Exercise 3 - KNN

First name: Brian

Last name: Schweigler

Matriculation number: 16-102-071

(1) Take the titanic dataset and use all attributes to predict the class ‘Survived’ with a k-nearest neighbours classifier, which one do you think is the best distance measure? and why?

(a) Manhattan distance

(b) Euclidian distance

(c) Cosine distance

First some imports and preprocessing

In [1]:
```
%load_ext autoreload
%autoreload 2
%matplotlib inline

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn import preprocessing
from sklearn.neighbors import KNeighborsClassifier
import scipy
from mlxtend.evaluate import accuracy_score

df = pd.read_csv("data/titanic.csv", index_col='Name')
pd.set_option('display.max_colwidth', None)
# df.describe(include='all')
knnclassifier_man = KNeighborsClassifier(n_neighbors = 5, metric='manhattan')
knnclassifier_cos = KNeighborsClassifier(n_neighbors = 5, metric='cosine')
knnclassifier_euc = KNeighborsClassifier(n_neighbors = 5, metric='euclidean')
le = preprocessing.LabelEncoder()
bins = [0, 4, 18, 65, 100]
labels = ['Infant', 'Child', 'Adult', 'Elderly']
labels = [1, 2, 3, 4]
df['AgeGroup'] = pd.cut(df['Age'], bins=bins, labels=labels, right=False)
df["Survived"] = le.fit_transform(df["Survived"])
df["Sex"] = le.fit_transform(df["Sex"])
print(df.head(3))
```

```
                                                    Survived  Pclass  Sex  \
Name
Mr. Owen Harris Braund                                     0       3    1
Mrs. John Bradley (Florence Briggs Thayer) Cumings         1       1    0
Miss. Laina Heikkinen                                      1       3    0

                                                     Age  \
Name
Mr. Owen Harris Braund                              22.0
Mrs. John Bradley (Florence Briggs Thayer) Cumings  38.0
Miss. Laina Heikkinen                               26.0

                                                    Siblings/Spouses Aboard  \
Name
Mr. Owen Harris Braund                                                    1
Mrs. John Bradley (Florence Briggs Thayer) Cumings                        1
```

```
Miss. Laina Heikkinen                                          0

                                               Parents/Children Aboard  \
Name
Mr. Owen Harris Braund                                               0
Mrs. John Bradley (Florence Briggs Thayer) Cumings                   0
Miss. Laina Heikkinen                                                0

                                                      Fare  AgeGroup
Name
Mr. Owen Harris Braund                               7.2500         3
Mrs. John Bradley (Florence Briggs Thayer) Cumings  71.2833         3
Miss. Laina Heikkinen                                7.9250         3
```

In [2]:

```
x_train, x_test, y_train, y_test =
train_test_split(df.loc[:, df.columns != 'Survived'], df["Survived"], test_size=0.4, random_sta
knnclassifier_man.fit(x_train, y_train)
knnclassifier_cos.fit(x_train, y_train)
knnclassifier_euc.fit(x_train, y_train)
y_pred_man = knnclassifier_man.predict(x_test)
y_pred_cos = knnclassifier_cos.predict(x_test)
y_pred_euc = knnclassifier_euc.predict(x_test)
accuracy_man = accuracy_score(y_test, y_pred_man)
accuracy_cos = accuracy_score(y_test, y_pred_cos)
accuracy_euc = accuracy_score(y_test, y_pred_euc)
print("Accuracy for KNN (k = 5) with Manhattan Distance Measure: ", round(accuracy_man, 3))
print("Accuracy for KNN (k = 5) with Cosine Distance Measure: ", round(accuracy_cos, 3))
print("Accuracy for KNN (k = 5) with Euclidean Distance Measure: ", round(accuracy_euc, 3))
```

```
Accuracy for KNN (k = 5) with Manhattan Distance Measure:  0.724
Accuracy for KNN (k = 5) with Cosine Distance Measure:  0.724
Accuracy for KNN (k = 5) with Euclidean Distance Measure:  0.707
```

It seems like both Manhattan and Cosine perform better than the Euclidean distance measure. Generally it depends on your data which measure to use:

- Manhattan distance is less intuitive than euclidean and likely to give a higher value than euclidean distance.
- Cosine only looks at the direction of vector, but not their magnitude.
- Euclidean is not scale in-variant.

Build a KNN model with your selected stock / market index using your best distance measure, determine the number attributes that is capable of giving the best prediction. (Select attributes in ascending order(ie 3, 5, 7, ...) and determine the accuracy for the selected attributes, compare the accuracies to find which is the best)

First simply import the dataset and set up the values:

In [3]:

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from itertools import combinations

pd.set_option('display.max_colwidth', None)
stock_df = pd.read_csv("data/Nasdaq.csv", index_col='Date')
print(stock_df.head(3))
# stock_df.describe(include='all')



daily_return = np.empty(stock_df['Close'].shape)
#  From Slides: Daily return (r): Difference in percentage between  price at time t+1 and time
daily_return[0] = float('NaN') # The first
daily_return[1:] = np.ediff1d(stock_df['Close']) / stock_df['Close'][:-1]
stock_df.insert(loc=len(stock_df.columns), column='Daily Return', value=daily_return)
```

