Tugas Materi 4

Zul Fauzi Oktaviansyah

2110181056

3 – D4 IT - B

Code 1 & 2

In [2]:
```
#1 #2
dataset = pd.read_csv('titanic.csv')
test_dataset = pd.read_csv('titanic_test.csv')
```

Membaca data csv titanic & titanic_test

Code 3

In [3]:
```
#3
train_data = dataset[['Age', 'Fare']].dropna()
```

In [4]:
```
#3
naIsTrue = dataset[['Age', 'Fare']].isna()
tempIsNa = naIsTrue[naIsTrue["Age"] == True]
tempIsNa1 = naIsTrue[naIsTrue["Fare"] == True]

pos_missing_train = np.append(tempIsNa1.index, tempIsNa.index)
```

Dropna() digunakan untuk menghapus 1 baris apabila
di salah satu kolomnya terdapat missing values,
Kemudian isna() digunakan untuk mengubah NaN menjadi true dan
Selain NaN menjadi false, index digunakan untuk mengambil semua posisi index
Dan append digunakan untuk menggabungkan banyak data.

Code 4

In [5]:
```
#4
test_data = test_dataset[['Age', 'Fare']].dropna()
```

In [6]:
```
#4
naIsTrue2 = test_dataset[['Age', 'Fare']].isna()
tempIsNa2 = naIsTrue2[naIsTrue2["Age"] == True]
tempIsNa3 = naIsTrue2[naIsTrue2["Fare"] == True]

pos_missing_test = np.append(tempIsNa3.index, tempIsNa2.index)
```

Dropna() digunakan untuk menghapus 1 baris apabila
di salah satu kolomnya terdapat missing values,
Kemudian isna() digunakan untuk mengubah NaN menjadi true dan
Selain NaN menjadi false, index digunakan untuk mengambil semua posisi index
Dan append digunakan untuk menggabungkan banyak data.

Code 5

```
In [11]: train_label = dataset.drop(pos_missing_train)
         train_label = train_label[['Survived']]
         train_label
```

Drop() digunakan sebagai menghapus 1 baris data sesuai index yang ingin dihapus.

Code 6

```
In [9]: label_data = pd.read_csv('titanic_testlabel.csv')
        test_label = label_data.drop(pos_missing_test)
        test_label = test_label[['Survived']]
```

Drop() digunakan sebagai menghapus 1 baris data sesuai index yang ingin dihapus.

Code 7

In [12]:
```
def min_max_scaling(data):

    data_norm = data.copy()

    for column in data_norm.columns:
        data_norm[column] = (data_norm[column] - data_norm[column].min()) / (data_norm[column].max() - data_norm[column].min())

    return data_norm
```

Fungsi diatas digunakan sebagai menormalisasikan suatu data dengan Rumus metode min max

Code 7

In [33]:
```
norm_train_data = min_max_scaling(train_data)
min_train_age = np.array(norm_train_data['Age']).min()
max_train_age = np.array(norm_train_data['Age']).max()
min_train_fare = np.array(norm_train_data['Fare']).min()
max_train_fare = np.array(norm_train_data['Fare']).max()
```

Np.array() digunakan sebagai mengubah sebuah kumpulan data menjadi Bertipe numpy array, min() untuk mencari nilai minimum dan max() maksimum.

Code 8

In [36]:
```
norm_test_data = min_max_scaling(test_data)
min_test_age = np.array(norm_test_data['Age']).min()
max_test_age = np.array(norm_test_data['Age']).max()
min_test_fare = np.array(norm_test_data['Fare']).min()
max_test_fare = np.array(norm_test_data['Fare']).max()
```

Np.array() digunakan sebagai mengubah sebuah kumpulan data menjadi Bertipe numpy array, min() untuk mencari nilai minimum dan max() maksimum.

Code 9 / K = 1

```
In [39]: from sklearn.neighbors import KNeighborsClassifier

In [40]: kNN=KNeighborsClassifier(n_neighbors=1, weights='distance')

In [41]: kNN.fit(norm_train_data, train_label)
         class_result = kNN.predict(norm_test_data)
         class_result
```

Code diatas digunakan untuk memanggil KNN pada library sklearn,
Fit() digunakan untuk mentraining sebuah model, predict() digunakan untuk Memprediksi hasil.

