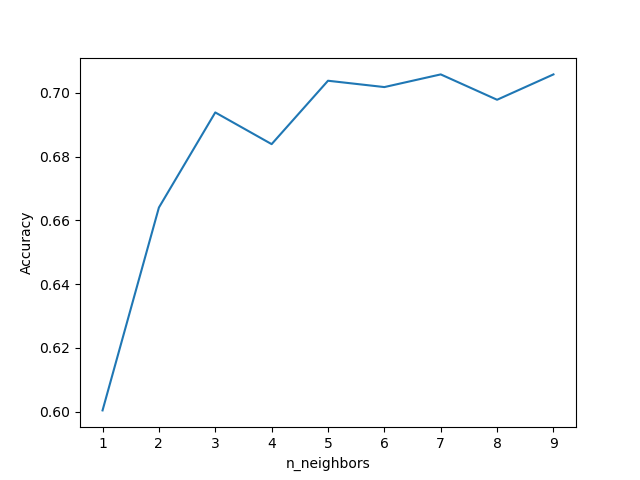
<Logistic Regression>



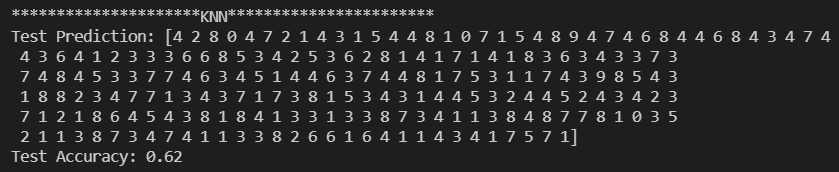
<Random Forest>



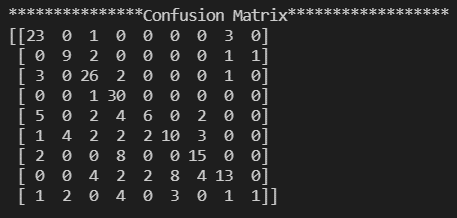
<KNN plot according to k value>

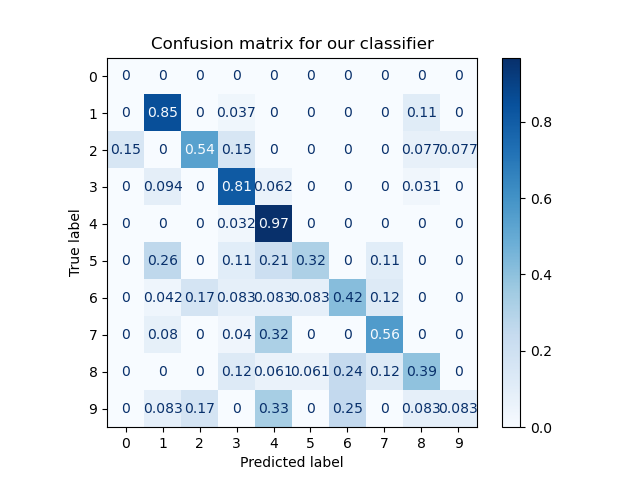


<Prediction and Accuracy of KNN (k = 7)>

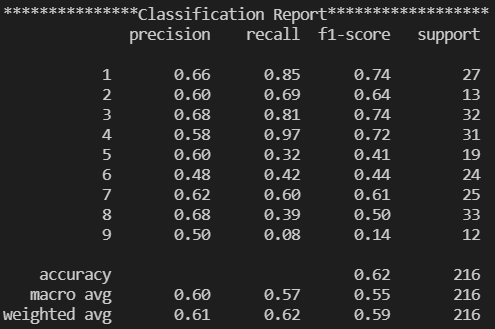


<Confusion Matrix – Logistic Regression>

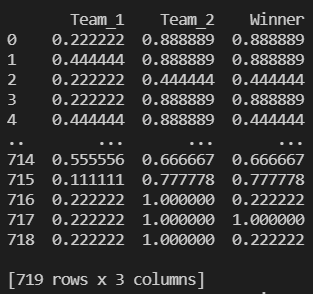




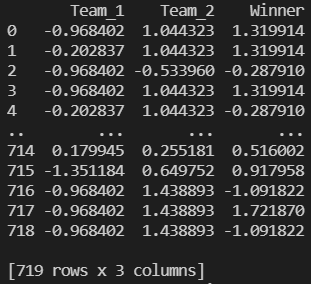
<Classification Report – Logistic Regression>



