Titanic.csv

First we read the dataset:

```
titanic = pd.read_csv("titanic.csv",
                      sep=',',
                      header=0,
)
pd.set_option('display.max_columns', None)
```

DataFrame has the following columns:

PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked

Dropped columns from all tests (since these 4 play no role in defying if one survived or not):

```
drop_columns = ["PassengerId", "Name", "Ticket", "Fare"]
titanic = titanic.drop(drop_columns, axis=1)
```

Dropped columns due to NaN values (hasCabin is a binary 0/1 if Cabin is NaN/has value(will explain later)):

```
drop_columns_with_empty_values = ["Age", "Embarked", "hasCabin"]
titanic_no_nan = titanic.drop(drop_columns_with_empty_values, axis=1)
```

Also changed column “Sex” to “Male” and “Female” (One-Hot_encoding):

```
#   One-Hot-Encoding column: "Sex" into "Male" or "Female"
titanic["Male"] = 0
titanic["Female"] = 0
titanic["Male"][titanic["Sex"] == "male"] = 1
titanic["Female"][titanic["Sex"] == "female"] = 1
titanic = titanic.drop("Sex", axis=1)
```

Filling NaN for 2nd test:

```
#   Change Cabin to binary 0/1 if Cabin = Nan then hasCabin = 0, else hasCabin = 1
titanic['hasCabin'] = 0
titanic["Cabin"] = titanic["Cabin"].fillna(0)
titanic["hasCabin"][titanic["Cabin"] != 0] = 1
titanic = titanic.drop("Cabin", axis=1)
```

```
#   Replace Embarked NaN with most common value
# print(titanic.groupby('Embarked').size())
# Embarked
# C    168
```

```
# Q      77
# S     644
embarked = titanic.groupby('Embarked').size().sort_values(ascending=False)
titanic["Embarked"] = titanic["Embarked"].fillna(list(dict(embarked))[0])
# and then map these values to integers
titanic['Embarked'] = titanic['Embarked'].map({'S': 0, 'C': 1, 'Q': 2}).astype(int)
```

For Age there were 2 options: Mean and Median. We chose Mean from .describe():

```
#   Replace Age NaN with mean OR median
mean = round(float(pd.DataFrame(titanic.describe())[['Age']].loc['mean']), 2)
# print(mean)                         # 29.7
# print(titanic["Age"].median())  # 28.0
titanic["Age"] = titanic["Age"].fillna(mean)
# titanic["Age"] = titanic["Age"].fillna(titanic["Age"].median())
```

Now we have 2 DataFrames, Titanic and Titanic_no_nan.

```
Titanic

Survived      int64
Pclass        int64
Age         float64
SibSp         int64
Parch         int64
Embarked      int32
hasCabin      int64
Male          int64
Female        int64
```

```
titanic_no_nan

Survived    int64
Pclass      int64
SibSp       int64
Parch       int64
Male        int64
Female      int64
```

(Example from titanic):

Then we split the dataset:

```
X = titanic.iloc[:, 1:len(titanic)]
y = titanic.iloc[:, 0]
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=42, train_size=0.75)
```

Then we normalized the features using MinMaxScaler and transform the data.

```
scaler = MinMaxScaler()
scaler.fit(X=X_train)
scaled_features = pd.DataFrame(scaler.transform(X))
```

Then we ran k-NN for:

```
n_neighbors = 1:200

weights     = [“uniform”, “distance”]

p           = [1, 2, 3]
```

```
knn = KNeighborsClassifier(
    n_neighbors=neighbours,
    weights=weight,
    metric='minkowski',
    p=p_value)
```

Results:

Dropped columns with NaN

p	Accuracy	Recall	Precision	Best F1	Neighbors count of best F1
2	0.811659	0.804902	0.799891	0.802104	**59**
2	0.789238	0.780852	0.777461	0.778998	**5**
1	0.816143	0.816058	0.796076	0.802804	**28**
1	0.789238	0.780852	0.777461	0.778998	**5**
3	0.811659	0.804175	0.801778	0.802904	**64**
3	0.789238	0.780852	0.777461	0.778998	**5**

Filled columns with NaN

p	Accuracy	Recall	Precision	Best F1	Neighbors count of best F1
2	0.825112	0.822334	0.809198	0.814218	**1**
2	0.825112	0.822334	0.809198	0.814218	**1**
1	0.820628	0.814408	0.809240	0.811528	**1**
1	0.820628	0.814408	0.809240	0.811528	**1**
3	0.820628	0.818258	0.803580	0.809010	**1**
3	0.820628	0.818258	0.803580	0.809010	**1**

Plot: