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)
```

Out[25]:

	x0	x1
0	0.3051	NaN
1	0.4949	20.2654
2	0.6974	15.2615
3	0.3769	17.5846
4	0.2231	12.4615
5	0.3410	15.8308
6	NaN	14.4962
7	0.5897	17.3269
8	0.6308	18.5346
9	NaN	21.6731

```
#DROPPING THE ROWS WITH MISSING VALUES

df=d.dropna()

#CHECKING THE RESULT

print(df)
```

Out[26]:

	x0	x1
1	0.4949	20.2654
2	0.6974	15.2615
3	0.3769	17.5846
4	0.2231	12.4615
5	0.3410	15.8308
7	0.5897	17.3269
8	0.6308	18.5346

#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)
```

34]:

	x0	x1
0	0.305100	17.048289
1	0.494900	20.265400
2	0.697400	15.261500
3	0.376900	17.584600
4	0.223100	12.461500
5	0.341000	15.830800
6	0.457363	14.496200
7	0.589700	17.326900
8	0.630800	18.534600
9	0.457363	21.673100

```
#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

Out[15]:

	0	1
0	0.172886	0.497936
1	0.573055	0.847182
2	1.000000	0.303965
3	0.324267	0.556157
4	0.000000	0.000000
5	0.248577	0.365767
6	0.493912	0.220885
7	0.772929	0.528182
8	0.859583	0.659288
9	0.493912	1.000000

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
```