In [268…

```
import pandas as pd
import numpy as np
import sklearn.neighbors
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.neighbors import NearestCentroid
import math
scaler = StandardScaler()

def proficCalculator(data, fund):
    # Week 0 case
    week1Data = data[0]
    week1Label = week1Data.Label[0] # week 0 label

    if week1Label == 1:
        stock = True
        buyPrice = week1Data.Close[0] # week 0 first day price
        sellPrice = week1Data.Close[len(week1Data)-1] # week 0 last day price
    else:
        stock = False
        buyPrice = week1Data.Close[len(week1Data)-1] # week 0 last day price
        sellPrice = week1Data.Close[len(week1Data)-1] # week 0 last day price


    for df in data[1:]:
        nextWeekColor = df.Label[0]
        nextClosePrice = df.Close[len(df)-1]

        # stock + green = no action
        if (stock == True) and (nextWeekColor == 1):
            stock == True # Keep holding the stock
            buyPrice = buyPrice # Buy point stay
            sellPrice = nextClosePrice # Sell point move forward

        # stock + red = sell
        elif (stock == True) and (nextWeekColor == 0):
            r = 1 + (sellPrice - buyPrice) / sellPrice
            fund = fund * r
            buyPrice = nextClosePrice
            sellPrice = nextClosePrice
            stock = False

        # money + green = buy stock
        elif (stock == False) and (nextWeekColor == 1):
            buyPrice = buyPrice
            sellPrice = nextClosePrice
            stock = True
        # money + red = no action
        elif (stock == False) and (nextWeekColor == 0):
            buyPrice = nextClosePrice
            sellPrice = nextClosePrice
            stock = False

    # Last withdraw
    r = 1 + (sellPrice - buyPrice) / sellPrice
    fund = fund * r
    return fund

def labelMapping(year, week, label):
    labelMap = {}
    for (y, w, l) in zip(year, week, label):
        key = (y, w)
        value = l
        labelMap[key] = value
    return labelMap

def cutWeek(weekNumber, data):
    weekdata = []
    for i in range(weekNumber):
         temp = data[data.Week_Number == i]
         temp = temp.reset_index(drop=True)
         weekdata.append(temp)
    return weekdata

# minkowski setting
def minkowski_p(a,b,p):
    return np.linalg.norm(a-b, ord=p)
p = 1.5
knn_Minkowski_p = KNeighborsClassifier(n_neighbors=3,
                                        metric = lambda a,b: minkowski_p(a,b,p))
```

In [269…

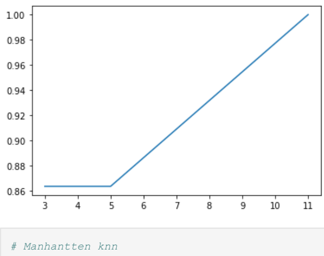
```
df = pd.read_csv('./GOOGL_weekly_return_volatility.csv')
year1 = df[df.Year == 2019]
year2 = df[df.Year == 2020]
```

Quesiton1 Manhatten

In [270…

```
# Regular KNN
kList = [3,5,7,9,11]
accuracy = []

x = year1[['mean_return', 'volatility']]
scaler.fit(x)
x = scaler.transform(x)

y = year1.label
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.4, random_state=0)

for k in kList:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(xTrain, yTrain)
    yPredict = knn.predict(xTest)
    accuracy.append(accuracy_score(yTest, yPredict))

plt.plot(kList, accuracy)
print(accuracy)

## Optimal k is 11

x = year1[['mean_return', 'volatility']]
scaler.fit(x)
x = scaler.transform(x)
y = year1.label
xTest = year2[['mean_return', 'volatility']]
scaler.fit(xTest)
xTest = scaler.transform(xTest)
yTest = year2.label

knn = KNeighborsClassifier(n_neighbors=11)
knn.fit(x, y)
yPredict = knn.predict(xTest)

accuracy = accuracy_score(yTest, yPredict)
print(accuracy)

## Confusion Matrix I choose
temp = confusion_matrix(yTest, yPredict)
print(temp)

tn = temp[0][0]
fn = temp[1][0]
tp = temp[1][1]
fp = temp[0][1]

tpr = tp / (tp + fn)
tnr = tn / (tn + fp)

print('TPR = {}, TNR = {}, k = 11'.format(tpr, tnr))



## trade with regular KNN
dfDetail = pd.read_csv('./GOOGL_weekly_return_volatility_detailed.csv')
year2Detail = dfDetail[dfDetail.Year == 2020]
year2Detail = year2Detail.reset_index(drop = True)
## Add label to detail
lMap = labelMapping(year2.Year, year2.Week_Number, yPredict)
temp = []
for (y, w) in zip(year2Detail.Year, year2Detail.Week_Number):
    key = (y, w)
    temp.append(lMap[key])
year2Detail['Label'] = temp
year2Detail = year2Detail[['Year', 'Week_Number', 'Close', 'Label']]
## Cut goo2020
goo2020Week = cutWeek(53, year2Detail)
## trading base on Manhattan KNN label
total = proficCalculator(goo2020Week, 100)
print("Using Label regular KNN: {}".format(total))
```

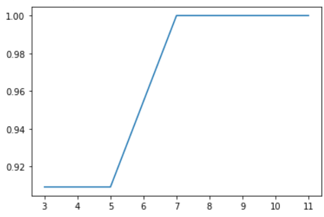
```
[0.8636363636363636, 0.8636363636363636, 0.9090909090909091, 0.9545454545454546, 1.0]
0.7924528301886793
[[21  4]
 [ 7 21]]
TPR = 0.75, TNR = 0.84, k = 11
Using Label regular KNN: 229.69001446054642
```



In [271…

```
# Manhantten knn
kList = [3,5,7,9, 11]
accuracy = []

x = year1[['mean_return', 'volatility']]
scaler.fit(x)
x = scaler.transform(x)

y = year1.label
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.4, random_state=0)

for k in kList:
    knn = KNeighborsClassifier(n_neighbors=k, p = 1)
    knn.fit(xTrain, yTrain)
    yPredict = knn.predict(xTest)
    accuracy.append(accuracy_score(yTest, yPredict))

plt.plot(kList, accuracy)
print(accuracy)

## Optimal k is 7

x = year1[['mean_return', 'volatility']]
scaler.fit(x)
x = scaler.transform(x)
y = year1.label
xTest = year2[['mean_return', 'volatility']]
scaler.fit(xTest)
xTest = scaler.transform(xTest)
yTest = year2.label

knn = KNeighborsClassifier(n_neighbors=7, p = 1)
knn.fit(x, y)
yPredict = knn.predict(xTest)

accuracy = accuracy_score(yTest, yPredict)
print(accuracy)

## Confusion Matrix I choose
temp = confusion_matrix(yTest, yPredict)
print(temp)

tn = temp[0][0]
fn = temp[1][0]
tp = temp[1][1]
fp = temp[0][1]

tpr = tp / (tp + fn)
tnr = tn / (tn + fp)

print('TPR = {}, TNR = {}, k = 7'.format(tpr, tnr))
```

```
[0.9090909090909091, 0.9090909090909091, 1.0, 1.0, 1.0]
0.8113207547169812
[[22  3]
 [ 7 21]]
TPR = 0.75, TNR = 0.88, k = 7
```



In [272…

```python
# Strategy check
dfDetail = pd.read_csv('./GOOGL_weekly_return_volatility_detailed.csv')
year2Detail = dfDetail[dfDetail.Year == 2020]
year2Detail = year2Detail.reset_index(drop = True)

## Add label to detail
lMap = labelMapping(year2.Year, year2.Week_Number, yPredict)
temp = []
for (y, w) in zip(year2Detail.Year, year2Detail.Week_Number):
    key = (y, w)
    temp.append(lMap[key])
year2Detail['Label'] = temp
year2Detail = year2Detail[['Year', 'Week_Number', 'Close', 'Label']]

## Cut goo2020
goo2020Week = cutWeek(53, year2Detail)




## trading base on Manhattan KNN label
total = proficCalculator(goo2020Week, 100)
print("Using Label Manhattan KNN: {}".format(total))

## trding BH
firstWeek = goo2020Week[0]
firstClose = firstWeek.Close[0]

lastWeek = goo2020Week[-1]
lastClose = lastWeek.Close[len(lastWeek)-1]

r = 1 + (lastClose - firstClose) / lastClose
total = 100 * r
print("Buy on first day and Sell on last day: {}".format(total))
```

