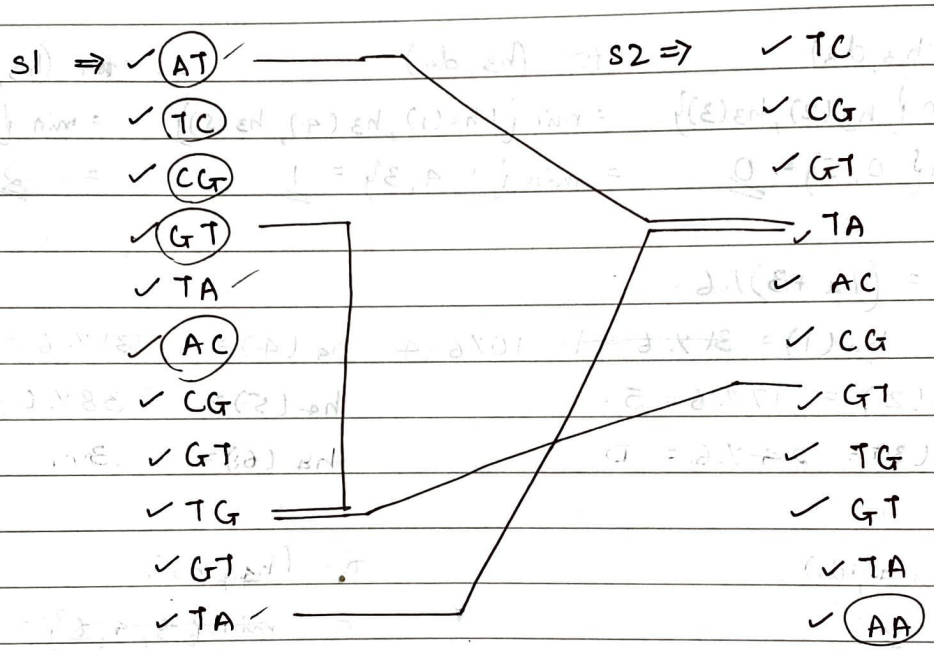
Q1) S1 = A T C G T A C G T G T A

S2 = T C G T A C G T G T A A

(a) Hamming dist = no of positions where characters do-not match. = 11

(b) for S1, S2 & len(shingle) = 2.



not considering repetations or ~~or~~ internal order of characters

shingles	S1	S2
AT	1	1
TC	1	1
CG	1	1
GT	1	1
AC	1	1
AA	0	1

Jaccard similarity

$$= \frac{|S1 \cap S2|}{|S1 \cup S2|}$$

$$= \frac{5}{6}.$$

Jaccard distance.

$= 1 - J_s$

$= 1 - 5/6 = 1/6$

considering 'AT' & 'TA'... as different shingle.

	S1	S2
AT	1	0
TC	1	1
CG	1	1
GT	1	1
TA	1	1
AC	1	1
~~CG~~	-	-
~~GT~~	-	-
TG	1	1
GT	1	1
~~TA~~	-	-
AA	0	1

Jaccard Similarity in this case = $\frac{7}{9}$.

(c) Hamming distance measures the no of alterations (only replacements) that can be done so that both of the strings are similar. Hence I this this measure measures the dissimilarity & it needs the two strings to be of same length, $H_d = 11$ i.e almost all alphabets are dissimilar.

Jaccard Similarity will measure how the two strings are similar here ignoring the '00' matches. Hence JS avoids the strings that are not present in both because they are of no help in calculating similarity.

As I can notice, just shifting the 1st element towards last in S1 will make both the strings S1 & S2 exactly the same. Hence the similarity should be high i.e dissimilarity should be low (logically).

But as we can see Hamming Distance ~~the~~ = 11, i.e almost all are dissimilar where as Jaccard Similarity is close to 1 i.e highly similar. Hence acc to me, in this case Jaccard Similarity is a good measure of similarity between the two strings. Hamming distance would be more useful in cases where there are few errors i.e typescript errors like 'error' written as 'werror'. But for similarity I think Jaccard similarity will perform well for almost all cases. but will take comparatively more time than Hamming.

(d) [next page] ⇒

(d) Consider Shingling without repetations. of size = 2.

given: S1 has n elements/alphabets
S2 has n alphabets

∴ no of shingles for S1 = n-1. , S2 = n-1.
considering that S1 & S2 has no shingles in common
Total no of shingles become 2(n-1).

Table will be:

	S1	S2.
1	☐	☐
2	☐	☐
⋮		
2(n-1)	☐	☐

Step by step proces:- (1) create shingle. — Time taken: 1 = ~~linear~~ Const
(2) Delete duplicates — 1 = Constant
(3) ~~Compute~~ assign 0/1 or form matrix — 1 = constant time
(4) Compute Jaccard similarity — 2(n-1) times.

we will iterate over 2(n-1) rows. &

2(n-1) times. { if S1 != 0 or S2 != 0 ⇒ union = union + 1.
if S1 = 1 and S2 = 1 ⇒ intersection = inter + 1

1 = Constant time — (5) Compute Jaccard similarity = $\frac{\text{inters}}{\text{union}}$

Total time = 1 + 1 + 1 + 2(n-1) + 1
= 2(n-1) + Constant.

∴ Order of equation is n. Time complexity = O(n).

changing the start point to 1 to simplify counting

Shingle	ID	d1	d2	d3	d4	d5	d6
0	0	0	0	0	1	0	0
1	1	0	0	1	0	0	0
2	2	1	1	1	0	0	0
3	3	1	1	0	1	1	1
4	4	1	0	0	1	1	1
5	5	0	1	0	0	1	1

2)

(a) Jaccard Similarity $= \frac{|d1 \cap d2|}{|d1 \cup d2|} = \frac{\text{Intersection}}{\text{Union}}$.

