In [1]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

In [2]:
```
df = pd.read_csv("titanic.csv", header = 0, index_col = 0)
df.replace({'CabinReduced': {'n': np.nan}}, inplace = True)
df
```

Out[2]:

	pclass	survived	name	sex	age	sibsp	parch	ticket	fare	cabin	embarked	boat	body	home.dest
0	1	1	Allen, Miss. Elisabeth Walton	female	29.0000	0	0	24160	211.3375	B5	S	2	NaN	St Louis, MO
1	1	1	Allison, Master. Hudson Trevor	male	0.9167	1	2	113781	151.5500	C22 C26	S	11	NaN	Montreal, PQ / Chesterville, ON
2	1	0	Allison, Miss. Helen Loraine	female	2.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
3	1	0	Allison, Mr. Hudson Joshua Creighton	male	30.0000	1	2	113781	151.5500	C22 C26	S	NaN	135.0	Montreal, PQ / Chesterville, ON
4	1	0	Allison, Mrs. Hudson J C (Bessie Waldo Daniels)	female	25.0000	1	2	113781	151.5500	C22 C26	S	NaN	NaN	Montreal, PQ / Chesterville, ON
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1304	3	0	Zabour, Miss. Hileni	female	14.5000	1	0	2665	14.4542	NaN	C	NaN	328.0	NaN
1305	3	0	Zabour, Miss. Thamine	female	NaN	1	0	2665	14.4542	NaN	C	NaN	NaN	NaN
1306	3	0	Zakarian, Mr. Mapriededer	male	26.5000	0	0	2656	7.2250	NaN	C	NaN	304.0	NaN
1307	3	0	Zakarian, Mr. Ortin	male	27.0000	0	0	2670	7.2250	NaN	C	NaN	NaN	NaN
1308	3	0	Zimmerman, Mr. Leo	male	29.0000	0	0	315082	7.8750	NaN	S	NaN	NaN	NaN

1309 rows × 15 columns

Funkcja **train_test_split** dzieli zbiór danych (przekazany jako lista, numpy array lub DataFrame) na zbiór treningowy i testowy. Jako parametry można podać jaki odsetek całego zbioru ma zostać uwzględniony przy tworzeniu zbioru (treningowego lub testowego). Jeśli któraś z wartości nie jest podana, to domyślnie odsetek drugiego jest wyliczany jako dopełnienie do 1 odsetka pierwszego zbioru.

Domyślnie (bez podania żadnej z tych wielkości): Zbiór treningowy - 75%, zbiór testowy - 25%

Funkcja zwraca listę podzielonych zbiorów

In [3]:
```
col_name = ['cabin', 'CabinReduced', 'sex']
```

In [4]:
```
X = df[col_name]
y = df[['survived']]
```

In [5]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3, random_state = 0)
```

In [6]:
```
names = ['X_train', 'X_test', 'y_train', 'y_test']
var = [X_train, X_test, y_train, y_test]

for i in range(4):
    print(f"Rozmiary zbioru {names[i]}: {var[i].shape}")
```

```
Rozmiary zbioru X_train: (916, 3)
Rozmiary zbioru X_test: (393, 3)
Rozmiary zbioru y_train: (916, 1)
Rozmiary zbioru y_test: (393, 1)
```

Zbiory treningowe mają po 916 wierszy, a testowe - 393. Stanowi to odpowiednio 70% i 30% całego zbioru. Suma ilości wierszy w zbiorach treningowych i testowych jest równa ilości rekordów w całym zbiorze danych.

Sprawdzamy czy rozkład etykiet jest równomierny

In [7]:
```
print("Ilość unikalnych etykiet w zbiorze testowym, które nie występują w treningowym")
for name in col_name:
    Unique_test = [x for x in X_test[name].unique() if x not in X_train[name].unique()]
    print(f"{name}: {len(Unique_test)}")
```