Code 10

```
In [42]: precision_ratio=kNN.score(norm_test_data, test_label)

In [43]: error_ratio=1-precision_ratio

In [44]: error_ratio
Out[44]: 0.4441087613293051
```

Code diatas untuk mendapatkan error ratio dengan mengkalkulasi selisih dari Ratio ketepatan dari hasil yg diprediksi dan yang sebenarnya.

K = 2

In [55]:
```
kNN=KNeighborsClassifier(n_neighbors=2, weights='distance')
```

In [56]:
```
kNN.fit(norm_train_data, train_label)
class_result = kNN.predict(norm_test_data)
class_result
```

In [57]:
```
precision_ratio=kNN.score(norm_test_data, test_label)
```

In [58]:
```
error_ratio=1-precision_ratio
```

In [59]:
```
error_ratio
```

Out[59]: 0.4441087613293051

K = 3

In [60]:
```
kNN=KNeighborsClassifier(n_neighbors=3, weights='distance')
```

In [61]:
```
kNN.fit(norm_train_data, train_label)
class_result = kNN.predict(norm_test_data)
class_result
```

In [62]:
```
precision_ratio=kNN.score(norm_test_data, test_label)
```

In [63]:
```
error_ratio=1-precision_ratio
```

In [64]:
```
error_ratio
```

Out[64]: 0.43504531722054385

K = 4

In [65]:
```
kNN=KNeighborsClassifier(n_neighbors=4, weights='distance')
```

In [66]:
```
kNN.fit(norm_train_data, train_label)
class_result = kNN.predict(norm_test_data)
class_result
```

In [67]:
```
precision_ratio=kNN.score(norm_test_data, test_label)
```

In [68]:
```
error_ratio=1-precision_ratio
```

In [69]:
```
error_ratio
```

Out[69]:
```
0.45317220543806647
```

K = 5

In [70]:
```
kNN=KNeighborsClassifier(n_neighbors=5, weights='distance')
```

In [71]:
```
kNN.fit(norm_train_data, train_label)
class_result = kNN.predict(norm_test_data)
class_result
```

In [72]:
```
precision_ratio=kNN.score(norm_test_data, test_label)
```

In [73]:
```
error_ratio=1-precision_ratio
```

In [74]:
```
error_ratio
```

Out[74]: 0.43202416918429

K = 6

In [75]:
```
kNN=KNeighborsClassifier(n_neighbors=6, weights='distance')
```

In [76]:
```
kNN.fit(norm_train_data, train_label)
class_result = kNN.predict(norm_test_data)
class_result
```

In [77]:
```
precision_ratio=kNN.score(norm_test_data, test_label)
```

In [78]:
```
error_ratio=1-precision_ratio
```

In [79]:
```
error_ratio
```

Out[79]:
```
0.43202416918429
```

K = 7

```
kNN=KNeighborsClassifier(n_neighbors=7, weights='distance')
```

```
kNN.fit(norm_train_data, train_label)
class_result = kNN.predict(norm_test_data)
class_result
```

In [82]:
```
precision_ratio=kNN.score(norm_test_data, test_label)
```

In [83]:
```
error_ratio=1-precision_ratio
```

In [84]:
```
error_ratio
```

Out[84]: 0.3867069486404834

K = 8

```
In [85]: kNN=KNeighborsClassifier(n_neighbors=8, weights='distance')

In [86]: kNN.fit(norm_train_data, train_label)
         class_result = kNN.predict(norm_test_data)
         class_result

In [87]: precision_ratio=kNN.score(norm_test_data, test_label)

In [88]: error_ratio=1-precision_ratio

In [89]: error_ratio
Out[89]: 0.41389728096676737
```

K = 9

In [90]:
```
kNN=KNeighborsClassifier(n_neighbors=9, weights='distance')
```

In [91]:
```
kNN.fit(norm_train_data, train_label)
class_result = kNN.predict(norm_test_data)
class_result
```

In [92]:
```
precision_ratio=kNN.score(norm_test_data, test_label)
```

In [93]:
```
error_ratio=1-precision_ratio
```

In [94]:
```
error_ratio
```

Out[94]:
```
0.3987915407854985
```

K = 10

In [95]:
```
kNN=KNeighborsClassifier(n_neighbors=10, weights='distance')
```

In [96]:
```
kNN.fit(norm_train_data, train_label)
class_result = kNN.predict(norm_test_data)
class_result
```

In [97]:
```
precision_ratio=kNN.score(norm_test_data, test_label)
```

In [98]:
```
error_ratio=1-precision_ratio
```

In [99]:
```
error_ratio
```

Out[99]: 0.4169184290030211