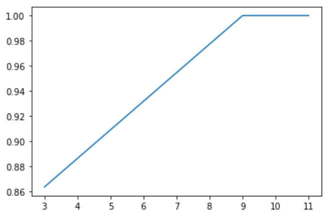
```
Using Label Manhattan KNN: 233.65539184081408
Buy on first day and Sell on last day: 121.17033527942765
```

Question2 Minkowski p = 1.5

In [273…

```python
# Year1 accuracy
kList = [3,5,7,9,11]
accuracy = []




x = year1[['mean_return', 'volatility']]
scaler.fit(x)
x = scaler.transform(x)

y = year1.label
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.4, random_state=0)

for k in kList:
    p = 1.5
    knn_Minkowski_p = KNeighborsClassifier(n_neighbors=k, metric = lambda a,b: minkowski_p(a,b,p))

    knn_Minkowski_p.fit(xTrain, yTrain)
    yPredict = knn_Minkowski_p.predict(xTest)
    accuracy.append(accuracy_score(yTest, yPredict))

plt.plot(kList, accuracy)
print(accuracy)
```

```
[0.8636363636363636, 0.9090909090909091, 0.9545454545454546, 1.0, 1.0]
```



In [274…

```python
# Year 2 prediction k = 9
x = year1[['mean_return', 'volatility']]
scaler.fit(x)
x = scaler.transform(x)
y = year1.label
xTest = year2[['mean_return', 'volatility']]
scaler.fit(xTest)
xTest = scaler.transform(xTest)
yTest = year2.label

knn_Minkowski_p = KNeighborsClassifier(n_neighbors=9, metric = lambda a,b: minkowski_p(a,b,p))
knn_Minkowski_p.fit(x, y)
yPredict = knn_Minkowski_p.predict(xTest)
print(accuracy_score(yTest, yPredict))
## Confusion Matrix I choose
temp = confusion_matrix(yTest, yPredict)
print(temp)

tn = temp[0][0]
fn = temp[1][0]
tp = temp[1][1]
fp = temp[0][1]

tpr = tp / (tp + fn)
tnr = tn / (tn + fp)

print('TPR = {}, TNR = {}, k = 9'.format(tpr, tnr))
```

```
0.7924528301886793
[[21  4]
 [ 7 21]]
TPR = 0.75, TNR = 0.84, k = 9
```

In [275…

```python
# Strategy check
dfDetail = pd.read_csv('./GOOGL_weekly_return_volatility_detailed.csv')
year2Detail = dfDetail[dfDetail.Year == 2020]
year2Detail = year2Detail.reset_index(drop = True)

## Add label to detail
lMap = labelMapping(year2.Year, year2.Week_Number, yPredict)
temp = []
for (y, w) in zip(year2Detail.Year, year2Detail.Week_Number):
    key = (y, w)
    temp.append(lMap[key])
year2Detail['Label'] = temp
year2Detail = year2Detail[['Year', 'Week_Number', 'Close', 'Label']]

## Cut goo2020
goo2020Week = cutWeek(53, year2Detail)


## trading base on Manhattan KNN label
total = proficCalculator(goo2020Week, 100)
print("Using Label p = 1.5 KNN: {}".format(total))

## trding BH
firstWeek = goo2020Week[0]
firstClose = firstWeek.Close[0]

lastWeek = goo2020Week[-1]
lastClose = lastWeek.Close[len(lastWeek)-1]

r = 1 + (lastClose - firstClose) / lastClose
total = 100 * r
print("Buy on first day and Sell on last day: {}".format(total))
```

```
Using Label Manhattan KNN: 235.35506721992058
Buy on first day and Sell on last day: 121.17033527942765
```