```
Ilość unikalnych etykiet w zbiorze testowym, które nie występują w treningowym
cabin: 37
CabinReduced: 1
sex: 0
```

Aby wyciągnąć dobre wnioski, sprawdźmy ilość unikalnych etykiet dla danych zmiennych (w zależnosci od tego czy jest to zbiór testowy czy treningowy)

In [8]:
```
for name in col_name:
    print(f"Ilość unikalnych etykiet dla zmiennej {name}:")
    print(f"Dla zbioru testowego: {len(X_test[name].unique())}")
    print(f"Dla zbioru treningowego: {len(X_train[name].unique())}")
    print(f"Dla całego zbioru: {len(df[name].unique())}")
    print()
```

```
Ilość unikalnych etykiet dla zmiennej cabin:
Dla zbioru testowego: 71
Dla zbioru treningowego: 151
Dla całego zbioru: 187

Ilość unikalnych etykiet dla zmiennej CabinReduced:
Dla zbioru testowego: 8
Dla zbioru treningowego: 9
Dla całego zbioru: 9

Ilość unikalnych etykiet dla zmiennej sex:
Dla zbioru testowego: 2
Dla zbioru treningowego: 2
Dla całego zbioru: 2
```

Dla zmiennej cabin w zasadzie połowa etykiet, które występują w zbiorze testowym, nie występują w zbiorze treningowym. Oznacza to, że rozkład może być zaburzony i, o ile zmienna cabin będzie kluczowa przy dalszej analizie, model może dać słabsze wyniki dla zbioru testowego (nie będzie możliwości odpowiedniego uwzględnienia zmiennych w trakcie uczenia).

Jednakże przed tym uchronić nas może redukcja, gdzie ilość etykiet została drastycznie pomniejszona, a rozkład w zbiorze testowym i treningowym jest już równomierny (CabinReduced).

Mapowanie zmiennych:

In [9]:
```
#tablica słowników do mapowania
dicts = []

for name in col_name:
    d = dict((name, count) for count, name in enumerate(df[name].unique()))
    dicts.append(d)
```

In [10]:
```
new_name = ['cabin_map', "cabin_reduced_map", "sex_map"]

X_train2 = pd.DataFrame()
X_test2 = pd.DataFrame()

for i in range(3):
    X_test2[new_name[i]] = X_test[col_name[i]].map(dicts[i], na_action = 'ignore').values
    X_train2[new_name[i]] = X_train[col_name[i]].map(dicts[i], na_action = 'ignore').values

X_test2
```

Out[10]:

	cabin_map	cabin_reduced_map	sex_map
0	NaN	NaN	1
1	NaN	NaN	0
2	NaN	NaN	1
3	NaN	NaN	1
4	NaN	NaN	1
...	...	...	...
388	NaN	NaN	1
389	NaN	NaN	0
390	NaN	NaN	1
391	186.0	7.0	1
392	NaN	NaN	0

393 rows × 3 columns

Liczba brakujących wartości dla nowych etykiet:

In [11]:
```
print("Dla zbioru testowego:")
print(X_test2[new_name].isnull().sum())
print()
print("Dla zbioru treningowego:")
print(X_train2[new_name].isnull().sum())
```

```
Dla zbioru testowego:
cabin_map            312
cabin_reduced_map    312
sex_map                0
dtype: int64

Dla zbioru treningowego:
cabin_map            702
cabin_reduced_map    702
sex_map                0
dtype: int64
```

Porównajmy to z sytuacją przed mapowaniem:

In [12]:
```
print("Dla zbioru testowego:")
print(X_test[col_name].isnull().sum())
print()
print("Dla zbioru treningowego:")
print(X_train[col_name].isnull().sum())
```

```
Dla zbioru testowego:
cabin           312
CabinReduced    312
sex               0
dtype: int64

Dla zbioru treningowego:
cabin           702
CabinReduced    702
sex               0
dtype: int64
```

A zatem zmapowanie etykiet nie dotyczyło wartości NaN.
Na numeryczne zostały zmapowane tylko wartości niebrakujące.

Zmiana wartości brakujących na 0

In [13]:
```
for name in new_name:
    X_test2.replace({np.nan : 0}, inplace = True)
    X_train2.replace({np.nan : 0}, inplace = True)

