Machine Learning Session No.8

Summary 28-05-2024

- Feature Elimination Process:- Feature elimination is a critical step in the process of building predictive models in data science and machine learning. It involves removing features (variables or columns) from the dataset that do not contribute significantly to the predictive power of the model. This process helps to simplify the model, improve its performance, and reduce overfitting.
- Here's an overview of the feature elimination process we used:

  1. Initial Feature Set:
     - ➢ We started with a set of features including Passenger ID, Name, Ticket Number, Fare, Pclass, Sex, Age, SibSp, Parch, Cabin, and Embarked.
  2. Eliminating Non-contributory Features:
     - ➢ Passenger ID, Name, and Ticket Number: These features were eliminated because they are unique identifiers or textual data that do not have a meaningful impact on the survival rate.
     - ➢ Fare: Although a numerical feature, it was found to have a minimal impact on the survival outcome and was thus excluded.
  3. Exploratory Data Analysis (EDA):
     - ➢ Using EDA, we investigated the relationship between each feature and the survival rate. This analysis helped us identify which features were important for predicting survival.
  4. Selection of Key Features:
     - ➢ The EDA concluded that the following features are significant in determining survival: Pclass, Sex, Age, SibSp, Parch, and Embarked.
  5. Handling Missing Data:
     - ➢ Cabin: Although the Cabin feature might provide valuable information, it was excluded from the model because a substantial portion of the data was missing, making it unreliable for analysis.
- By following this feature elimination process, we refined our dataset to include only features likely to improve the model's predictive accuracy, thereby making our analysis more efficient and effective.

- Let's Start the code
- Import Libraries and Load Dataset
- **Import pandas**: This imports the pandas library, which is used for data manipulation and analysis
- **Load Dataset**: The Titanic dataset is loaded from a CSV file into a pandas DataFrame named **dataset**

```
import pandas
```

```
dataset = pandas.read_csv('titanic_train.csv')
```

```
In [8]:  1 dataset
```

Out[8]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss.	female	26.0	0	0	STON/O2.	7.9250	NaN	S

- **Filter by Class**: We check which passengers are in 1st class

```
In [7]:  1 dataset['Pclass'] == 1

Out[7]: 0      False
        1       True
        2      False
        3       True
        4      False
               ...
        886    False
        887     True
        888    False
        889     True
        890    False
        Name: Pclass, Length: 891, dtype: bool
```

- **Check Type**: We confirm that the **dataset** is a DataFrame (a type of table)

```
In [9]:  1 type(dataset)

Out[9]: pandas.core.frame.DataFrame
```

- **1st Class Ages**: We get the ages of 1st class passengers

```
In [10]:  1 dataset[ dataset['Pclass'] == 1 ]
```

Out[10]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily	female	35.0	1	0	113803	53.1000	C123	S

```
In [11]:  1 dataset[ dataset['Pclass'] == 1 ]  ['Age']

Out[11]: 1      38.0
         3      35.0
         6      54.0
         11     58.0
         23     28.0
                ...
         871    47.0
         872    33.0
         879    56.0
         887    19.0
```

- **Mean Age**: We calculate the average age of 1st class passengers

```
In [12]:  1 dataset[ dataset['Pclass'] == 1 ]  ['Age'].mean()

Out[12]: 38.233440860215055

In [ ]:  1
```

- **Integer Mean Age**: We convert that average age to an integer (whole number)

```
In [13]:  1 int ( dataset[ dataset['Pclass'] == 1 ]  ['Age'].mean() )

Out[13]: 38
```

- After that we store this in age1 variable and also we find the null value

```
In [14]:  1 age1 = int ( dataset[ dataset['Pclass'] == 1 ]  ['Age'].mean() )

In [18]:  1 dataset [ dataset['Pclass'] == 1 ]['Age'].isnull()

Out[18]: 1      False
         3      False
         6      False
         11     False
         23     False
                ...
         871    False
         872    False
         879    False
         887    False
         889    False
         Name: Age, Length: 216, dtype: bool
```