```python
binary = (daily_return > 0).astype(float)
stock_df.insert(loc=len(stock_df.columns), column='Binary Decision', value=binary)
stock_df["Binary Decision"] = le.fit_transform(stock_df["Binary Decision"])

stock_df['Rolling Mean 5'] = stock_df['Close'].rolling(5).mean()
stock_df['Rolling Mean 10'] = stock_df['Close'].rolling(10).mean()
stock_df['Rolling Mean 20'] = stock_df['Close'].rolling(20).mean()
stock_df['Rolling Mean 50'] = stock_df['Close'].rolling(50).mean()
stock_df['Rolling Mean 200'] = stock_df['Close'].rolling(200).mean()
stock_df = stock_df.fillna(0)  # NAs replaced with zero

print(stock_df.tail(3))
```

```
                  Open        High         Low       Close   Adj Close  Volume
Date
1971-02-05  100.000000  100.000000  100.000000  100.000000  100.000000       0
1971-02-08  100.839996  100.839996  100.839996  100.839996  100.839996       0
1971-02-09  100.760002  100.760002  100.760002  100.760002  100.760002       0
                    Open          High           Low         Close  \
Date
2021-09-17  15163.360352  15166.559570  14998.730469  15043.969727
2021-09-20  14758.139648  14841.820312  14530.070312  14713.900391
2021-09-21  14803.400391  14847.027344  14696.467773  14779.216797

               Adj Close      Volume  Daily Return  Binary Decision  \
Date
2021-09-17  15043.969727  6682650000     -0.009086                0
2021-09-20  14713.900391  4860630000     -0.021940                0
2021-09-21  14779.216797  3083208000      0.004439                1

            Rolling Mean 5  Rolling Mean 10  Rolling Mean 20  Rolling Mean 50  \
Date
2021-09-17    15106.151953     15191.898926     15143.947949     14875.465586
2021-09-20    15027.816016     15126.937012     15143.909961     14875.705195
2021-09-21    14976.107422     15067.425684     15135.738281     14876.624727

            Rolling Mean 200
Date
2021-09-17      13844.889136
2021-09-20      13856.711787
2021-09-21      13868.721973
```

Smaller df and functions we will use. As Cosine and Manhattan had similar results, we will simply use Manhattan.

In [4]:

```python
smaller_stock_df = stock_df[
    ['Binary Decision', 'Volume', 'Rolling Mean 5', 'Rolling Mean 10',
     'Rolling Mean 20', 'Rolling Mean 50', 'Rolling Mean 200']]
columns = smaller_stock_df.loc[:, smaller_stock_df.columns != 'Binary Decision'].columns

def knn_accuracy(cols):
    x = smaller_stock_df[cols]
    y = smaller_stock_df['Binary Decision']
    test_size = 100/smaller_stock_df.shape[0]

    x_stock_train, x_stock_test, y_stock_train, y_stock_test = \
        train_test_split(x, y, test_size=test_size, shuffle=False, random_state=6)

    stock_knnclassifier_man = KNeighborsClassifier(metric='manhattan')
    stock_knnclassifier_man.fit(x_stock_train, y_stock_train)
    stock_y_pred_man = stock_knnclassifier_man.predict(x_stock_test)

    return accuracy_score(y_stock_test, stock_y_pred_man)

def all_combinations():
    for i in range(1, len(columns)):
        features_list = [list(c) for c in combinations(columns, i)]
```

```python
        for features in features_list:
            yield features

accuracies = pd.DataFrame(
    map(lambda c: [c, knn_accuracy(c)], all_combinations()),
    columns=['Feature Combination', 'Accuracy'],
).sort_values(by='Accuracy', ascending=False)

accuracies
```

Out[4]:

	Feature Combination	Accuracy
53	[Rolling Mean 5, Rolling Mean 10, Rolling Mean 50, Rolling Mean 200]	0.60
14	[Rolling Mean 5, Rolling Mean 200]	0.60
33	[Rolling Mean 5, Rolling Mean 10, Rolling Mean 200]	0.59
51	[Rolling Mean 5, Rolling Mean 10, Rolling Mean 20, Rolling Mean 50]	0.58
61	[Rolling Mean 5, Rolling Mean 10, Rolling Mean 20, Rolling Mean 50, Rolling Mean 200]	0.57
...	...	...
13	[Rolling Mean 5, Rolling Mean 50]	0.47
31	[Rolling Mean 5, Rolling Mean 10, Rolling Mean 20]	0.45
11	[Rolling Mean 5, Rolling Mean 10]	0.44
12	[Rolling Mean 5, Rolling Mean 20]	0.44
2	[Rolling Mean 10]	0.44

62 rows × 2 columns

We can see that "[Rolling Mean 5, Rolling Mean 10, Rolling Mean 50, Rolling Mean 200]" works best, but "[Rolling Mean 5, Rolling Mean 200]" have similar performance using only two variables. Thus we would use the latter as they have a higher performance. Furthermore, intuitively we can say that Rolling Mean 200 gives us the general trend of the stock, while Rolling Mean 5 shows us the short-term changes.