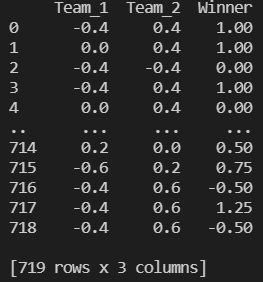
<MinMaxScaler result – df\_teams>



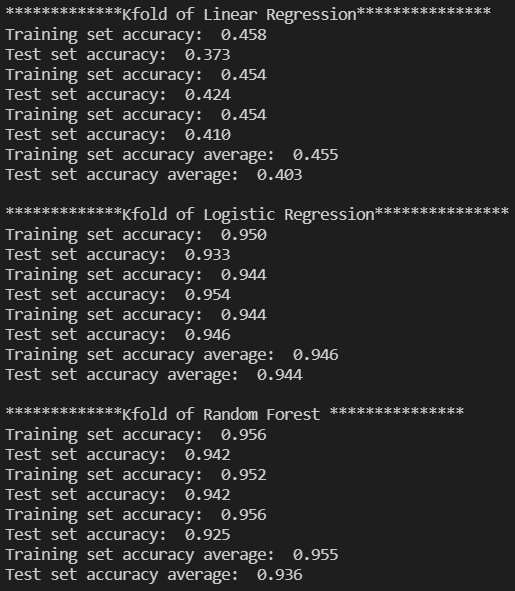
<StandardScaler result – df\_teams>



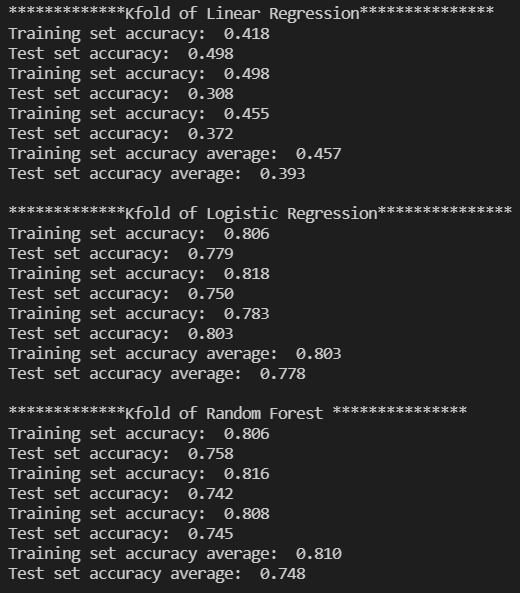
<RobustScaler result – df\_teams>



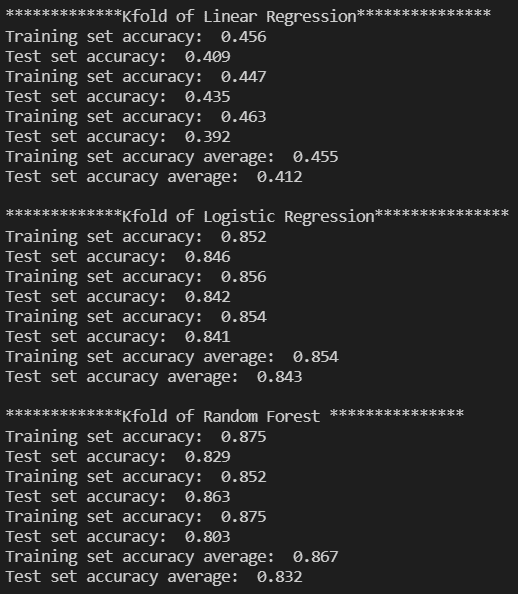
<MinMaxScaler & Kfold (Linear regression, Logistic Regression, Random Forest>



< StandardScaler & Kfold (Linear regression, Logistic Regression, Random Forest >



< RobustScaler & Kfold (Linear regression, Logistic Regression, Random Forest >



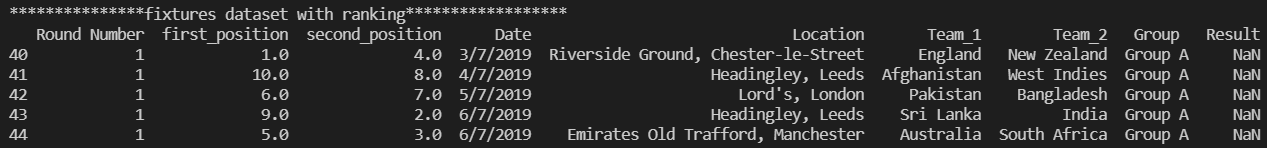
<Best Combination Accuracy>



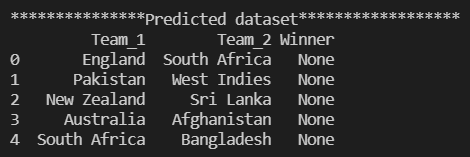
<Describe ‘icc\_rankings’ and ‘Fixtures’ dataset>



<Ranked using ‘icc\_rankings’ dataset>



<Top 5 of team to predict the match>



<Predicted result>





