Report about the titanic data

Link to the Gitlab page :

https://gitlab.com/python7963908/cassie_doguet_iris_dataset_dia/-

Description of the algorithm used for each of the solution

With the titanic dataset we have to predict if the passenger of titanic will survive or not. I decided to apply the logistic regression model to this dataset. Here are the 5 first rows of the dataset and its details:

```
train = pd.read_csv("...")
train.head()
```

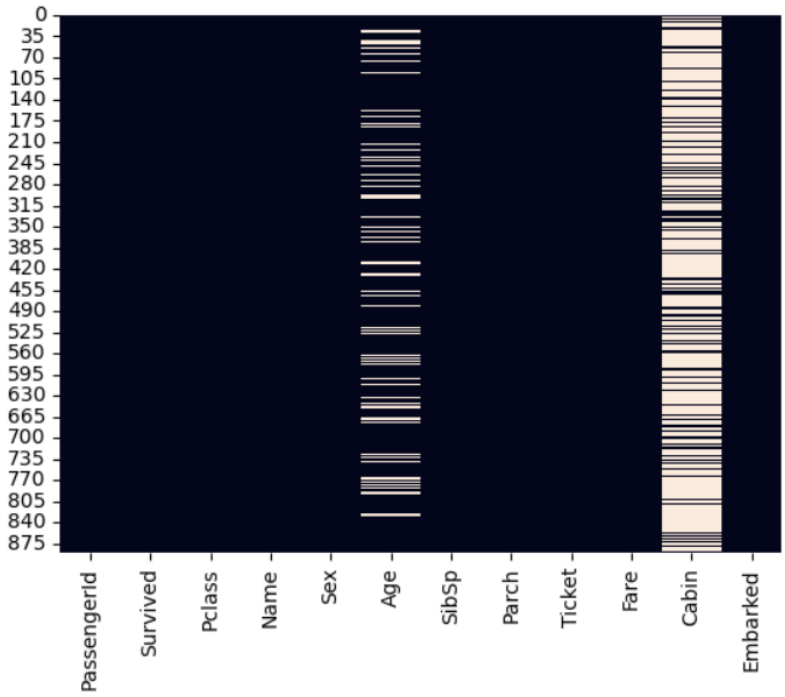
	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```
train.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

Now we have to clean the data, and first plot where do we have a lack of information:

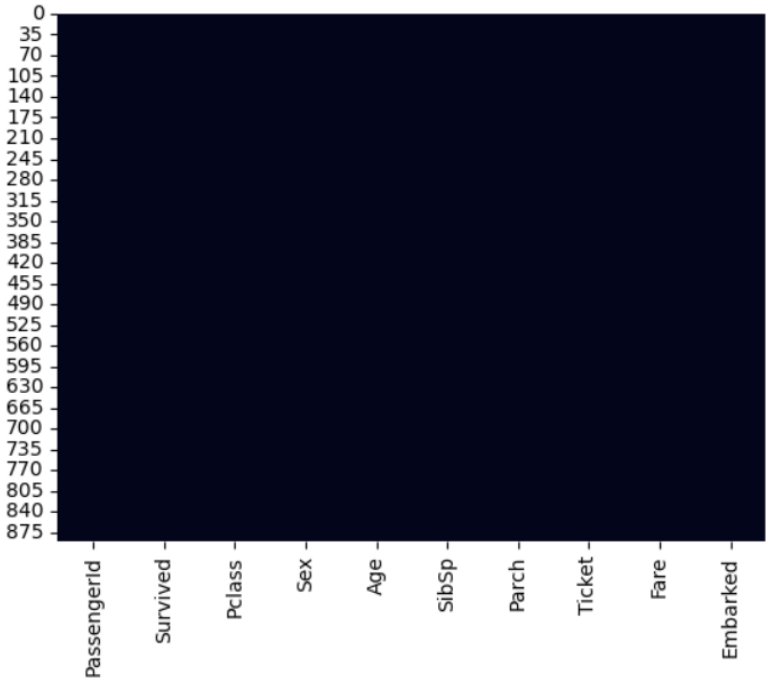
```
sns.heatmap(train.isnull(),cbar=False)
plt.show()
```



There are several things to do to clean the data :

- Encode the sex column so that it's no longer "male" and "female" but "0" and "1"
- Encode the Embarked column
- Encode the ticket column
- Drop the columns Name and Cabin
- Replace the NaN values of Age by the mean value of age

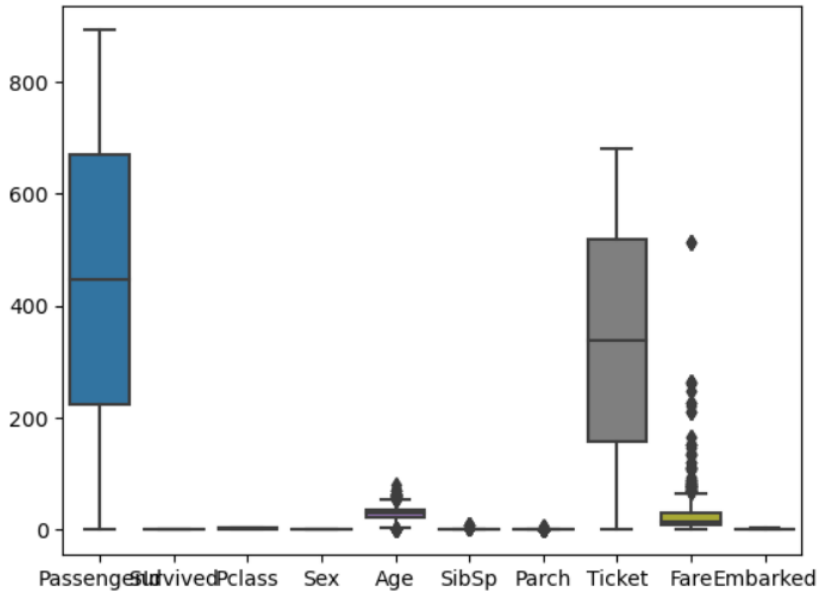
```
le = LabelEncoder()
train['Sex'] = le.fit_transform(train['Sex'])
train['Embarked']=le.fit_transform(train['Embarked'])
train['Ticket']=le.fit_transform(train['Ticket'])
train = train.drop(columns=["Name","Cabin"])
train.fillna(train.mean(),inplace=True)
sns.heatmap(train.isnull(),cbar=False)
plt.show()
```



I chose to drop the name's and cabin's columns. The names are not use full as we have the parch and we do not have enough values in the cabin's column.

We now want to see if we have outliers values.

```
sns.boxplot(data = train)
pyplot.show()
```



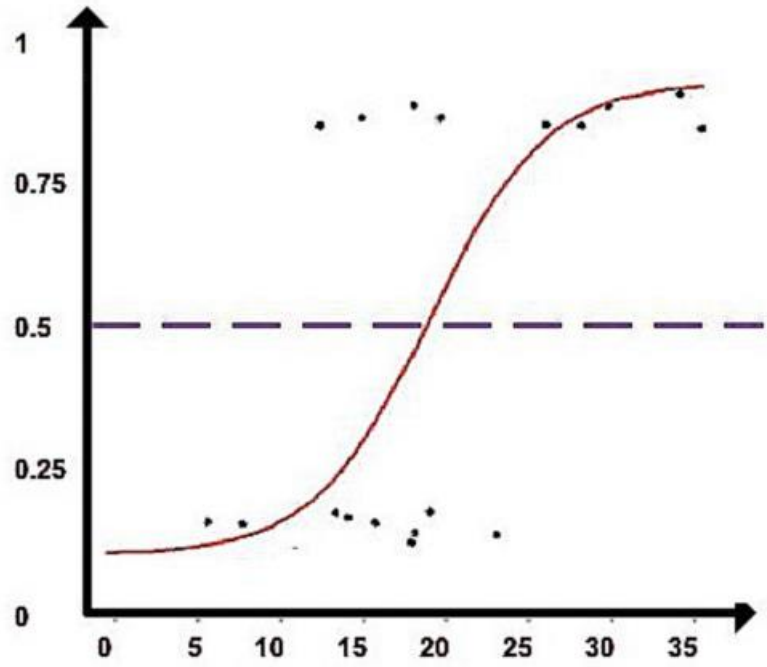
I do the same with the testing dataset.

Then we have to define the training and testing data :

```
x_train = train.drop(columns=["Survived"])
y_train = train['Survived']
#We also have x_test from the cell above
y_test = pd.read_csv("/Users/cassi/OneDrive/Documents/ESILV_A3S2/Data_science_a
y_test = y_test.to_numpy()
y_test_col_2 = y_test[:, 1]
y_test = pd.DataFrame(y_test_col_2)
```

How the model performance was improved

First, we have to see how Logistic regression works. Logistic regression is used to predict the probability based on features. The prediction is always between 0 and 1. It's very suitable for our data set because the closest we are from 1, the more chance we have of surviving. As we can see on the picture below it calculated the probability of the event using the data :



I first tried to improve the model performance by changing the solver hyperparameter. I could then see the accuracy of each model using different solvers :

```
def Solver_parameter(sol) :
    lr = LogisticRegression(C=1, max_iter=10000, solver=sol, random_state=100)

    # Entraîner le modèle sur les données d'entraînement
    lr.fit(x_train, y_train)

    # Faire des prédictions sur les données de test
    y_pred = lr.predict(x_test)

    # Calculer la précision du modèle
    accuracy = accuracy_score(y_test, y_pred)
    print("Precision with",i,": {:.2f}%".format(accuracy * 100))
    return accuracy*100

sols = ['newton-cg','lbfgs','liblinear','sag','saga','newton-cholesky']
liste = []
for i in sols :
    liste.append(Solver_parameter(i))

liste = np.array(liste)

print("\nThe best solver for our model is : ",sols[np.argmax(liste)]," with an accuracy of : ",np.max(liste), "%")
```

```
Precision with newton-cg : 92.82%
Precision with lbfgs : 92.82%
Precision with liblinear : 94.02%
Precision with sag : 91.87%
Precision with saga : 88.28%
Precision with newton-cholesky : 92.82%

The best solver for our model is :  liblinear  with an accuracy of :  94.01913875598086 %
```

Then I decided to use the Grid Search exactly like I did with the iris data set but for the hyperparameters of a logistical regression model.

```
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['lbfgs', 'newton-cg', 'liblinear', 'sag', 'saga']
}

logreg = LogisticRegression(max_iter=10000)

grid_search = GridSearchCV(estimator=logreg, param_grid=param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)
best_params = grid_search.best_params_

final_model = LogisticRegression(**best_params)
final_model.fit(x_train, y_train)

y_pred = final_model.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
Accuracy: 0.9401913875598086
```

I found an accuracy almost like the one where I was only changing the solver.