$\text{Sim}(d1, d2) = \frac{2}{4} = \frac{1}{2}$

$\text{Sim}(d2, d1)$

$\text{Sim}(d1, d3) = \frac{1}{4}$

$\text{Sim}(d3, d1)$

$\text{Sim}(d1, d4) = \frac{2}{4} = \frac{1}{2}$

$\text{Sim}(d4, d1)$

$\text{Sim}(d1, d5) = \frac{2}{4} = \frac{1}{2}$

$\text{Sim}(d5, d1)$

$\text{Sim}(d1, d6) = \frac{2}{4} = \frac{1}{2}$

$\text{Sim}(d6, d1)$

$\text{Sim}(d2, d3) = \frac{1}{4}$

$\text{Sim}(d2, d5) = \frac{2}{4} = \frac{1}{2}$

$\text{Sim}(d2, d4) = \frac{1}{5}$

$\text{Sim}(d2, d6) = \frac{2}{4} = \frac{1}{2}$

$Sim(d3, d4) = \frac{0}{5} = 0$ $\quad Sim(d4, d5) = \frac{2}{4} = \frac{1}{2}$

$Sim(d3, d5) = \frac{0}{5} = 0$ $\quad Sim(d4, d6) = \frac{2}{4} = \frac{1}{2}$

$Sim(d3, d6) = \frac{0}{5} = 0$ $\quad Sim(d5, d6) = \frac{3}{3} = 1.$

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

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

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

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

$h_1(1) = (2+1) \% 6 = 3.$

$h_1(2) = (3+2) \% 6 = 5$ ~~$(6+2) \% 6 = 2.$~~

$h_1(3) = (8+2) \% 6 = 5$

$h_1(4) =$

$h_1(5) =$

$h_1(6) =$

* for (h_1, d_1).

$h_1(1) = (2+1) \% 6 = 3$

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

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

$h_1(4) = (8+1) \% 6 = 3$

$h_1(5) = (10+1) \% 6 = 5$

$h_1(6) = (42+3) \% 6 = 3.$

$(h_1, d_1) = \min\{h_1(5), h_1(3), h_1(4)\}$

$= \min\{5, 1, 3\}$

$= 1.$

for (h_1, d_2)

$(h_1, d_2) = \min\{h_1(2), h_1(3), h_1(5)\}$

$= \min\{5, 1, 5\}$

$= 1.$

for h_1, d_3.

$(h_1, d_3) = \min\{h_1$

D1) for d1. , consider [2, 3, 4].

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

$h_1(2) = 5$ $h_1(3) = 1$ $h_1(4) = 3$.

min = 1.

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

$h_2(2) = 2$, $h_2(3) = 5$, $h_2(4) = 2$

min = 2

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

$h_3(2) = 0$ $h_3(3) = 5$ $h_3(4) = 4$

min = 0

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

$h_4(2) = 5$ $h_4(3) = 0$ $h_4(4) = 1$

min = 0.

D2) consider 2, 3, 5 for D2.

$h_1(2) = 5$ $h_1(3) = 1$ $h_1(5) = 5$

min = 1

$h_2(2) = 2$ $h_2(3) = 5$ $h_2(5) = 5$

min = 2

$h_3(2) = 0$ $h_3(3) = 5$ $h_3(5) = 3$

min = 0

$h_4(2) = 5$ $h_4(3) = 0$ $h_4(5) = 2$

min = 0

D3) consider 1, 2.

values for $h_1 \Rightarrow$ 3, 5 min = 3
$h_2 \Rightarrow$ 5, 2 min = 2
$h_3 \Rightarrow$ 1, 0 min = 0
$h_4 \Rightarrow$ 4, 5. min = 4

D4) consider. 0, 3, 4.

values will be. $h_1 \Rightarrow$ 1, 1, 3 min = 1
$h_2 \Rightarrow$ 2, 5, 2. min = 2
$h_3 \Rightarrow$ 2, 5, 4 min = 2
$h_4 \Rightarrow$ 3, 0, 1 min = 0

D6, D5) taking x as 3, 4, 5. substitue in $h_n(x)$. we get

$h_1 \Rightarrow$ 1, 3, 5 min = 1
$h_2 \Rightarrow$ 5, 2, 5 min = 2
$h_3 \Rightarrow$ 5, 4, 3 min = 3
$h_4 \Rightarrow$ 0, 1, 2. min = 0

Min hash signature matrix $\Rightarrow$

	d1	d2	d3	d4	d5	d6.
h_1	1	1	3	1	1	1
h_2	2	2	2	2	2	2
h_3	0	0	0	2	3	3
h_4	0	0	4	0	0	0.

calculating & substituting similarities in the column below

	word doc matrix	min-hash.	
$d_1 d_2$	0.5	2/2 = 1	
$d_1 d_3$	0.25	1/3 = 0.33	} closer
$d_1 d_4$	0.5	2/3 = 0.66	
$d_1 d_5$	0.5	2/3 = 0.66	
$d_1 d_6$	0.5	2/3 = 0.66	
$d_2 d_3$	0.25	1/3 = 0.33	closer
$d_2 d_4$	0.2	2/3 = 0.66	
$d_2 d_5$	0.5	2/3 = 0.66	} closer
$d_2 d_6$	0.5	2/3 = 0.66	
$d_3 d_4$	0	1/4 = 0.25	} close
$d_3 d_5$	0	1/4 = 0.25	
$d_3 d_6$	0	1/4 = 0.25	
$d_4 d_5$	0.5	2/3 = 0.66	} closer.
$d_4 d_6$	0.5	2/3 = 0.66.	
$d_5 d_6$	1	3/3 = 1	} same

(d) Out of 15 similarities, (1) is exactly the same (7%), ~~7 are almost the same (60%). &~~ 9 (45/60%) are almost the same (60%), 3 (20%) are similar. i.e around 90% of the similarities are very close to each other. So here min hash does not provide exactly the same results but the relative similarity is consistent in it. for eg. $d_1 d_3$ is less similarity score than $d_1 d_4$ for both of the matrices (the non-min hased one & min hashedone) yes minhash provides good resemblance of similarity, considering the time it takes, the results are very similar & represents the actual data.

3) (a) Yes pre-processing steps are done

- Interer data for 'class' is converted to boolean data type.
- Standard scaling is done for Amount where the new mean for it will be 0 and Standard Deviation will be 1. & for the same Amount column is passed as: (n rows & 1 column).
- Train Test split is also done with 20% of the total data as test.
- Since power = 2, euclidean distance is used.
- Also we have checked for the missing values in data.

KNN script(default)

In [16]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sb
from sklearn.neighbors import KNeighborsClassifier

#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()
```

Out[16]:

	Time	V1	V2	V3	V4	
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+
mean	94813.859575	3.919560e-15	5.688174e-16	-8.769071e-15	2.782312e-15	-1.552563e
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.43358
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e-

8 rows × 31 columns

In [5]:
```
data['Class'].value_counts()
```

Out[5]:
```
0    284315
1       492
Name: Class, dtype: int64
```

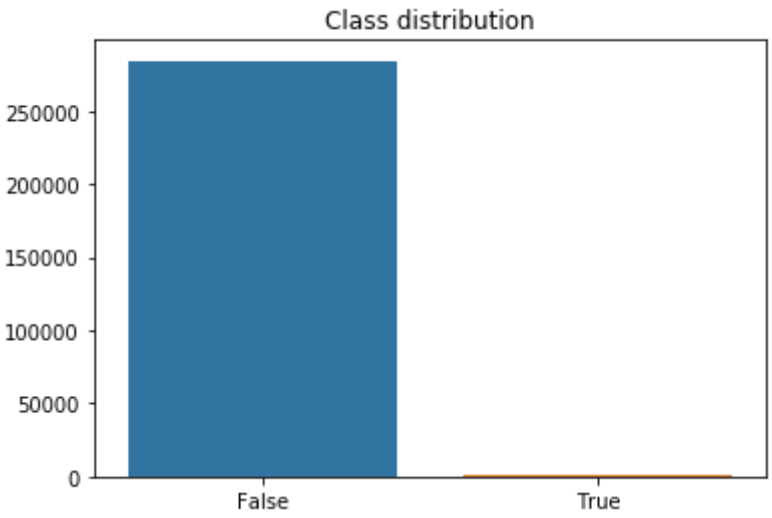
In [6]:
```
#check if there are missing data
data.isnull().any().any()

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

In [7]:
```
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
```

In [8]:
```
sb.barplot(x=data.Class.value_counts().index.values, y=data.Class.value_counts(
plt.title("Class distribution")
```

Out[8]:
```
Text(0.5, 1.0, 'Class distribution')
```



In [9]:
```
data['Amount']
```

Out[9]:
```
0         149.62
1           2.69
2         378.66
3         123.50
4          69.99
           ...
284802      0.77
284803     24.79
284804     67.88
284805     10.00
284806    217.00
Name: Amount, Length: 284807, dtype: float64
```

In [10]:
```
data['Amount'].shape
```

Out[10]:
```
(284807,)
```

In [11]:
```
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
data['AmountNormalized'].describe()
```

Out[11]:
```
count    2.848070e+05
mean     3.202236e-16
std      1.000002e+00
min     -3.532294e-01
25%     -3.308401e-01
50%     -2.652715e-01
75%     -4.471707e-02
max      1.023622e+02
Name: AmountNormalized, dtype: float64
```

In [12]:
```
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, rand
```

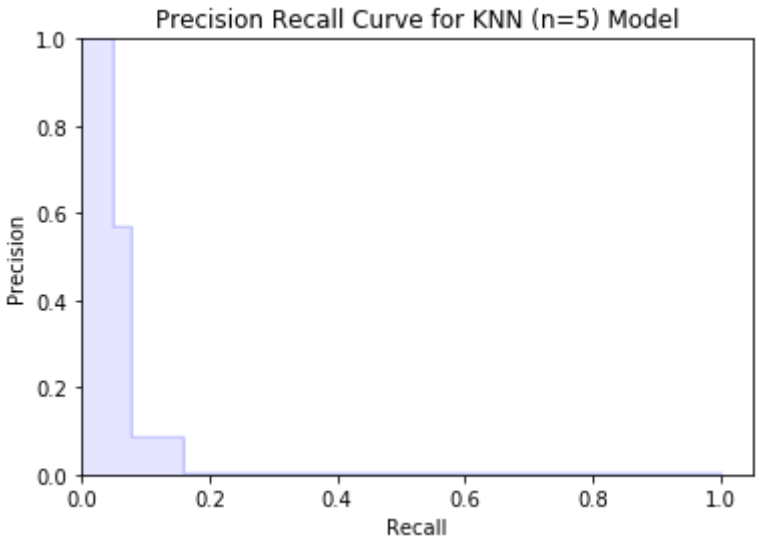
In [13]:
```
def plot_precision_recall_curve(y_actual, y_score, model_name):
    precision, recall, _ = metrics.precision_recall_curve(y_actual, y_score)
    curve_data = pd.DataFrame(columns = range(0, len(precision)))
    curve_data.loc['Precision'] = precision
    curve_data.loc['Recall'] = recall
    #print (curve_data)
    plt.step(recall, precision, color='b', alpha=0.1, where='post')
    plt.fill_between(recall, precision, step='post', alpha=0.1, color='b')
    plt.title('Precision Recall Curve for {} Model'.format(model_name))
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.xlim([0, 1.05])
    plt.ylim([0, 1.0])
    plt.show()

def evaluate_model(y_actual, y_pred, y_score, model_name):
    cm = metrics.confusion_matrix(y_actual, y_pred)
    print ('Confusion Matrix for {} Model'.format(model_name))
    print (cm)
    print ('Classification Report for {} Model'.format(model_name))
    print (metrics.classification_report(y_actual, y_pred, digits=6))
    print ('Area under under ROC curve for {} Model'.format(model_name))
    print (metrics.roc_auc_score(y_actual, y_score))
    plot_precision_recall_curve(y_actual, y_score, model_name)
```

In [42]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(227845, 31)
(227845, 1)
(56962, 31)
(56962, 1)
```

In [15]:
```
#KNN
#train
knn = KNeighborsClassifier(n_neighbors=5, 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)

print(knn.score(X_test, y_test))
evaluate_model(y_test, y_pred_knn, y_prob_knn[:, [1]], 'KNN (n=5)')
```

```
0.9983146659176293
Confusion Matrix for KNN (n=5) Model
[[56861     0]
 [   96     5]]
Classification Report for KNN (n=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 (n=5) Model
0.5777933195088735
```

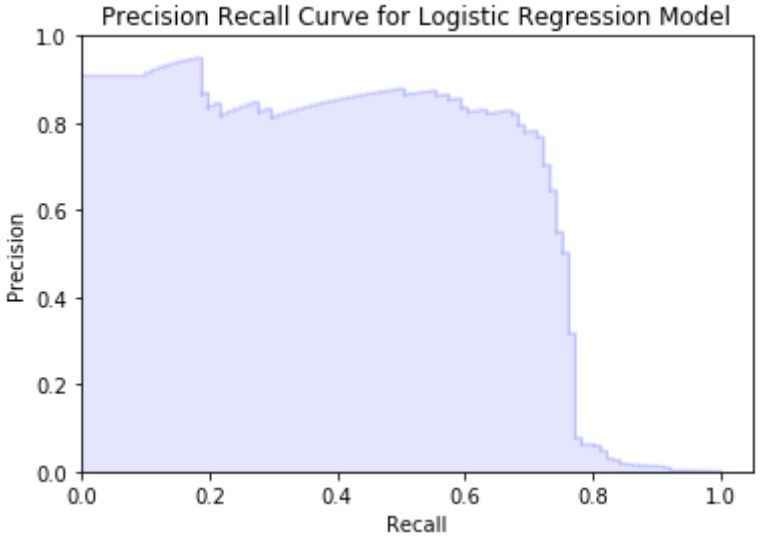


In [44]:

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression(solver='lbfgs')
#note y_train.ravel()
lr.fit(X_train, y_train.ravel())
y_pred_lr = lr.predict(X_test)
y_score_lr = lr.decision_function(X_test)
y_prob_lr = lr.predict_proba(X_test)

evaluate_model(y_test, y_pred_lr, y_prob_lr[:,[1]], 'Logistic Regression')
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/linear_model/logistic.py:947: C
onvergenceWarning: lbfgs failed to converge. Increase the number of iteration
s.
  "of iterations.", ConvergenceWarning)
```

```
Confusion Matrix for Logistic Regression Model
[[56836    25]
 [   28    73]]
Classification Report for Logistic Regression Model
              precision    recall  f1-score   support

       False   0.999508  0.999560  0.999534     56861
        True   0.744898  0.722772  0.733668       101

    accuracy                       0.999070     56962
   macro avg   0.872203  0.861166  0.866601     56962
weighted avg   0.999056  0.999070  0.999063     56962

Area under under ROC curve for Logistic Regression Model
0.9331238711180523
```



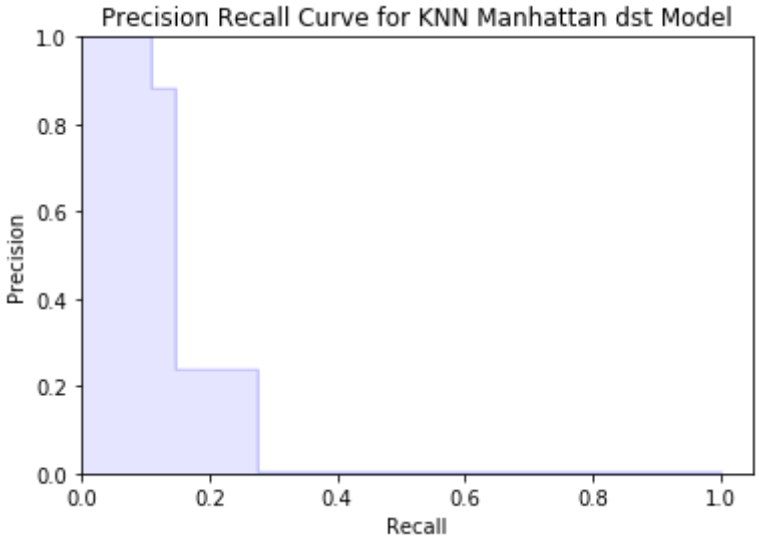
Modifying distance metric

Trying put various distance measures to check how the performance of KNN varies. Trying out manhattan distance, euclidean distance and minkwoski distance below, and saving the precision and recall in an array for further plotting. Here I think that precion and recall should be more important metric than accuracy becaise of the class imbalance, accuracy can't represent both of the classes equally. RecallTP/TP+TN. Ideally a tradeoff between Precision and Recall is what we need to consider, but we alsways aim for both High Precision and High Recall

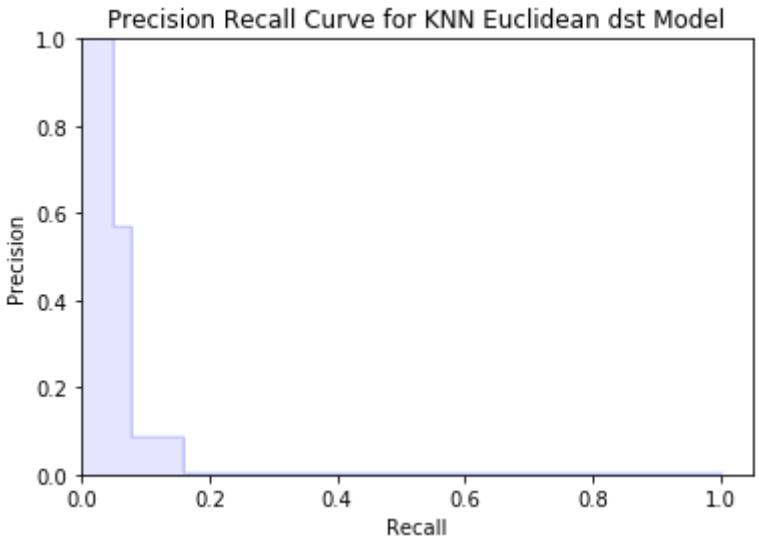
In [45]:
```
from sklearn import metrics
metric = []
# manhattan distance
knn = KNeighborsClassifier(n_neighbors=5, metric= 'minkowski', p=1)
knn.fit(X_train, y_train.ravel())
#test
y_pred_knn = knn.predict(X_test)
y_prob_knn = knn.predict_proba(X_test)
metric.append(metrics.precision_score(y_test, y_pred_knn))
metric.append(metrics.recall_score(y_test, y_pred_knn))

evaluate_model(y_test, y_pred_knn, y_prob_knn[:, [1]], 'KNN Manhattan dst')
```

```
print(1)
# euclidean distance
knn = KNeighborsClassifier(n_neighbors=5, 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)
metric.append(metrics.precision_score(y_test, y_pred_knn))
metric.append(metrics.recall_score(y_test, y_pred_knn))

evaluate_model(y_test, y_pred_knn, y_prob_knn[:, [1]], 'KNN Euclidean dst')
print(2)
#minkwoski distance
knn = KNeighborsClassifier(n_neighbors=5, metric= 'minkowski', p=3)
knn.fit(X_train, y_train.ravel())
#test
y_pred_knn = knn.predict(X_test)
y_prob_knn = knn.predict_proba(X_test)
metric.append(metrics.precision_score(y_test, y_pred_knn))
metric.append(metrics.recall_score(y_test, y_pred_knn))

evaluate_model(y_test, y_pred_knn, y_prob_knn[:, [1]], 'KNN Minkowski dst')
```