Question3 Nearest Centroid

In [276…

```python
year1Green = year1[year1.label == 1]
greenCentroid = (year1Green.mean_return.mean(), year1Green.volatility.mean())
print("Green Centroid")
print(greenCentroid)

year1Red = year1[year1.label == 0]
redCentroid = (year1Red.mean_return.mean(), year1Red.volatility.mean())
print("Red Centroid")
print(redCentroid)

# average and mediam distance
## green
greenDistance = []
for (m, s) in zip(year1Green.mean_return, year1Green.volatility):
    t = (m, s)
    greenDistance.append(math.dist(t, greenCentroid))
print('average and mediam')
print(np.mean(greenDistance), np.median(greenDistance))

## red
redDistance = []
for (m, s) in zip(year1Red.mean_return, year1Red.volatility):
    t = (m, s)
    redDistance.append(math.dist(t, redCentroid))
print('average and mediam')
print(np.mean(redDistance), np.median(redDistance))
```

```
Green Centroid
(0.49741049381481467, 1.177831770814815)
Red Centroid
(-0.3026615384615384, 1.379016029923077)
average and mediam
0.7289493662019179 0.5145998971085394
average and mediam
0.7773815584328501 0.6090608424078319
```

In [277…

```python
# KNN Centroid
x = year1[['mean_return', 'volatility']]
scaler.fit(x)
x = scaler.transform(x)
y = year1.label

xTest = year2[['mean_return', 'volatility']]
scaler.fit(xTest)
xTest = scaler.transform(xTest)
yTest = year2.label

clf = NearestCentroid()
clf.fit(x, y)
yPredict = clf.predict(xTest)
print(accuracy_score(yTest, yPredict))

## Confusion Matrix I choose
temp = confusion_matrix(yTest, yPredict)
print(temp)

tn = temp[0][0]
fn = temp[1][0]
tp = temp[1][1]
fp = temp[0][1]

tpr = tp / (tp + fn)
tnr = tn / (tn + fp)
```

```
print('TPR = {}, TNR = {}'.format(tpr, tnr))
```

```
0.7924528301886793
[[19  6]
 [ 5 23]]
TPR = 0.8214285714285714, TNR = 0.76, k = 11
```

In [278…

```
# Strategy check
dfDetail = pd.read_csv('./GOOGL_weekly_return_volatility_detailed.csv')
year2Detail = dfDetail[dfDetail.Year == 2020]
year2Detail = year2Detail.reset_index(drop = True)

## Add label to detail
lMap = labelMapping(year2.Year, year2.Week_Number, yPredict)
temp = []
for (y, w) in zip(year2Detail.Year, year2Detail.Week_Number):
    key = (y, w)
    temp.append(lMap[key])
year2Detail['Label'] = temp
year2Detail = year2Detail[['Year', 'Week_Number', 'Close', 'Label']]

## Cut goo2020
goo2020Week = cutWeek(53, year2Detail)


## trading base on Manhattan KNN label
total = proficCalculator(goo2020Week, 100)
print("Using Label Manhattan KNN: {}".format(total))

## trding BH
firstWeek = goo2020Week[0]
firstClose = firstWeek.Close[0]

lastWeek = goo2020Week[-1]
lastClose = lastWeek.Close[len(lastWeek)-1]

r = 1 + (lastClose - firstClose) / lastClose
total = 100 * r
print("Buy on first day and Sell on last day: {}".format(total))
```

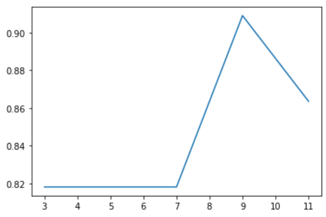
```
Using Label Manhattan KNN: 239.64550014107087
Buy on first day and Sell on last day: 121.17033527942765
```

Question4 Domain Transformation

In [279…

```
# data setting
yearT1 = year1
yearT1 = yearT1.assign(xx = yearT1.mean_return**2)
yearT1 = yearT1.assign(xy = yearT1.mean_return * yearT1.volatility * math.sqrt(2))
yearT1 = yearT1.assign(yy = yearT1.volatility**2)

yearT2 = year2
yearT2 = yearT2.assign(xx = yearT2.mean_return**2)
yearT2 = yearT2.assign(xy = yearT2.mean_return * yearT2.volatility * math.sqrt(2))
yearT2 = yearT2.assign(yy = yearT2.volatility**2)



# Regular KNN
kList = [3,5,7,9,11]
accuracy = []

x = yearT1[['xx', 'xy', 'yy']]
scaler.fit(x)
x = scaler.transform(x)

y = yearT1.label
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.4, random_state=0)

for k in kList:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(xTrain, yTrain)
    yPredict = knn.predict(xTest)
    accuracy.append(accuracy_score(yTest, yPredict))

plt.plot(kList, accuracy)
print(accuracy)


## Optimal k is 9

x = yearT1[['xx', 'xy', 'yy']]
scaler.fit(x)
x = scaler.transform(x)
y = yearT1.label
xTest = yearT2[['xx', 'xy', 'yy']]
scaler.fit(xTest)
xTest = scaler.transform(xTest)
yTest = year2.label

knn = KNeighborsClassifier(n_neighbors=9)
knn.fit(x, y)
yPredict = knn.predict(xTest)

accuracy = accuracy_score(yTest, yPredict)
print(accuracy)


## Confusion Matrix I choose
temp = confusion_matrix(yTest, yPredict)
print(temp)

tn = temp[0][0]
fn = temp[1][0]
tp = temp[1][1]
fp = temp[0][1]

tpr = tp / (tp + fn)
tnr = tn / (tn + fp)

print('TPR = {}, TNR = {}, k = 9'.format(tpr, tnr))
```

```
[0.8181818181818182, 0.8181818181818182, 0.8181818181818182, 0.9090909090909091, 0.8636363636363636]
0.8113207547169812
[[20  5]
 [ 5 23]]
TPR = 0.8214285714285714, TNR = 0.8, k = 9
```



In [280…

```
# Strategy check
dfDetail = pd.read_csv('./GOOGL_weekly_return_volatility_detailed.csv')
year2Detail = dfDetail[dfDetail.Year == 2020]
year2Detail = year2Detail.reset_index(drop = True)

## Add label to detail
lMap = labelMapping(year2.Year, year2.Week_Number, yPredict)
temp = []
for (y, w) in zip(year2Detail.Year, year2Detail.Week_Number):
    key = (y, w)
    temp.append(lMap[key])
year2Detail['Label'] = temp
year2Detail = year2Detail[['Year', 'Week_Number', 'Close', 'Label']]

## Cut goo2020
goo2020Week = cutWeek(53, year2Detail)



## trading base on Manhattan KNN label
total = proficCalculator(goo2020Week, 100)
print("Using Label Manhattan KNN: {}".format(total))

## trding BH
firstWeek = goo2020Week[0]
firstClose = firstWeek.Close[0]

lastWeek = goo2020Week[-1]
lastClose = lastWeek.Close[len(lastWeek)-1]

r = 1 + (lastClose - firstClose) / lastClose
total = 100 * r
print("Buy on first day and Sell on last day: {}".format(total))
```

```
Using Label Manhattan KNN: 238.55325493337438
Buy on first day and Sell on last day: 121.17033527942765
```

Question5 k-predicted Neighbors

In [281…

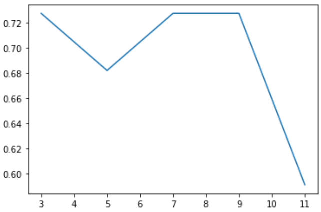
```
def distanceList(testPoint, xTrainSet):
    temp = []
    for (m, s) in zip(xTrainSet.mean_return, xTrainSet.volatility):
        temp.append((m, s))

    dis = []
    for t in temp:
        d = math.dist(testPoint, t)
        dis.append((t[0], t[1], d))

    dis = sorted(dis, reverse=False, key = lambda x:x[2])
    return dis

def kNeighborPoint(testPoint, xTrainSet, k):
    d = distanceList(testPoint, xTrainSet)
    temp = []
    for i in range(k):
        node = d[i]
        temp.append((node[0], node[1]))
    return temp


def kNeighborPridct(testPoint, xTrainSet, yTrainSet, k):

    neighborPoiont = kNeighborPoint(testPoint, xTrainSet, k)

    knn = KNeighborsClassifier(n_neighbors=k)
    scaler.fit(xTrainSet)
    xTrainSet = scaler.transform(xTrainSet)

    knn.fit(xTrainSet, yTrainSet)
    yPredict = knn.predict(neighborPoiont)

    return yPredict

def kpn(xTrain, yTrain, xTest, k):
    yPredict = []
    for (f1, f2) in zip(xTest.mean_return, xTest.volatility):
        kNPoint = kNeighborPridct((f1, f2), xTrain, yTrain, k)
        if sum(kNPoint) >= k/2:
            yPredict.append(1)
        else:
            yPredict.append(0)
    return yPredict
```