- After that we Calculate Mean Ages for All Classes
- **Mean Ages**: We find the average ages for 1st, 2nd, and 3rd class passengers and save them as **age1**, **age2**, and **age3**

```
In [14]:  age1 = int ( dataset[ dataset['Pclass'] == 1 ]  ['Age'].mean() )

In [21]:  age2 = int ( dataset[ dataset['Pclass'] == 2 ]  ['Age'].mean() )

In [23]:  age3 = int ( dataset[ dataset['Pclass'] == 3 ]  ['Age'].mean() )

In [ ]:
```

- **Check for Missing Age**: **numpy.isnan([])** is used to check if an age is missing (NaN)
- To utilize the numpy isnan function for identifying missing age values

```
In [29]:  numpy.isnan([])

Out[29]:  array([], dtype=bool)
```

- **ageMissing Function**: This function takes a passenger's class and age. If the age is missing, it returns the average age for that class. Otherwise, it returns the given age
- Define a function that takes passenger class and age as inputs, and returns the mean age of the class if the age is missing

```
In [30]:
def ageMissing(p, a):
    Pclass = p
    age = a

    if numpy.isnan(age):
        if Pclass == 1:
            return age1
        elif Pclass == 2:
            return age2
        elif Pclass == 3:
            return age3
        else:
            return 0

    else:
        return age
```

- Define Two Local Variables
    - If **Pclass** is 1, the function returns **age1**, a predefined average age for first-class passengers
    - If **Pclass** is 2, the function returns **age2**, a predefined average age for second-class passengers
    - If **Pclass** is 3, the function returns **age3**, a predefined average age for third-class passengers
    - If **Pclass** is not 1, 2, or 3 (which ideally should not happen in this context), the function returns 0
- We retrieve the first row of the **dataset**

```
In [32]:    1  dataset.iloc[0]

Out[32]: PassengerId                          1
         Survived                             0
         Pclass                               3
         Name           Braund, Mr. Owen Harris
         Sex                               male
         Age                               22.0
         SibSp                                1
         Parch                                0
         Ticket                       A/5 21171
         Fare                              7.25
         Cabin                              NaN
         Embarked                             S
         Name: 0, dtype: object
```

- We retrieve the 'Pclass' and 'Age' columns from the first row of the **dataset**

```
In [34]:    1  dataset.iloc[0][ ['Pclass', 'Age'] ]

Out[34]: Pclass       3
         Age       22.0
         Name: 0, dtype: object
```

- The **ageMissing** function with a third-class passenger (Pclass 3) and an age of 22.0. Since the age is not missing, it returns 22.0

```
In [35]:    1  ageMissing(3, 22.0)

Out[35]: 22.0
```

- The **ageMissing** function with the 'Pclass' and 'Age' values from the row with index 888
- If 'Age' is missing (NaN), it returns the average age based on 'Pclass'; otherwise, it returns the actual age

```
In [38]:  1 ageMissing( dataset.iloc[888][ 'Pclass' ], dataset.iloc[888][ 'Age' ])
Out[38]: 25
```

- After that we retrieve the 'Pclass' and 'Age' columns for all rows in the **dataset**

```
In [42]:  1 dataset[[ 'Pclass' , 'Age']]
Out[42]:
```

	Pclass	Age
0	3	22.0
1	1	38.0
2	3	26.0
3	1	35.0
4	3	35.0

- After that we replace the null values with the age by using Apply function

```
In [43]:  1 dataset[[ 'Pclass' , 'Age']].apply( ageMissing )
Out[43]:
```

	Pclass	Age
0	3	22.0
1	1	38.0
2	3	26.0
3	1	35.0
4	3	35.0
...	...	...
886	2	27.0
887	1	19.0
888	3	NaN

- After that we create the same function age missing and pass one variable d

```
In [54]:
def ageMissing( d ):
    print(d)
    Pclass = d[0]
    age = d[1]

    if numpy.isnan(age):
        if Pclass == 1:
            return age1
        elif Pclass == 2:
            return age2
        elif Pclass == 3:
            return age3
        else:
            return 0

    else:
        return age
```

- This line of code updates the 'Age' column in the **dataset** DataFrame by applying the **ageMissing** function to each row, which fills missing age values based on passenger class