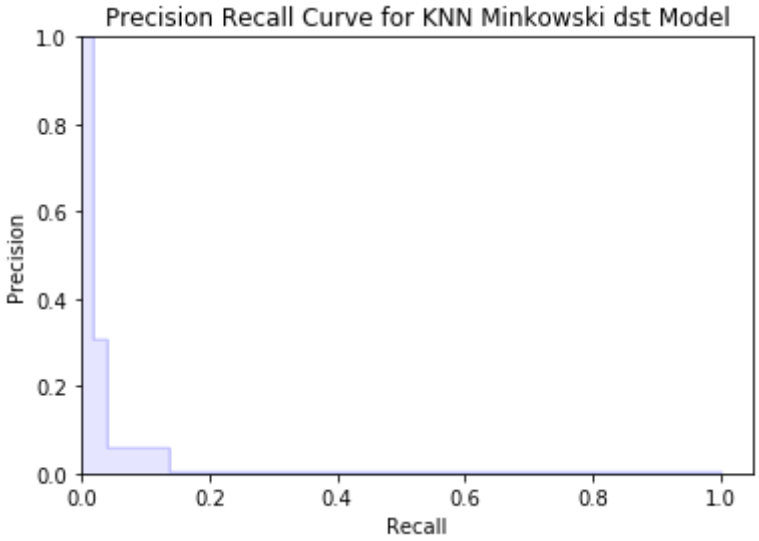
```
Confusion Matrix for KNN Manhattan dst Model
[[56861     0]
 [   90    11]]
Classification Report for KNN Manhattan dst Model
              precision    recall  f1-score   support

       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 Manhattan dst Model
0.6379445202570591
```



```
1
Confusion Matrix for KNN Euclidean dst Model
[[56861     0]
 [   96     5]]
Classification Report for KNN Euclidean dst 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 Euclidean dst Model
0.5777933195088735
```



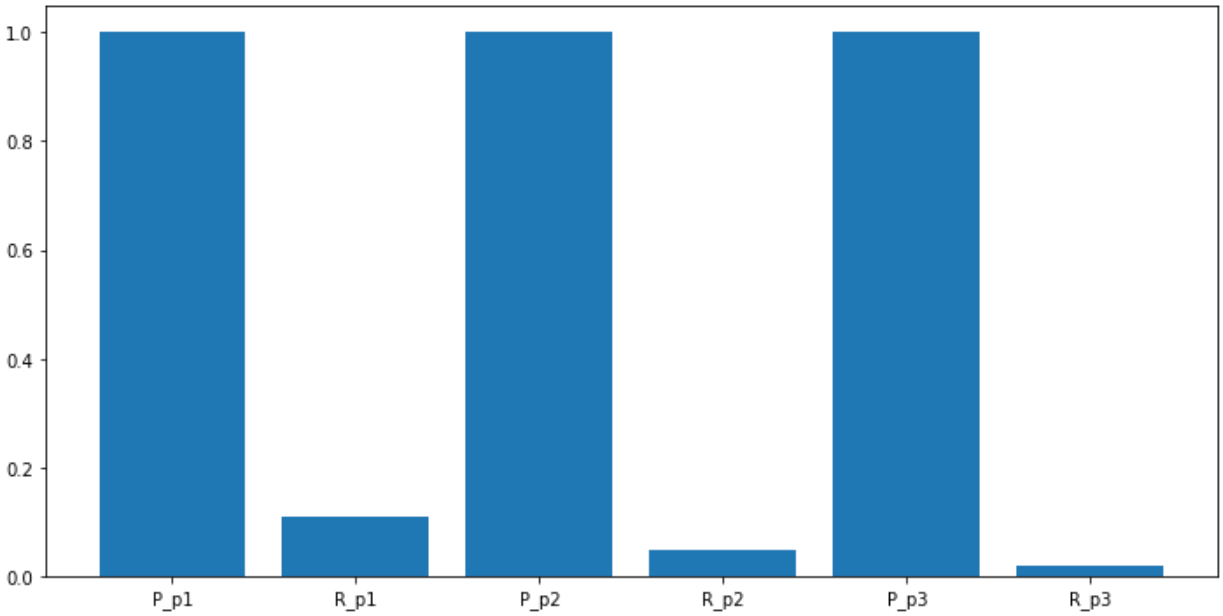

```
2
Confusion Matrix for KNN Minkowski dst Model
[[56861     0]
 [   99     2]]
Classification Report for KNN Minkowski dst Model
              precision    recall  f1-score   support

       False   0.998262  1.000000  0.999130     56861
        True   1.000000  0.019802  0.038835       101

    accuracy                       0.998262     56962
   macro avg   0.999131  0.509901  0.518983     56962
weighted avg   0.998265  0.998262  0.997428     56962

Area under under ROC curve for KNN Minkowski dst Model
0.5673298147070823
```



From the above plotted data, its hard to say which distance measure we should select, but for now it seems, the best recall is given by the model using Euclidean Distance. What to consider totally depends on the tradeoff we are making for, just to be on the safer side, I will select the 1st model as the F1 score is also highest among all others, and change th evalue of n in further experiments

Below is the bar plot showing Precision and Recall for each of the distance metrics uses

In [46]:
```
fig, ax = plt.subplots(figsize=(12,6))
ticks = ['P_p1', 'R_p1', 'P_p2', 'R_p2','P_p3', 'R_p3']
ax.bar(ticks,metric)
plt.show()
print(metric)
```



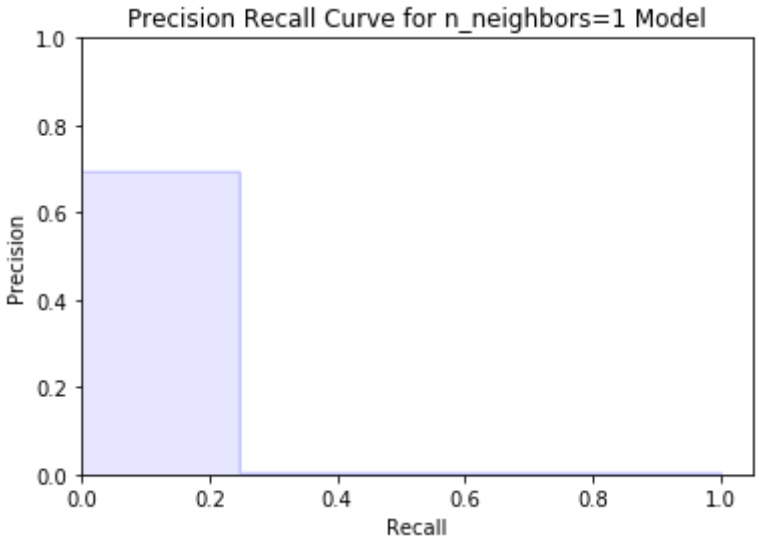

```
[1.0, 0.10891089108910891, 1.0, 0.04950495049504951, 1.0, 0.019801980198019802]
```

From the above bar graph, we can see that precision remains same for all distance metrics and recall changes slightsly, moreover they perform almost the same