In [282…

```
# kpn for year 1
kList = [3,5,7,9,11]
accuracy = []

x = year1[['mean_return', 'volatility']]
y = year1.label
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.4, random_state=0)

for k in kList:

    yPredict = kpn(xTrain, yTrain, xTest, k)
    accuracy.append(accuracy_score(yTest, yPredict))

plt.plot(kList, accuracy)
print(accuracy)


# choose optimal k is 3
```

```
x = year1[['mean_return', 'volatility']]
y = year1.label
xTest = year2[['mean_return', 'volatility']]
yTest = year2.label

yPredict = kpn(x, y, xTest, 3)

accuracy = accuracy_score(yTest, yPredict)
print(accuracy)


## Confusion Matrix I choose
temp = confusion_matrix(yTest, yPredict)
print(temp)

tn = temp[0][0]
fn = temp[1][0]
tp = temp[1][1]
fp = temp[0][1]

tpr = tp / (tp + fn)
tnr = tn / (tn + fp)

print('TPR = {}, TNR = {}, k = 3'.format(tpr, tnr))
```

```
[0.7272727272727273, 0.6818181818181818, 0.7272727272727273, 0.7272727272727273, 0.5909090909090909]
0.7735849056603774
[[22  3]
 [ 9 19]]
TPR = 0.6785714285714286, TNR = 0.88, k = 3
```



In [283…

```
# Strategy check
dfDetail = pd.read_csv('./GOOGL_weekly_return_volatility_detailed.csv')
year2Detail = dfDetail[dfDetail.Year == 2020]
year2Detail = year2Detail.reset_index(drop = True)

## Add label to detail
lMap = labelMapping(year2.Year, year2.Week_Number, yPredict)
temp = []
for (y, w) in zip(year2Detail.Year, year2Detail.Week_Number):
    key = (y, w)
    temp.append(lMap[key])
year2Detail['Label'] = temp
year2Detail = year2Detail[['Year', 'Week_Number', 'Close', 'Label']]

## Cut goo2020
goo2020Week = cutWeek(53, year2Detail)




## trading base on Manhattan KNN label
total = proficCalculator(goo2020Week, 100)
print("Using Label Manhattan KNN: {}".format(total))

## trding BH
firstWeek = goo2020Week[0]
firstClose = firstWeek.Close[0]

lastWeek = goo2020Week[-1]
lastClose = lastWeek.Close[len(lastWeek)-1]

r = 1 + (lastClose - firstClose) / lastClose
total = 100 * r
print("Buy on first day and Sell on last day: {}".format(total))
```

```
Using Label Manhattan KNN: 200.15101296856818
Buy on first day and Sell on last day: 121.17033527942765
```