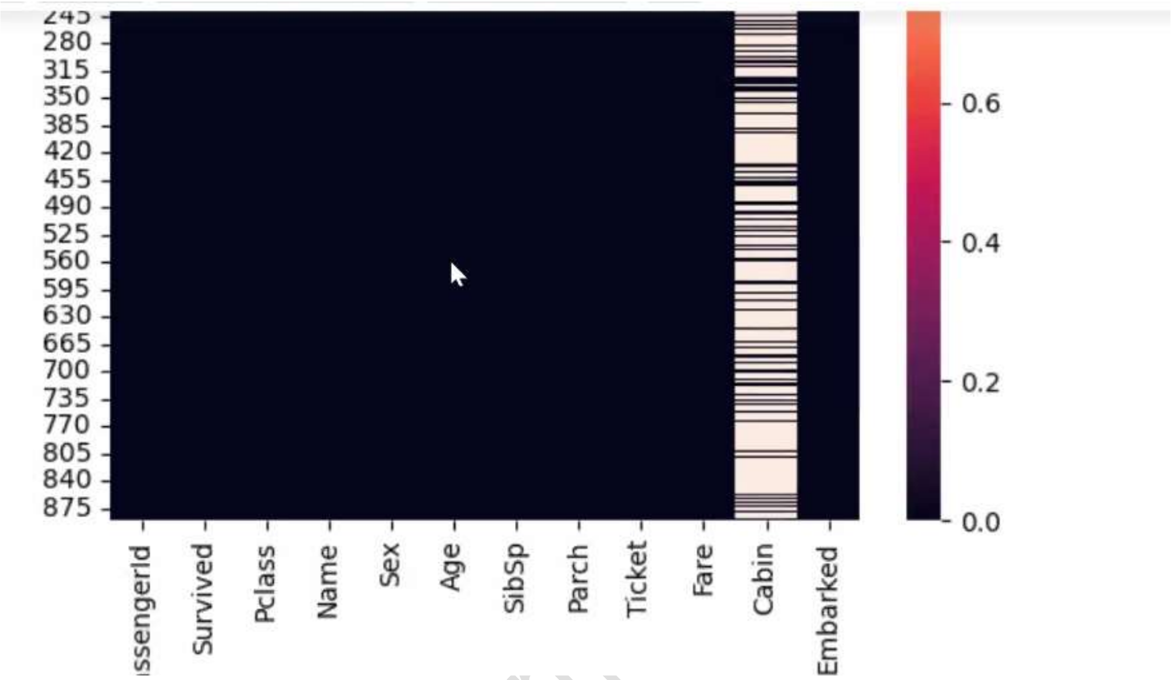
```
In [61]: dataset[[ 'Pclass' , 'Age']].apply(ageMissing ,  axis=1  )

Out[61]: 0      22.0
         1      38.0
         2      26.0
         3      35.0
         4      35.0
                ...
         886    27.0
         887    19.0
         888    25.0
         889    26.0
         890    32.0
         Length: 891, dtype: float64
```

- After that store this value in one variable

```
In [64]: dataset['Age'] = dataset[[ 'Pclass' , 'Age']].apply( ageMissing ,  axis=1  )
```