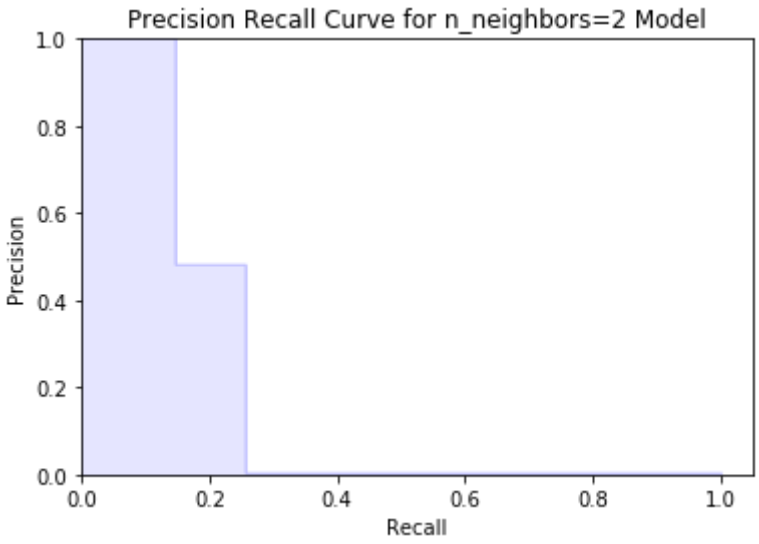
Modifying the no of neighbors

In [ ]:
```
accuracy = []
accuracy_train = []
ns = list(range(1,11))
for i in ns:
    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)
    accuracy_train.append(knn.score(X_train, y_train))
    accuracy.append(knn.score(X_test, y_test))

    evaluate_model(y_test, y_pred_knn, y_prob_knn[:, [1]], 'n_neighbors='+str(i
    print('-'*30)
```

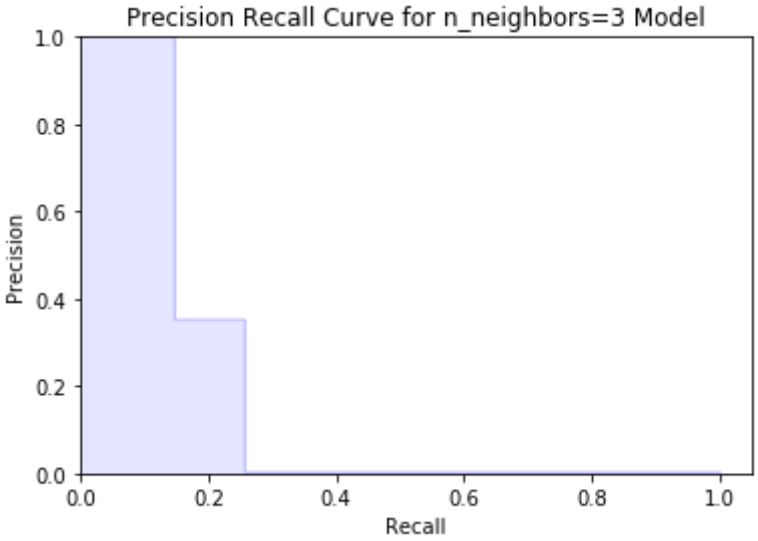
```
Confusion Matrix for n_neighbors=1 Model
[[56850    11]
 [   76    25]]
Classification Report for n_neighbors=1 Model
              precision    recall  f1-score   support

       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 n_neighbors=1 Model
0.6236656491311712
```



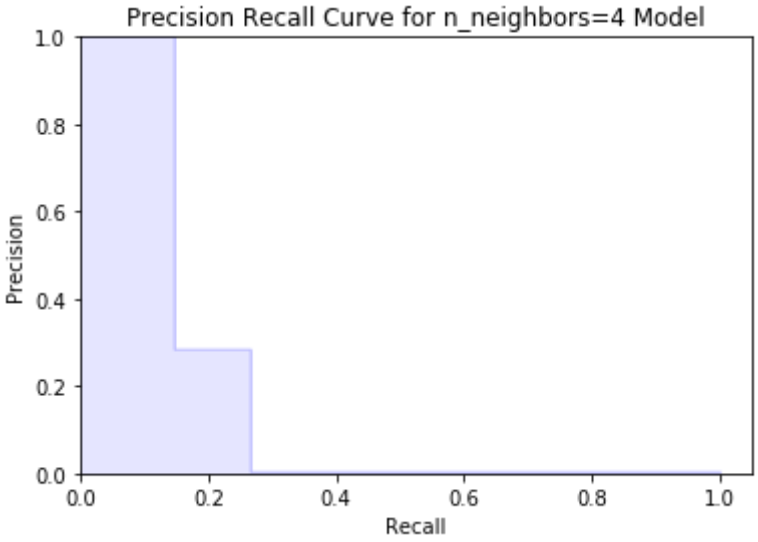
```
------------------------------
Confusion Matrix for n_neighbors=2 Model
[[56861     0]
 [   86    15]]
Classification Report for n_neighbors=2 Model
              precision    recall  f1-score   support

       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 n_neighbors=2 Model
0.6285032233372297
```



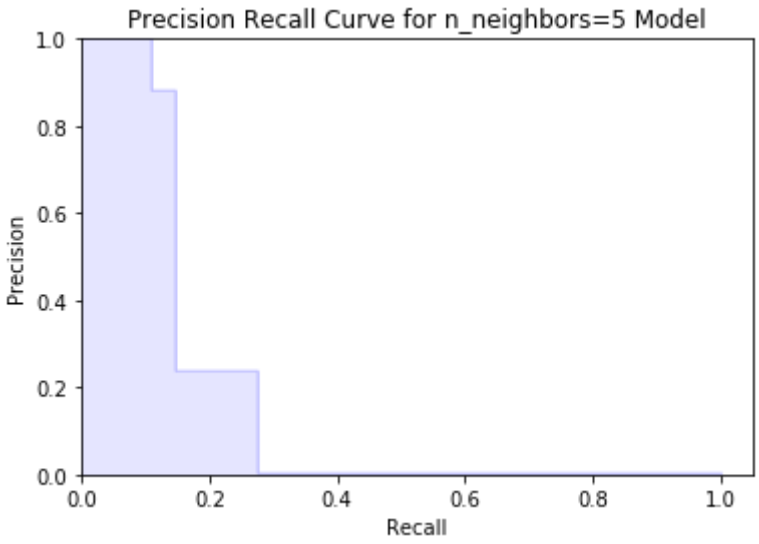

```
------------------------------
Confusion Matrix for n_neighbors=3 Model
[[56861     0]
 [   86    15]]
Classification Report for n_neighbors=3 Model
              precision    recall  f1-score   support

       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 n_neighbors=3 Model
0.6283534748015875
```




```
------------------------------
Confusion Matrix for n_neighbors=4 Model
[[56861     0]
 [   90    11]]
Classification Report for n_neighbors=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 n_neighbors=4 Model
0.6331542213154503
```



```
------------------------------
Confusion Matrix for n_neighbors=5 Model
[[56861     0]
 [   90    11]]
Classification Report for n_neighbors=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 n_neighbors=5 Model
0.6379445202570591
```




```
------------------------------
```

From the above cell, where I changed the value of n, I observed that after some value of n, the precision and recall remains constant and the accuracy doesnt change that much. For instance precision remains constant after n=6 and recall remains the same after n=8. Also, in the specific conditions in a way I have performed experiments, accuracy remains the same(almost) major changes are seen in the precision and recall when n is small and they tend to remain the same when k increases.

Train and Test accuracy for different values of n

In [ ]:
```
fig, ax = plt.subplots(figsize=(12,8))
xs = list(range(1,11))
ax.plot(xs, accuracy, color='navy', label='test_acc', marker='o')
ax.plot(xs, accuracy_train, color='orange', label='train_acc', marker='o')
ax.set_title('Test and Test Accuracies')
ax.set_xlabel('Neighbors')
ax.set_ylabel('Accuracy')
ax.grid()
ax.legend()
plt.show()
```

From the accuracy chart we can see that as the, when n is in smaller range train accuracy decreases while the test accuracy increases until on epoint, and then both accuracies

start on decreasing, but the difference is not much regardless they are nearly the same, so we can say that in this case the accuracy does not change at all.

Now trying to change the underlying data structure, to kd Tree and change distance measure and check the results

In [ ]:
```
accuracy = []
accuracy_train = []
ns = list(range(1,11, 2))
for i in ns:
    knn = KNeighborsClassifier(n_neighbors=i, algorithm='kd_tree', metric= 'mi
    knn.fit(X_train, y_train.ravel())
    #test
    y_pred_knn = knn.predict(X_test)
    y_prob_knn = knn.predict_proba(X_test)
    accuracy_train.append(knn.score(X_train, y_train))
    accuracy.append(knn.score(X_test, y_test))

    evaluate_model(y_test, y_pred_knn, y_prob_knn[:, [1]], 'n_neighbors='+str(i
    print('-'*30)
```

In [ ]:
```
fig, ax = plt.subplots(figsize=(12,8))
xs = list(range(1,11,2))
ax.plot(xs, accuracy, color='navy', label='test_acc', marker='o')
ax.plot(xs, accuracy_train, color='orange', label='train_acc', marker='o')
ax.set_title('Test and Test Accuracies')
ax.set_xlabel('Neighbors')
ax.set_ylabel('Accuracy')
ax.grid()
ax.legend()
plt.show()
```

Changing the algorithm does not change the results that much, but as seen in the provided cell, Logistic Regression provides a better fir to data, with High Precision as well as Higher Recall, which is what is required in such problem.

Q4 - C

Power Iteration program

In [1]:
```
import numpy as np
def power_itr(M, max_iter, E, custom_r0 = None):
    if custom_r0:
        r = custom_r0
    else:
        #get the length of matrix, which is needed to initialize the rank vecto
        n = len(M)

        #initialize the rank vector where every page ahs the same rank i.e 0th
        r = np.ones(n)/n
    print('r{}: {}'.format(0,r))

    #iterate until the rank vector stopps changing(but no more iterations than
    for i in range(max_iter):
        #calculate the new r, which is dot product of matrix M and rank vector
        r_new = np.dot(M,r)
```

```
        #calculate the l1 norm for the new and previous vector(similar to calcu
        l1_norm = np.sum(np.abs(r_new-r))
        #stopping criterion: if l1_norm i.e difference between the absolute val
        if l1_norm<E:
            return ('Stopped at Iteration: {} | with ranks : {}'.format(i+1, r_
        if i<5:
            print('r{}: {}'.format(i+1,r_new))
        r=r_new
    return ('Max Iteration reached: {} | with ranks : {}'.format(i+1, r_new))
```

Trying out with a simple matrix to validate if function is running correctly

In [2]:
```
M = np.array([[1/2,1/2,0],
              [1/2,0,1],
              [0,1/2,0]])
print(M)
power_itr(M, max_iter = 100, E = 1e-9)
```

```
[[0.5 0.5 0. ]
 [0.5 0.  1. ]
 [0.  0.5 0. ]]
r0: [0.33333333 0.33333333 0.33333333]
r1: [0.33333333 0.5        0.16666667]
r2: [0.41666667 0.33333333 0.25      ]
r3: [0.375      0.45833333 0.16666667]
r4: [0.41666667 0.35416667 0.22916667]
r5: [0.38541667 0.4375     0.17708333]
```

Out[2]:
```
'Stopped at Iteration: 95 | with ranks : [0.4 0.4 0.2]'
```

M is the matrix from the graph given in Q4

In [3]:
```
M = np.array([[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]])
print(M)
power_itr(M, max_iter = 100, E = 1e-9)
```

```
[[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.  ]]
r0: [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667]
r1: [0.125      0.08333333 0.20833333 0.20833333 0.16666667 0.20833333]
r2: [0.125      0.0625     0.20833333 0.22916667 0.16666667 0.20833333]
r3: [0.11979167 0.0625     0.203125   0.22395833 0.17708333 0.21354167]
r4: [0.12239583 0.05989583 0.2109375  0.21875    0.171875   0.21614583]
r5: [0.12304687 0.06119792 0.20898438 0.22591146 0.17057292 0.21028646]
```

Out[3]:
```
'Stopped at Iteration: 73 | with ranks : [0.12182741 0.06091371 0.20812183 0.2
2335025 0.17258883 0.21319797]'
```

Passing Initial rank vectors(3 differnt) where the probablity of a random surfer is not the same to go to any of the page

In [9]:
```
M = np.array([[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]])
print(M)
r = [0.1,0.2,0.1,1/5,1/5,1/5]
print(np.sum(r))
power_itr(M, max_iter = 100, E = 1e-9, custom_r0 = r)
```

```
[[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.  ]]
1.0
r0: [0.1, 0.2, 0.1, 0.2, 0.2, 0.2]
r1: [0.15 0.05 0.25 0.15 0.15 0.25]
r2: [0.1375 0.075  0.2125 0.2625 0.15   0.1625]
r3: [0.1     0.06875 0.175   0.23125 0.2     0.225  ]
r4: [0.1296875 0.05      0.2296875 0.1921875 0.165625  0.2328125]
r5: [0.12890625 0.06484375 0.21171875 0.2421875  0.1609375  0.19140625]
```