Quesiton6 k-hyperplanes

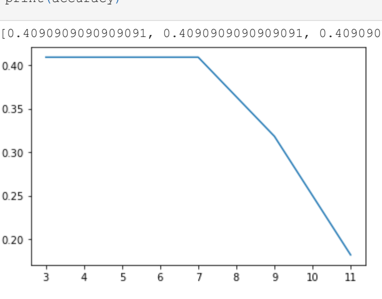
In [316…

```
def distanceNode(testNode, xTrainNode):
    dis = []
    for x in xTrainNode:
        d = math.dist(testNode.coordinate, x.coordinate)
        dis.append((x, d))
    dis = sorted(dis, reverse=False, key = lambda x:x[1])
    return dis


def kNeighborNode(testNode, xTrainNode, k):
    d = distanceNode(testNode, xTrainNode)
    knei = []
    for i in range(k):
        knei.append(d[i][0])
    return knei

def nodePlane(testNode, neighborNode, xTrainNode):
    tnCor = testNode.coordinate
    nnCor = neighborNode.coordinate
    tnCor1, tnCor2 = tnCor[0], tnCor[1]
    nnCor1, nnCor2 = nnCor[0], nnCor[1]
    keyVector = np.array([tnCor1 - nnCor1, tnCor2 - nnCor2])
    negativeNode = []

    for xNode in xTrainNode:
        xnCor = xNode.coordinate
        xnCor1 = xnCor[0]
        xnCor2 = xnCor[1]
        sideVector = np.array([xnCor1 - nnCor1, xnCor2 - nnCor2])
        dot = np.dot(keyVector, sideVector)
        if dot < 0:
            negativeNode.append(xNode.label)

    if sum(negativeNode) >= len(negativeNode) / 2:
         return 1
    else:
        return 0

def kHyperPlane(testNodeSet, trainNodeSet, k):
    yPredict = []
    for tn in testNodeSet:
        kNeighbor = kNeighborNode(tn, trainNodeSet, k)
        neiLabel = []
        for kn in kNeighbor:
            neiLabel.append(nodePlane(tn, kn, trainNodeSet))

        if sum(neiLabel) >= len(neiLabel)/2:
            yPredict.append(1)
        else:
            yPredict.append(0)

    return yPredict

class node:
    def __init__(self, coordinate, label):
        self.coordinate = coordinate
        self.label = label
```

In [317…

```python
# kHyperplane for year1
kList = [3,5,7,9,11]
accuracy = []

x = year1[['mean_return', 'volatility']]
y = year1.label
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size=0.4, random_state=0)

trainNodeSet = []
testNodeSet = []
for (f1, f2, l) in zip(xTrain.mean_return, xTrain.volatility, yTrain ):
    trainNodeSet.append( node( (f1, f2 ) , l ) )
for (f1, f2, l) in zip( xTest.mean_return , xTest.volatility , yTest ):
    testNodeSet.append( node( ( f1 , f2 ) , l ) )



for k in kList:
    yPredict = kHyperPlane(testNodeSet, trainNodeSet, k)
    accuracy.append(accuracy_score(yTest, yPredict))
plt.plot(kList, accuracy)
print(accuracy)
```

```
[0.4090909090909091, 0.4090909090909091, 0.4090909090909091, 0.3181818181818182, 0.18181818181818182]
```



In [330…

```python
# kHyperplane for year2 k = 3
x = year1[['mean_return', 'volatility']]
y = year1.label
trainNodeSet = []
for (f1, f2, l) in zip(x.mean_return, x.volatility, y ):
    trainNodeSet.append( node( (f1, f2 ) , l ) )


x = year2[['mean_return', 'volatility']]
ytest = year2.label
testNodeSet = []
for (f1, f2, l) in zip(x.mean_return, x.volatility, ytest ):
    testNodeSet.append( node( (f1, f2 ) , l ) )

yPredict = kHyperPlane(testNodeSet, trainNodeSet, 3)

print(accuracy_score(ytest, yPredict))

# confusion matrix
a = confusion_matrix(ytest, yPredict)
print(a)
tn = a[0][0]
fn = a[1][0]
tp = a[1][1]
fp = a[0][1]

tpr = tp / (tp + fn)
tnr = tn / (tn + fp)

print('TPR = {}, TNR = {}, k = 3'.format(tpr, tnr))
```

```
0.41509433962264153
[[ 8 17]
 [14 14]]
TPR = 0.5, TNR = 0.32, k = 3
```