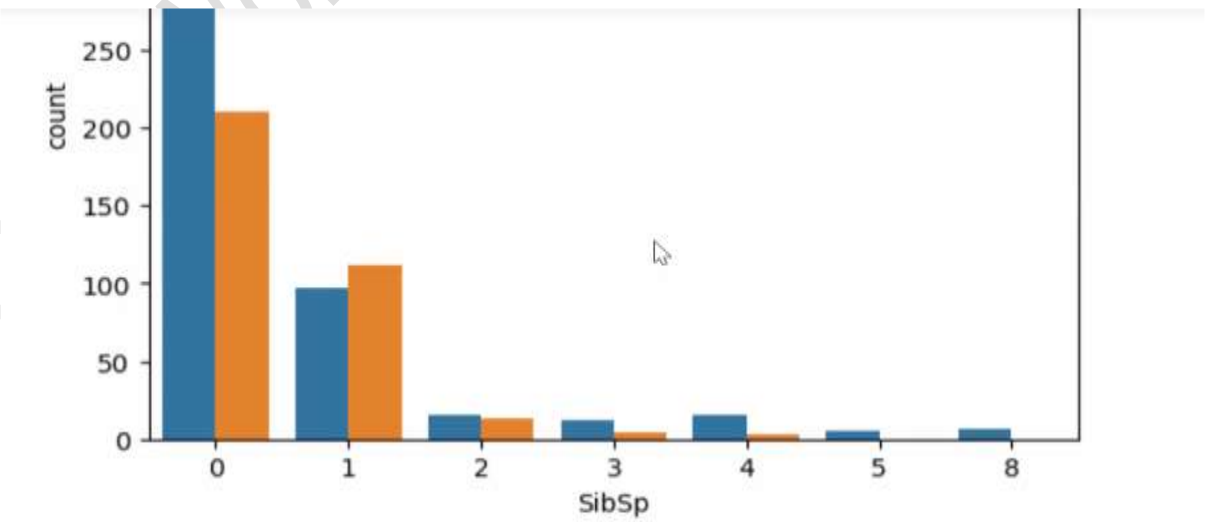
- This line of code uses Seaborn to create a heatmap visualization of missing (null) values in the **dataset**, where each cell in the heatmap indicates whether a value is missing in the corresponding cell of the DataFrame

```
In [21]:  sns.heatmap(  dataset.isnull()  )
Out[21]: <Axes: >
```

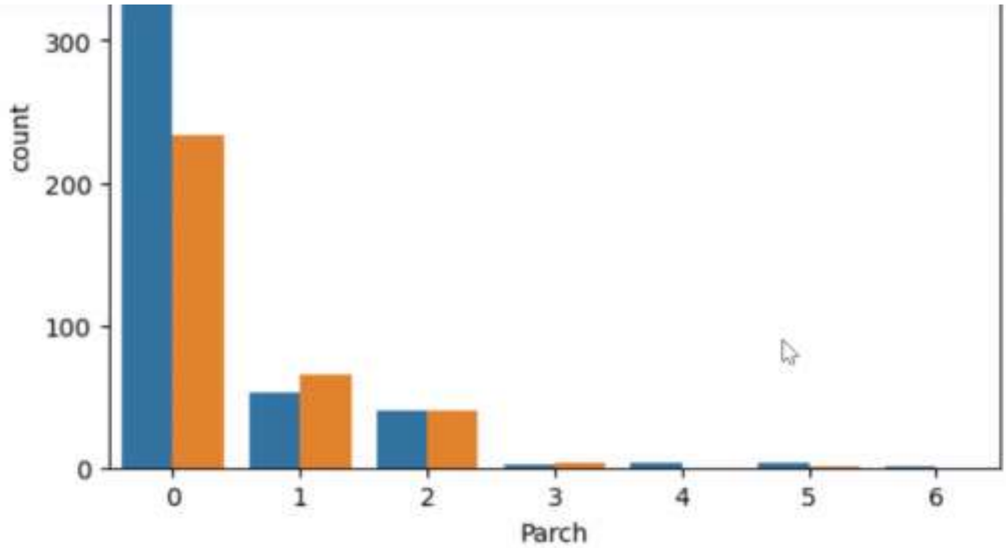


- These lines of code use Seaborn to create count plots showing the distribution of 'SibSp' (number of siblings/spouses) and 'Parch' (number of parents/children) variables, with survival status ('Survived') as the hue

```
In [70]:  1  sns.countplot( x=dataset['SibSp'], hue='Survived', data=dataset)
Out[70]: <Axes: xlabel='SibSp', ylabel='count'>
```



```
In [72]:  1 sns.countplot( x=dataset['Parch'], hue='Survived', data=dataset)
Out[72]: <Axes: xlabel='Parch', ylabel='count'>
```



- After that we see the columns of the dataset

```
In [74]:  1 dataset.columns
Out[74]: Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
                'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
               dtype='object')
```

- Pclass and sex a categorical variables To handle categorical variables like 'Pclass' and 'Sex', we need to create dummy variables

```
In [78]:  1 final_pclass = pandas.get_dummies(dataset['Pclass'] , drop_first=True)
          2

In [79]:  1 final_gender = pandas.get_dummies(dataset['Sex'] , drop_first=True)
          2

In [80]:  1 dataset['Age']  =  dataset[[ 'Pclass' , 'Age']].apply( ageMissing ,  axis=1  )

In [82]:  1 dataset[  'Age'  ]
Out[82]: 0      22.0
         1      38.0
         2      26.0
         3      35.0
         4      35.0
                ...
         886    27.0
         887    19.0
         888    25.0
         889    26.0
```

- After creating a dummy variable Concatenate we all the final variable

```
In [85]:  1 pandas.concat( [ dataset['Age'], final_pclass , final_gender ], axis=1)
Out[85]:
```

	Age	2	3	male
0	22.0	False	True	True
1	38.0	False	False	False
2	26.0	False	True	False
3	35.0	False	False	False
4	35.0	False	True	True
...	...	...	...	...
886	27.0	True	False	True
887	19.0	False	False	False

- After that store this value in X variable

```
In [86]:  1 X = pandas.concat( [ dataset['Age'], final_pclass , final_gender ], axis=1)

In [87]:  1 y = dataset['Survived']

In [ ]:   1 |
```

- Now we have X and y value
- After that we import Logistic Regression from sci-kit-learn, initializes a logistic regression model

```
In [90]:  1 from sklearn.linear_model import LogisticRegression

In [91]:  1 model = LogisticRegression()
```

- convert column names to string type

```
In [96]:  1 X.columns  = X.columns.astype(str)

In [97]:  1 X.columns
Out[97]: Index(['Age', '2', '3', 'male'], dtype='object')
```

- fits the model with input features X and target variable y

```
In [99]:  1 model.fit(X, y)
Out[99]: ▸ LogisticRegression
         LogisticRegression()
```

- After that retrieve coefficients, and predicting target values using the trained model

```
In [101]:  1 model.coef_
Out[101]: array([[-0.03370228, -1.02947963, -2.26171984, -2.4897456 ]])
```

```
In [102]:  1 X
Out[102]:
```

	Age	2	3	male
0	22.0	False	True	True
1	38.0	False	False	False
2	26.0	False	True	False
3	35.0	False	False	False
4	35.0	False	True	True

```
In [105]:  1 model.predict(X)
Out[105]: array([0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
                 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1,
                 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0,
                 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0,
                 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1,
                 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0,
                 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
                 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0,
                 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,
                 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1,
                 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0,
                 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1,
                 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1
```

- We Train a logistic regression model and make predictions
- Once we've trained our model, the next step is to evaluate its performance using a separate test dataset. The process for testing the model remains the same as for training, beginning with reading the test data CSV file.

```
In [106]:  1 dataset_test = pandas.read_csv('titanic_test.csv')
```

```
In [107]:  1 dataset_test
Out[107]:
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E	female	22.0	1	1	3101298	12.2875	NaN	S

```
In [108]:
final_pclass = pandas.get_dummies(dataset_test['Pclass'] , drop_first=True)
final_gender = pandas.get_dummies(dataset_test['Sex'] , drop_first=True)

dataset_test['Age']  =  dataset_test[[ 'Pclass' , 'Age']].apply( ageMissing ,  axis=1  )
X_test = pandas.concat( [ dataset_test['Age'], final_pclass , final_gender ], axis=1)
```

```
In [109]:
X_test
```

Out[109]:

	Age	2	3	male
0	34.5	False	True	True
1	47.0	False	True	False
2	62.0	True	False	True
3	27.0	False	True	True
4	22.0	False	True	False

- Prepare the data for modeling by creating dummy variables for 'Sex' and 'Pclass', handling missing values in 'Age', and concatenating all the final variables. Then, utilize both the training and testing datasets to train the model and obtain predictions

```
In [110]:
X_test.columns  = X_test.columns.astype(str)
```

```
In [111]:
X_test
```

Out[111]:

	Age	2	3	male
0	34.5	False	True	True
1	47.0	False	True	False
2	62.0	True	False	True
3	27.0	False	True	True
4	22.0	False	True	False

```
In [114]:
model.predict(X_test)
```

```
Out[114]: array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0,
```