DATA MINING HW 3 – TEJAS (2000902539) NOTE: ANSWER FOR Q2 is at the end !!

Q1)

DATA MINING HW 3

Q S_1 = abc def abcdf

S_2 = bcd efabdcfp.

k = 3 Shingle of 3

Shingles of S_1 = (abc) (bcd) (cde) (def) (efa) (fab) ~~(abc)~~ copy ~~(bcd)~~ copy (cdf)

Shingles of S_2 = (bcd) (cde) (def) (efa) (fab) (abd) (bdc) (dcf) (cfp)

$S_1 \cap S^2$ = (bcd) (cde) (dcf) (efa) (fab) = 5

$S_1 \cup S_2$ = (abd) (bcd) (cde) (def) (efa) (fab) (cdf) (abd) (bdc) (dcf) (cfp)

Jaccard Similarity $= \frac{5}{11} \left[\frac{|S1 \cap S2|}{|S1 \cup S2|} \right]$

$= 0.45$

Hamming Code = no of bits / elements unique in S1 & S2

$= \underline{\underline{6}}$

Q 3

a) Yes we have performed Data processing steps before using the KNN Model for prediction.
Step 1:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb

#in Kaggle, File -> Add or upload data -> search for credit card
#note about the folder: ../input/creditcard
#change the folder if you have data in a different folder
data = pd.read_csv("../input/creditcard/creditcard.csv")
#data.head()
data.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e-15	2.010663e-15	-1.694249e-15	-1.927028e-16	-3.13
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.34
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.43
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.14
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559

As we see in step 1 we see that we have imported all libraries for future use. We have gone ahead and read the dataset and used descried to tell u more about the data.
Step 2:

```
#check if there are missing data
data.isnull().any().any()

#change 'Class' dtype to "bool"
data['Class'] = data['Class'].astype('bool')
```

```
class_zero = data.Class.value_counts().values[0]
class_one = data.Class.value_counts().values[1]
print(data["Class"].value_counts())
```

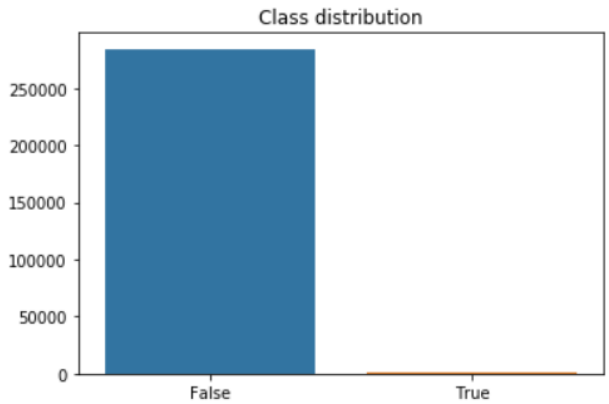
```
False    284315
True        492
Name: Class, dtype: int64
```

We have further gone ahead and checked for any null values or missing data. We have also converted every element of Class column into bool type to assess how many values are false or true (False:0 True:1). This step will ignore all NA.

Visual assessment of the effect of the code above:

```
sb.barplot(x=data.Class.value_counts().index.values, y=data.Class.value_counts().values)
plt.title("Class distribution")
```

```
Text(0.5, 1.0, 'Class distribution')
```



Step 3:

Feature scaling has been done using standardscaler from sklearn. Standardscaler gives us the mean.

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import metrics
data['AmountNormalized'] = StandardScaler().fit_transform(data['Amount'].values.reshape(-1, 1))
data['AmountNormalized'].describe()
```

Step4:

```
X = data.iloc[:, data.columns != 'Class'].values
y = data.iloc[:, data.columns == 'Class'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

Here we have split the data into test and train using model selection from sklearn. This will help us split data into unseen and seen and will improve the efficiency of the model.

b)

Metric : Minkowski Distance

(In a normed vector space it's a generalised form of Euclidean and Manhattan distance methods)

```
knn = KNeighborsClassifier(n_neighbors=5, metric= 'minkowski', p=2)
```