In [332…

```python
#Strategy check
dfDetail = pd.read_csv('./GOOGL_weekly_return_volatility_detailed.csv')
year2Detail = dfDetail[dfDetail.Year == 2020]
year2Detail = year2Detail.reset_index(drop = True)

## Add label to detail
lMap = labelMapping(year2.Year, year2.Week_Number, yPredict)
temp = []
for (y, w) in zip(year2Detail.Year, year2Detail.Week_Number):
    key = (y, w)
    temp.append(lMap[key])
year2Detail['Label'] = temp
year2Detail = year2Detail[['Year', 'Week_Number', 'Close', 'Label']]

## Cut goo2020
goo2020Week = cutWeek(53, year2Detail)



## trading base on Manhattan KNN label
total = proficCalculator(goo2020Week, 100)
print("Using Label Manhattan KNN: {}".format(total))

## trding BH
firstWeek = goo2020Week[0]
firstClose = firstWeek.Close[0]

lastWeek = goo2020Week[-1]
lastClose = lastWeek.Close[len(lastWeek)-1]

r = 1 + (lastClose - firstClose) / lastClose
total = 100 * r
print("Buy on first day and Sell on last day: {}".format(total))
```

```
Using Label Manhattan KNN: 59.94200186154975
Buy on first day and Sell on last day: 121.17033527942765
```

Question8 - self define

In [426…

```python
def circularBool(f1, f2, r, center):
    inCircle = False
    x = center[0]
    y = center[1]
    r = r * 0.5

    if (f1-x)*(f1-x) + (f2-y)*(f2-y) <= r*r: # (x-x0)**2 + (y-y0)**2 = r*r
        inCircle = True
    return inCircle

def circleFeature(testPointCordinate, trainSet):


    dis = []
    for (f1, f2) in zip(trainSet.mean_return, trainSet.volatility):
        dis.append( math.dist(testPointCordinate, (f1, f2)))
    dis = sorted(dis, reverse=False)
    r = np.mean(dis)


    temp = []
    for (f1, f2, label) in zip(trainSet.mean_return, trainSet.volatility, trainSet.label):
        inCircle = circularBool(f1, f2, r, testPointCordinate)
        if inCircle:
            temp.append(label)
    if sum(temp) >= len(temp) / 2:
        return 1
    else:
        return 0


def circleNeighbor(testSetCor, trainSet):
    yPredict = []
    for (f1, f2) in zip(testSetCor.mean_return, testSetCor.volatility):
        yPredict.append(circleFeature((f1, f2), trainSet))
    return yPredict
```

In [429…

```python
trainData = year1[['mean_return', 'volatility', 'label']]
testPoint = year2.loc[:, ['mean_return', 'volatility']]
yTest = year2.label

yPredict = circleNeighbor(testPoint, trainData)
print(accuracy_score(yTest, yPredict))


# confusion matrix
a = confusion_matrix(ytest, yPredict)
print(a)
tn = a[0][0]
fn = a[1][0]
tp = a[1][1]
fp = a[0][1]

tpr = tp / (tp + fn)
tnr = tn / (tn + fp)

print('TPR = {}, TNR = {}, k = 3'.format(tpr, tnr))
```

```
0.7735849056603774
[[18  7]
 [ 5 23]]
TPR = 0.8214285714285714, TNR = 0.72, k = 3
```

In [430…

```python
#Strategy check
dfDetail = pd.read_csv('./GOOGL_weekly_return_volatility_detailed.csv')
year2Detail = dfDetail[dfDetail.Year == 2020]
year2Detail = year2Detail.reset_index(drop = True)

## Add label to detail
lMap = labelMapping(year2.Year, year2.Week_Number, yPredict)
temp = []
for (y, w) in zip(year2Detail.Year, year2Detail.Week_Number):
    key = (y, w)
    temp.append(lMap[key])
year2Detail['Label'] = temp
year2Detail = year2Detail[['Year', 'Week_Number', 'Close', 'Label']]

## Cut goo2020
goo2020Week = cutWeek(53, year2Detail)



## trading base on Manhattan KNN label
total = proficCalculator(goo2020Week, 100)
print("Using Label Manhattan KNN: {}".format(total))

## trding BH
firstWeek = goo2020Week[0]
firstClose = firstWeek.Close[0]

lastWeek = goo2020Week[-1]
lastClose = lastWeek.Close[len(lastWeek)-1]

r = 1 + (lastClose - firstClose) / lastClose
total = 100 * r
print("Buy on first day and Sell on last day: {}".format(total))
```

```
Using Label Manhattan KNN: 223.5667766598095
Buy on first day and Sell on last day: 121.17033527942765
```