Out[9]:
```
'Stopped at Iteration: 80 | with ranks : [0.12182741 0.06091371 0.20812183 0.2
2335025 0.17258883 0.21319797]'
```

In [10]:
```
M = np.array([[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]])
print(M)
r = [0,0.2,0.2,0,2/5,1/5]
print(np.sum(r))
power_itr(M, max_iter = 100, E = 1e-9, custom_r0 = r)
```

```
[[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.  ]]
1.0
r0: [0, 0.2, 0.2, 0, 0.4, 0.2]
r1: [0.15 0.   0.35 0.25 0.   0.25]
r2: [0.125 0.075 0.125 0.35  0.2   0.125]
r3: [0.08125 0.0625  0.18125 0.14375 0.2375  0.29375]
r4: [0.1625   0.040625 0.28125  0.196875 0.1125   0.20625 ]
r5: [0.11328125 0.08125    0.16953125 0.29140625 0.1796875  0.16484375]
```

Out[10]:
```
'Stopped at Iteration: 84 | with ranks : [0.12182741 0.06091371 0.20812183 0.2
2335025 0.17258883 0.21319797]'
```

In [11]:
```
M = np.array([[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]])
print(M)
r = [0,0,0,0,0,1]
print(np.sum(r))
power_itr(M, max_iter = 100, E = 1e-9, custom_r0 = r)
```

```
[[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.  ]]
1
r0: [0, 0, 0, 0, 0, 1]
r1: [0.5 0.  0.5 0.  0.  0. ]
r2: [0.   0.25 0.   0.5  0.25 0.  ]
r3: [0.0625 0.     0.1875 0.0625 0.25   0.4375]
r4: [0.21875 0.03125 0.34375 0.1875  0.0625  0.15625]
r5: [0.0859375 0.109375  0.1171875 0.3515625 0.203125  0.1328125]
```

Out[11]:
```
'Stopped at Iteration: 86 | with ranks : [0.12182741 0.06091371 0.20812183 0.2
2335025 0.17258883 0.21319797]'
```

As we can see, the final rank is the same, just the number of iterations to reach to it has slightly increased. Even though some of the probablities are 0 we reach to the same rank values everytime

In [16]:
```
45%6
```

Out[16]:
```
3
```

In [ ]:
```

```

4) (a) Transition probability Matrix.

	A	B	C	D	E	F	outlinks.
A	0	1/4	0	0	0	1/2	2
B	1/2	0	0	0	0	0	4
C	0	1/4	0	0	1/2	1/2	1
D	0	1/4	1	0	0	0	2
E	1/2	0	0	1/2	0	0	2
F	0	1/4	0	1/2	1/2	0	2
							13

(b) Surfer starts at A.

$E_1 = P(A \rightarrow B) = 1/2$

$E_2 = P(B \rightarrow D) = 1/4$

} (from transition matrix).

considering the event that surfer goes from A to B and B to D independent of each other.

$$P(E_1 \cap E_2) = P(E_1) * P(E_2)$$
$$= (1/2) * (1/4)$$
$$= \frac{1}{8}$$

~~(b) $h_1(x) = 2x + 1$~~

(c) Code is above along with Q3 (same notebook).

5) Precision nutrition is where a user will get personalized nutrition & diet recommendations based on how his/her body reacts to certain food items, since everyone reacts differently. This will help reduce the cause of disease & can also serve as a preventive measure. Ultimately increasing lifespan & reducing costs. This is different than traditional approach where individual dietry recommendations were given based on ~~average~~ from research or findings from a sample of population which does not necessarily fit all.

- Author mentions that research in genomics have shown that individual genes are related to how one's body reacts to certain foods. & everyone has diffrent nutrition requirements. Also other factors like gut microbia, meal timing, physical activity, environmental & social factors, etc must be considered while giving dietry recommendation. Author also mentions that ~~PN~~ more research on PN & related studies can help early detection of disease i.e prevention over post medication. Furthermore the author explains what are limiting factors like, mainly the cost of research & testing as well as smart devices that will monitor responses from the body regularly. Also, there is much more reseach to be done around other areas similar to PN.
- Author also states that the impact of poor diet is not considered much & instead undernutrition or hunger is assumed as causes. to diet related conditions. Even though lifespan has increased over many years author believes that average years a person lives without disease is not reduced. and poor diet is responsible for chronic conditions, & health complications. Author states that only some metrics like. (metabolic) blood pressure, glucose, hemoglobin, cholestrol, lipoprotein, etc provide limited information. But instead using deep phenotyping, more personal factors can be considered & can provide individual personalized recommendations. Further economic, socological & geographic factors can also be used since these provide more complexity to improve recommendatio

- Deep Learning (Deep phenotyping) must be used to integrate multiple diverging attributes which will result in better responses to diet recommend. Author suggests the use of wearable devices & hand held devices to continously monitor ones responses to a food consumed., so that better recommendations can be given. Data mining tasks here would be firstly identify which measurements are important or will be needed, then the raw sensor data must be transformed in a way, it can be used for further modelling. Big data management & dimensionality reduction algorithms must be used to speed up the process. Also integration of real-time data collection & modelling i.e continously improving the results or recommendations is what mentioned by the author.
- Also using apps, users can themselves measure their parameters & get recommendations quickly. Other data mining tasks that I think can be., to transform one or more features into one, so that the value of information ~~through~~ increase & dimensionality reduces. Various factors are mentioned by author which include, biological (microbiome), genome, epigenome), personal (lifestyle, sleep activity), social (~~so~~ norms), heridetry (family ~~of~~ history), & many more other diverging data types needs to be processed in such a way that they collectively are responsible for recommendations hence encoding, standard scaling & data cleaning, etc must be used. Also data warehousing or collection steps must be identified & from that analysis of data will help better solve the problem. AI & computer vision can be used (scan barcodes of packaged foods). to keep track of ones diet effectively & easily.
- I think the main task here (Data mining) will be to integrate data from different sources & then cleaning it & pre-processing it. There is no clear dataset mentioned in this article but the various attributs that are related & can be used are mentioned & basically an idea is proposed with some statistical findings which suggests that Personalized Nutrition can be more beneficial than general recommendations.