Here p= 2 means use Euclidean distance & p = 1 for Manhattan distance

c)

```
#KNN
from sklearn.neighbors import KNeighborsClassifier
#train
for i in range(1,6):

    knn = KNeighborsClassifier(n_neighbors=i, metric= 'minkowski', p=2)
    knn.fit(X_train, y_train.ravel())
    #test
    y_pred_knn = knn.predict(X_test)
    y_prob_knn = knn.predict_proba(X_test)

    evaluate_model(y_test, y_pred_knn, y_prob_knn[:, [1]], 'KNN ({})'.format(i))
```

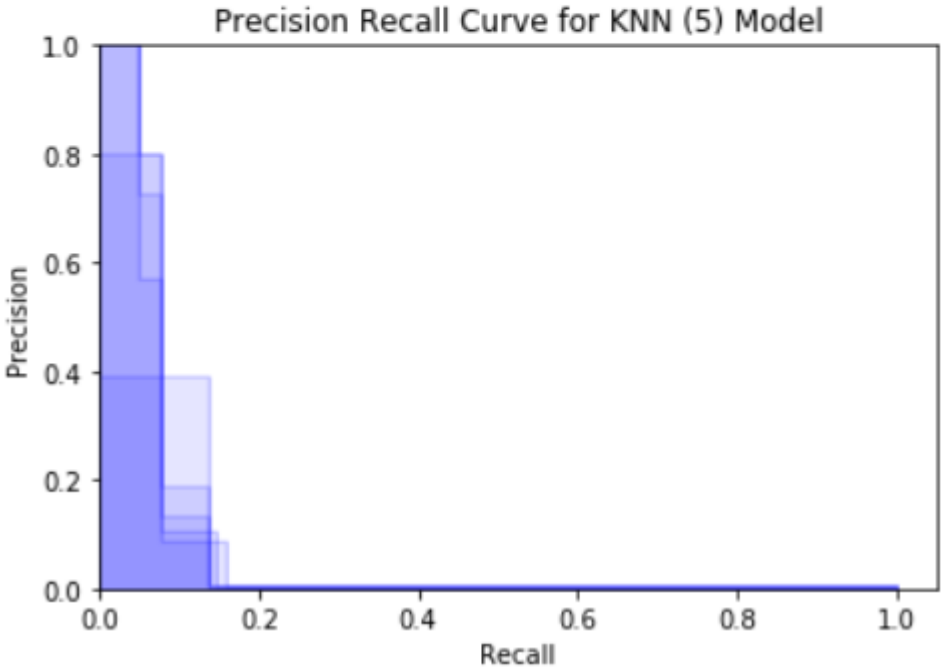
I have modified the code to add the for loop to get many evaluations done easily.

K = 2 has highest Accuracy that is 0.98332122. Max ROC: 0.577 at n = 5

```
Confusion Matrix for KNN (1) Model
[[56839    22]
 [   87    14]]
Classification Report for KNN (1) Model
              precision    recall  f1-score   support

       False   0.998472  0.999613  0.999042     56861
        True   0.388889  0.138614  0.204380       101

    accuracy                       0.998086     56962
   macro avg   0.693680  0.569113  0.601711     56962
weighted avg   0.997391  0.998086  0.997633     56962

Area under under ROC curve for KNN (1) Model
0.5691134764801642
Confusion Matrix for KNN (2) Model
[[56859     2]
 [   93     8]]
Classification Report for KNN (2) Model
              precision    recall  f1-score   support

       False   0.998367  0.999965  0.999165     56861
        True   0.800000  0.079208  0.144144       101

    accuracy                       0.998332     56962
   macro avg   0.899184  0.539586  0.571655     56962
weighted avg   0.998015  0.998332  0.997649     56962

Area under under ROC curve for KNN (2) Model
0.5688186808163942
Confusion Matrix for KNN (3) Model
[[56859     2]
 [   93     8]]
```

```
Classification Report for KNN (3) Model
              precision    recall  f1-score   support

       False   0.998367  0.999965  0.999165     56861
        True   0.800000  0.079208  0.144144       101

    accuracy                       0.998332     56962
   macro avg   0.899184  0.539586  0.571655     56962
weighted avg   0.998015  0.998332  0.997649     56962

Area under under ROC curve for KNN (3) Model
0.5685685485240106
Confusion Matrix for KNN (4) Model
[[56861     0]
 [   96     5]]
Classification Report for KNN (4) Model
              precision    recall  f1-score   support

       False   0.998315  1.000000  0.999157     56861
        True   1.000000  0.049505  0.094340       101

    accuracy                       0.998315     56962
   macro avg   0.999157  0.524752  0.546748     56962
weighted avg   0.998318  0.998315  0.997552     56962

Area under under ROC curve for KNN (4) Model
0.5732184146819036
Confusion Matrix for KNN (5) Model
[[56861     0]
 [   96     5]]
```

```
Classification Report for KNN (5) Model
              precision    recall  f1-score   support

       False   0.998315  1.000000  0.999157     56861
        True   1.000000  0.049505  0.094340       101

    accuracy                       0.998315     56962
   macro avg   0.999157  0.524752  0.546748     56962
weighted avg   0.998318  0.998315  0.997552     56962

Area under under ROC curve for KNN (5) Model
0.5777933195088735
```



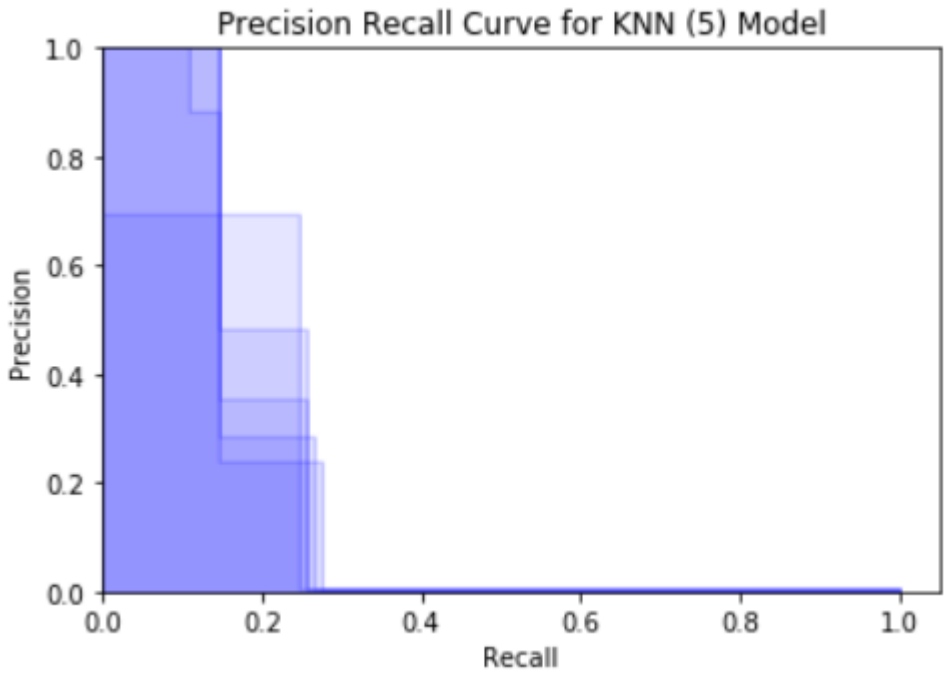
Now with p = 1

```python
#KNN
from sklearn.neighbors import KNeighborsClassifier
#train
for i in range(1,6):

    knn = KNeighborsClassifier(n_neighbors=i, metric= 'minkowski', p=1)
    knn.fit(X_train, y_train.ravel())
    #test
    y_pred_knn = knn.predict(X_test)
    y_prob_knn = knn.predict_proba(X_test)

    evaluate_model(y_test, y_pred_knn, y_prob_knn[:, [1]], 'KNN ({})'.format(i
```

```
Confusion Matrix for KNN (1) Model
[[56850    11]
 [   76    25]]
Classification Report for KNN (1) Model
              precision    recall  f1-score   support

       False   0.998665  0.999807  0.999235     56861
        True   0.694444  0.247525  0.364964       101

    accuracy                       0.998473     56962
   macro avg   0.846555  0.623666  0.682099     56962
weighted avg   0.998126  0.998473  0.998111     56962

Area under under ROC curve for KNN (1) Model
0.6236656491311712
```

```
Confusion Matrix for KNN (2) Model
[[56861     0]
 [   86    15]]
Classification Report for KNN (2) Model
              precision    recall  f1-score   support

       False   0.998490  1.000000  0.999244     56861
        True   1.000000  0.148515  0.258621       101

    accuracy                       0.998490     56962
   macro avg   0.999245  0.574257  0.628933     56962
weighted avg   0.998493  0.998490  0.997931     56962

Area under under ROC curve for KNN (2) Model
0.6285032233372297
Confusion Matrix for KNN (3) Model
[[56861     0]
 [   86    15]]
Classification Report for KNN (3) Model
              precision    recall  f1-score   support

       False   0.998490  1.000000  0.999244     56861
        True   1.000000  0.148515  0.258621       101

    accuracy                       0.998490     56962
   macro avg   0.999245  0.574257  0.628933     56962
weighted avg   0.998493  0.998490  0.997931     56962

Area under under ROC curve for KNN (3) Model
0.6283534748015875
```

```
Confusion Matrix for KNN (4) Model
[[56861     0]
 [   90    11]]
Classification Report for KNN (4) Model
              precision    recall  f1-score   support

       False   0.998420  1.000000  0.999209     56861
        True   1.000000  0.108911  0.196429       101

    accuracy                       0.998420     56962
   macro avg   0.999210  0.554455  0.597819     56962
weighted avg   0.998422  0.998420  0.997786     56962

Area under under ROC curve for KNN (4) Model
0.6331542213154503
Confusion Matrix for KNN (5) Model
[[56861     0]
 [   90    11]]
Classification Report for KNN (5) Model
              precision    recall  f1-score   support

       False   0.998420  1.000000  0.999209     56861
        True   1.000000  0.108911  0.196429       101

    accuracy                       0.998420     56962
   macro avg   0.999210  0.554455  0.597819     56962
weighted avg   0.998422  0.998420  0.997786     56962

Area under under ROC curve for KNN (5) Model
0.6379445202570591
```



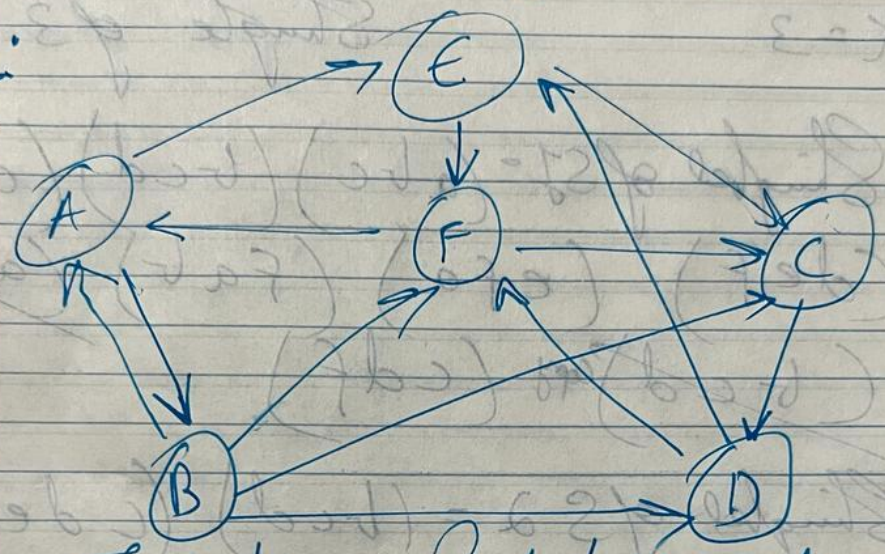
For p =1:

maximum accuracy of 0.9983322214809873 at K=2

Max roc at k = 5 : 0.6379

Q4)

