# Q1. Demonstrate Missing value analysis and normalisation using sample data

#Importing Necessary Packages import pandas as pd

import numpy as np

#Creating Sample Data d= pd.DataFrame()

d['x0'] = [0.3051, 0.4949, 0.6974, 0.3769, 0.2231, 0.341, np.nan,

0.5897, 0.6308, np.nan]

d['x1'] = [np.nan, 20.2654, 15.2615, 17.5846, 12.4615, 15.8308,

14.4962, 17.3269, 18.5346, 21.6731]

#Checking for Table Description And Null values print(d.info())

#Counting Total Missing Values In Each Column print(d.isnull().sum())

**OUTPUT**

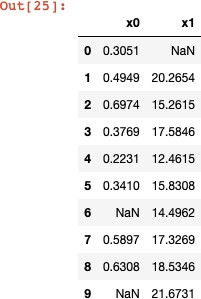
<class 'pandas.core.frame.DataFrame'> RangeIndex: 10 entries, 0 to 9

Data columns (total 2 columns): x0 8 non-null float64

x1 9 non-null float64 dtypes: float64(2)

memory usage: 240.0 bytes None

Out[23]:

x0 2

x1 1

dtype: int64

**#METHOD 1 : DROPPING MISSING VALUES**

### #CHECKING THE ORIGINAL SAMPLE

print (d)

#DROPPING THE ROWS WITH MISSING VALUES

### df=d.dropna() #CHECKING THE RESULT

print(df)

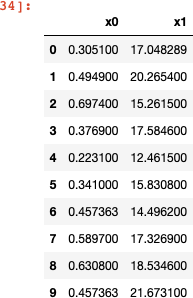
**#METHOD 2: FILLING MISSING VALUES WITH SUITABLE VALUES**

#FINDING THE MEAN FOR EACH COLUMN PRESENT

mean\_value=d.mean()

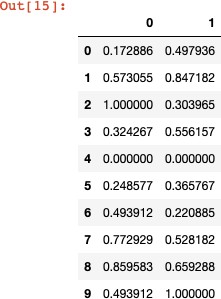
#REPLACING THE MISSING/NULL VALUES WITH THE AVERAGE/MEAN OF THEIR CORRESPONDING COLUMN

#MODIFYING AND SAVING CHANGES INTO THE SAME DATAFRAME

d.fillna(mean\_value,inplace=TRUE) #CHECKING THE RESULT

print(d)

#NORMALIZING THE VALUES AFTER TAKING CARE OF MISSING VALUES

x = d.values #returns a numpy array min\_max\_scaler = preprocessing.MinMaxScaler()

x\_scaled = min\_max\_scaler.fit\_transform(x) d = pd.DataFrame(x\_scaled)

d

**OUTPUT**

**Q2.Implement and visualise k- NN classifier. Evaluate the algorithm using any dataset of your choice from UCI**

**repository. Output should include accuracy, error rate, sensitivity, specificity, precision, recall.**

# K-Nearest Neighbors (K-NN)

# Importing the libraries import numpy as np

import matplotlib.pyplot as plt import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Social\_Network\_Ads.csv')

X = dataset.iloc[:, [2, 3]].values y = dataset.iloc[:, 4].values

# Splitting the dataset into the Training set and Test set from sklearn.cross\_validation import train\_test\_split X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.25, random\_state = 0)

# Feature Scaling

from sklearn.preprocessing import StandardScaler sc = StandardScaler()

X\_train = sc.fit\_transform(X\_train) X\_test = sc.transform(X\_test)

# Fitting K-NN to the Training set

from sklearn.neighbors import KNeighborsClassifier classifier = KNeighborsClassifier(n\_neighbors = 5, metric = 'minkowski', p = 2)

classifier.fit(X\_train, y\_train)

# Predicting the Test set results y\_pred = classifier.predict(X\_test)

# Making the Confusion Matrix

from sklearn.metrics import confusion\_matrix cm = confusion\_matrix(y\_test, y\_pred)

# Visualising the Test set results

from matplotlib.colors import ListedColormap X\_set, y\_set = X\_test, y\_test

X1, X2 = np.meshgrid(np.arange(start = X\_set[:, 0].min() - 1, stop

= X\_set[:, 0].max() + 1, step = 0.01),

np.arange(start = X\_set[:, 1].min() - 1, stop

= X\_set[:, 1].max() + 1, step = 0.01))

plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shape),

alpha = 0.75, cmap = ListedColormap(('red',

'green'))) plt.xlim(X1.min(), X1.max())

plt.ylim(X2.min(), X2.max())

for i, j in enumerate(np.unique(y\_set)): plt.scatter(X\_set[y\_set == j, 0], X\_set[y\_set == j, 1],

c = ListedColormap(('red', 'green'))(i), label =

j)

plt.title('K-NN (Test set)') plt.xlabel('Age') plt.ylabel('Estimated Salary') plt.legend()

plt.show()

#TO CALCULATE GIVEN METRICS USING CONFUSION MATRIX TP=cm[0][0]

FP=cm[1][0]

FN=cm[0][1]

TN=cm[1][1]

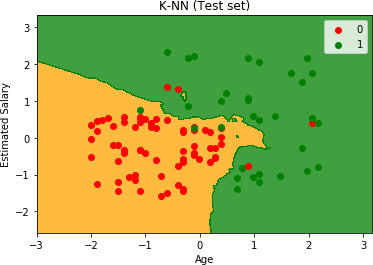
error\_rate=(FP+FN)/(TP+FP+TN+FN) acc=1-error\_rate sensitivity=TP/(TP+FN) specivity=TN/(TN+FP) pre=TP/(TP+FP)

recall=TP/(TP+FN)

print("error rate= ",error\_rate) print("accuracy= ", acc) print("sensitivity= ", sensitivity) print("specivity= ", specivity)

print("precision= ", pre) print("recall= ", recall)

## OUTPUT

error rate= 0.07 accuracy= 0.9299999999999999

sensitivity= 0.9411764705882353

specivity= 0.90625 precision= 0.9552238805970149

recall= 0.9411764705882353