X_test2
```

Out[13]:

	cabin_map	cabin_reduced_map	sex_map
0	0.0	0.0	1
1	0.0	0.0	0
2	0.0	0.0	1
3	0.0	0.0	1
4	0.0	0.0	1
...	...	...	...
388	0.0	0.0	1
389	0.0	0.0	0
390	0.0	0.0	1
391	186.0	7.0	1
392	0.0	0.0	0

393 rows × 3 columns

Takie działanie może jednak nie być najlepszym wyjściem, ponieważ 0 zostało już wykorzystane do mapowania i oznaczało inną wartość. Wobec tego można postąpić inaczej: mapowanie (przy użyciu funkcji enumerate) wykonać **zaczynając liczenie od 1**. Wtedy zmapowanie wartości NaN na 0 będzie jednoznaczne.

In [14]:
```
#tablica słowników do mapowania
dicts = []

for name in col_name:
    d = dict((name, count) for count, name in enumerate(df[name].unique(), 1))
    dicts.append(d)

#mapowanie
for i in range(3):
    X_test2[new_name[i]] = X_test[col_name[i]].map(dicts[i], na_action = 'ignore').values
    X_train2[new_name[i]] = X_train[col_name[i]].map(dicts[i], na_action = 'ignore').values

    #zmiana wartości brakujących na 0
    X_test2.replace({np.nan : 0}, inplace = True)
    X_train2.replace({np.nan : 0}, inplace = True)

X_train2
```

Out[14]:

	cabin_map	cabin_reduced_map	sex_map
0	0.0	0.0	1
1	0.0	0.0	1
2	0.0	0.0	1
3	0.0	0.0	2
4	0.0	0.0	1
...	...	...	...
911	0.0	0.0	1
912	0.0	0.0	2
913	0.0	0.0	1
914	0.0	0.0	1
915	0.0	0.0	1

916 rows × 3 columns

Porównanie ilości unikalnych etykiet przed i po mapowaniu:

In [15]:
```
for i in range(3):

    name = col_name[i]
    map_name = new_name[i]

    print(f"Ilość unikalnych etykiet dla zmiennej {name}")
    print("Przed mapowaniem:")
    print(f"Zbiór testowy: {len(X_test[name].unique())}")
    print(f"Zbiór treningowy: {len(X_train[name].unique())}")
    print()
    print("Po mapowaniu:")
```

```
    print(f"Zbiór testowy: {len(X_test2[map_name].unique())}")
    print(f"Zbiór treningowy: {len(X_train2[map_name].unique())}")
    print('==================================\n')
```

```
Ilość unikalnych etykiet dla zmiennej cabin
Przed mapowaniem:
Zbiór testowy: 71
Zbiór treningowy: 151

Po mapowaniu:
Zbiór testowy: 71
Zbiór treningowy: 151
==================================

Ilość unikalnych etykiet dla zmiennej CabinReduced
Przed mapowaniem:
Zbiór testowy: 8
Zbiór treningowy: 9

Po mapowaniu:
Zbiór testowy: 8
Zbiór treningowy: 9
==================================

Ilość unikalnych etykiet dla zmiennej sex
Przed mapowaniem:
Zbiór testowy: 2
Zbiór treningowy: 2

Po mapowaniu:
Zbiór testowy: 2
Zbiór treningowy: 2
==================================
```

Jak widać, wartości przed i po mapowaniu nie uległy zmianie. Udało się zatem zakodować etykiety jako wartości liczbowe, dzięki czemu będą mogły zostać użyte przy tworzeniu modelu.

Biorąc pod uwagę, że wartości zostały zakodowane jednoznacznie, to mapowanie nie zmieni to wpływu na jakość modelu. Wpływ może mieć za to redukcja zmiennej cabin, gdzie ilość kategorii została drastycznie zmniejszona. Redukcja tej zmiennej zapewniła nam także to, że rozkład etykiet w zbiorze testowym i treningowym będzie równomiernym, czego nie udało się osiągnąć przed redukcją.