Q4:



a) Transition Probability Matrix

$$M = \begin{array}{c} \\ A \\ B \\ C \\ D \\ E \\ F \end{array} \begin{array}{c} \begin{array}{cccccc} A & B & C & D & E & F \end{array} \\ \begin{bmatrix} 0 & 1/4 & 0 & 0 & 0 & 1/2 \\ 1/2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1/4 & 0 & 0 & 1/2 & 1/2 \\ 0 & 1/4 & 1 & 0 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 & 0 \\ 0 & 1/4 & 0 & 1/2 & 1/2 & 0 \end{bmatrix} \end{array}$$

b) The Sum of each Element of the Column had to be 1.

for Transition matrix M to be ~~S~~ stochastic

1 : $0 + 1/2 + 0 + 0 + 1/2 + 0 = 1$
2 : $1/4 + 0 + 1/4 + 0 + 1/4 + 1/4 = 1$
3 : $0 + 0 + 0 + 1 + 0 + 0 = 1$
4 : $0 + 0 + 0 + 0 + 1/2 + 1/2 = 1$
5 : $1/2 + 0 + 1/2 + 0 + 0 + 0 = 1$

Since Sum of all Value ~~in~~ every Colum is 1
M is stochastic

c) Assumption : Random Surfer is at A

Probability :

A to E $= 1/2$

E to C $= 1/2$

A to E to C $= 1/2 \times 1/2 = 1/4 = 0.25$

Prob of Surfer going from pag A to page E to page C is 0.25

Q 4)

d)

In [12]:

```
import numpy as np

n = int(input("Number of Nodes: "))
print("enter the values ")
#data = list(map(float,input().split()))
#M=np.array(data).reshape(n,n)
#print(M)
row = int(input("Enter number of rows (same as nodes): "))
cols = int(input("Enter number of cols (same as nodes): "))
list1 = []
for i in range(n):
    inputs = list(map(float,input().split()))
    list1.append(inputs)
M = np.array(list1).reshape(n,n)
print(M)
```

```
Number of Nodes: 6
enter the values
Enter number of rows (same as nodes): 6
Enter number of cols (same as nodes): 6
0 0.25 0 0 0 0.5
0.5 0 0 0 0 0
0 0.25 0 0 0.5 0.5
0 0.25 1 0 0 0
0.5 0 0 0.5 0 0
0 0.25 0 0.5 0.5 0
[[0.   0.25 0.   0.   0.   0.5 ]
 [0.5  0.   0.   0.   0.   0.  ]
 [0.   0.25 0.   0.   0.5  0.5 ]
 [0.   0.25 1.   0.   0.   0.  ]
 [0.5  0.   0.   0.5  0.   0.  ]
 [0.   0.25 0.   0.5  0.5  0.  ]]
```

In [13]:

```
import numpy as np

#print("enter the values ")
#data = list(map(float,input().split()))
#M=np.array(data).reshape(n,n)
#print(M)


max_iter=int(input("Maximum Iterations: "))

epsilon=1e-9
r = np.ones(n)/n
for _ in range(max_iter):
    r_last=r
    r = np.dot(M, r)
    if  np.sum(np.abs(r-r_last)) < epsilon:
        break

print(r)
```

```
Maximum Iterations: 2
[0.125      0.0625     0.20833333 0.22916667 0.16666667 0.20833333]
```

In [14]:

```
max_iter=int(input("Maximum Iterations: "))

epsilon=1e-9
r = np.ones(n)/n
for _ in range(max_iter):
    r_last=r
    r = np.dot(M, r)
    if  np.sum(np.abs(r-r_last)) < epsilon:
        break

print(r)
```

```
Maximum Iterations: 100
[0.12182741 0.06091371 0.20812183 0.22335025 0.17258883 0.21319797]
```

In [ ]:

```

```

Q2)

a) Jaccard Similarity $(d_1, d_2) = \frac{f_{11}}{f_{11} + f_{01} + f_{10}}$

$Sim(d_1, d_2) = \frac{1}{1+3+2} = 1/6$

$Sim(d_1, d_3) = \frac{1}{1+0+2} = 1/3$

$Sim(d_1, d_4) = \frac{2}{2+1+1} = 2/4 = 1/2$

$Sim(d_1, d_5) = \frac{2}{2+1+2} = 2/5$

$Sim(d_1, d_6) = \frac{2}{2+0+1} = 2/3$

$Sim(d_2, d_3) = 0$

$Sim(d_2, d_4) = \frac{2}{5}$

$Sim(d_2, d_5) = 3/5$

$Sim(d_2, d_6) = 1/5$

$Sim(3, 4) = 0$

$Sim(3, 5) = 0$

$Sim(3, 6) = 0$

$Sim(4, 5) = \frac{3}{4}$

$Sim(4, 6) = 2/3$

$Sim(5, 6) = \frac{2}{4} = 1/2$

Index: 0 1 2 3 4 5

b) d_1 — 0 1 0 1 1 0

$h_1(x) = (2x+1) \% 6$

$h_1(1) = 3 \% 6 = 3$

$h_1(3) = 7 \% 6 = 1$

$h_1(4) = 9 \% 6 = 3$

Minimum Sig $h_1(x) = 1$

from $h_2(x) = (3x+2) \% 6$

$h_2(1) = 5 \% 6 = 5$

$h_2(3) = 11 \% 6 = 5$

$h_3(4) = 14 \% 6 = 2$

Minimum Sig for $h_2(x) = 2$

from $h_3(x) = (5x+2) \% 6$

$h_3(1) = 7 \% 6 = 1$

$h_3(3) = 17 \% 6 = 5$

$h_3(4) = 22 \% 6 = 4$

from $h_4(x) = (7x+3) \% 6$

$h_4(1) = 10 \% 6 = 4$

$h_4(3) = 24 \% 6 = 0$

$h_4(4) = 31 \% 6 = 1$

Vëmbasës Sëpaj $h_a(x) = 0$.

(ii) $h_1(x) = (2x+1) \% 6)$

$h_1(0) = 1$

$h_1(2) = 5 \% 6 = 5$

$h_1(3) = 7 \% 6 = 1$

$h_1(5) = 11 \% 6 = 5$

Sëpaj $\Rightarrow h_1(x) = 1$

$h_2(0) = 2$

$h_2(2) = 2$

$h_2(3) = 11 \% 6 = 5$

$h_2(5) = 17 \% 6 = 5$

Sëpaj $h_2(x) = 2$

$h_3(0) = 2 \% 6 = 2$

$h_3(2) = 0$

$h_3(3) = 17 \% 6 = 5$

$h_3(5) = 27 \% 6 = 3$

Sig $h_3(n) = 0$.

$h_4(6) = 3$

$h_4(2) = 5$

$h_4(3) = 24 \% 6 = \underline{\underline{0}}$.

$h_4(5) = 38 \% 6 = 2$

Sig $h_4(x) = 0$

iii) $d_3 = 0 \quad 1 \; 0 \; 0 \; 0 \; 0$

$h_1(1) = 3$

Sig $h_1(x) = \underline{\underline{3}}$.

$h_2(x) = 5 \% 6 = 5$

$h_2(n) = 5$

Sig: $h_2(n) = \underline{\underline{5}}$.

$h_3(n) = (5n+2) \% 6$

$h_3(1) = 7 \% 6 = 1$

Sig $h_3(n) = 1$

$h_4(n) = (7n+3) \% 6 = 1$

$h_4(1) = 10 \% 6 = 4$

Sig $h_4(n) = \underline{\underline{4}}$

iv) $d = 1\ 0\ 0\ 1\ 1\ 1$

$(2n+1) \% 5$

$h_1(0) = 1$

$h_1(3) = 1$

$h_1(4) = 3$

$h_1(5) = 5$

Sig $h_1(n) = \underline{\underline{1}}$

$h_2(0) = 2$

$h_2(3) = 5$

$h_2(4) = 2$

$h_2(5) = 5$

Sig $h_2(n) = \underline{\underline{2}}$

B

$h_3(0) = 2$

$h_3(3) = 5$

$h_3(4) = 4$

$h_3(5) = 3$

Sig $h_3(x) = 2$

$h_4(0) = 3$

$h_4(3) = 6$

$h_4(4) = 1$

$h_4(5) = 2$

Sig $h_4(x) = 0$

$d_4 = 100110$

$h_1(0) = 1 \% 6 = 1$

$h_1(3) = 7 \% 6 = 1$

$h_1(4) = 9 \% 6 = 3$

Sig $h_1(x) = 1$

$h_2(0) = 2$

$h_2(3) = 5$

$h_2(4) = 2$

Sig $h_2(x) = 2$

$h_3(0) = 2$

$h_3(3) = 5$

$h_3(4) = 4$

Sig $h_3(n) = \underline{\underline{2}}$

$h_4(0) = 3$

$h_4(3) = 0$

$h_4(4) = 1$

Sig $h_4(n) = \underline{\underline{0}}$

ii) $d = 0 \quad 0 \quad 0 \quad \| \quad 0$

$h_1(3) = 1$

$h_1(4) = 3$

Sig $h_1(n) = 1$

$h_2(3) = 5$

$h_2(4) = 2$

Sig $h_2(n) = 2$

$h_3(3) = 5$

$h_3(4) = 4$

Sig $h_3(n) = \underline{\underline{4}}$

$h_4(3) = 0$
$h_4(4) = 1$

So $h_4(x) = 0$.

Sig Matrix

	d_1	d_2	d_3	d_4	d_5	d_6
$h_1(x)$	1	1	3	1	1	1
$h_2(x)$	2	2	5	2	2	2
$h_3(x)$	1	0	1	2	2	4
$h_4(x)$	0	0	4	0	0	0

c)

	0	1	2	3	4	5
$h_1(x) = (2x+1)\%6$	1	3	5	1	3	5
$h_2(x) = (3x+2)\%6$	2	5	2	5	2	5
$h_3(x) = (5x+2)\%6$	2	1	0	5	4	3
$h_4(x) = (7x+3)\%6$	3	4	5	0	1	2

Only h_3 & h_4 give unique values for each column.

Hence they are <u>Two Permutation</u>

d) Sig M

	d_1	d_2	d_3	d_4	d_5	d_6
$h_1(x)$	1	1	3	1	1	1
$h_2(x)$	2	2	5	2	2	2
$h_3(x)$	1	0	1	2	2	1
$h_4(x)$	0	0	4	0	0	0

$$Sim(x_1, x_2) = \frac{x_1 \wedge x_2}{x_1 \vee x_2}$$

$Sim(d_1, d_2) = 3/4$

$Sim(d_1, d_3) = 1/4$

$Sim(d_1, d_4) = 3/4$

$Sim(d_1, d_5) = 3/4$

$Sim(d_1, d_6) = 3/4$

$Sim(d_2, d_3) = 0$

$Sim(d_2, d_4) = 3/4$

$Sim(d_2, d_5) = 3/4$

$Sim(d_2, d_6) = 3/4$

$Sim(d_3, d_4) = 0$

$Sim(d_3, d_5) = 0$

$Sim(d_3, d_4) = 0$

$Sim(d_4, d_5) = 1$

$Sim(d_4, d_6) = 3/4$

$Sim(d_5, d_6) = 3/4$

~~Minhash provides good signatures for such documents. In our case similarity was $0.75 \approx (\frac{3}{4})$~~

c) No, Minhash is not good for such documents.
Similarity here solved above is 0.75 while documents similarity varies between 0.165 – 0.75

~~Difference is lower in a~~

f) Yes, the other design that gives true permutation is $(7x+1) \% 6$.

	0	1	2	3	4	5
$(17x+1) \% 6$	1	8	5	3	4	1

$(7x+1) \% 6$	1	2	3	4	5	0