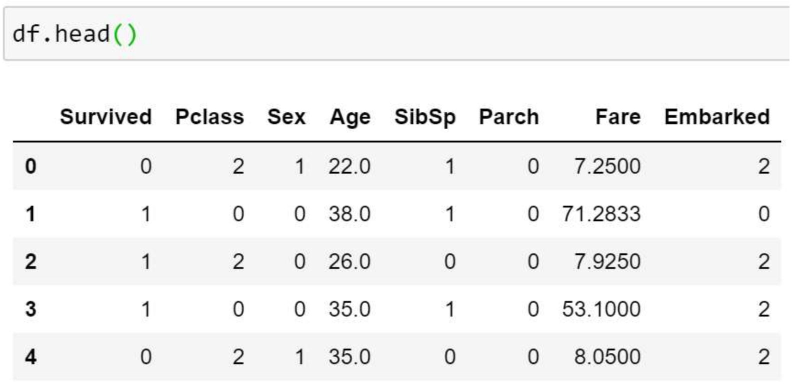
PROJECT 5: DEEP LEARNING

Objective
The goal of this project is similar to that of Project 3: analyse the Titanic dataset to predict whether a given passenger survived. This time, we will use deep learning models.

Description of data
For more details on the dataset and the process of cleaning the data, see Project 1. Here is a summary. The data initially had 891 entries and 13 features. The target feature is Survived. In cleaning the data, we did the following:

1. Dropped PassengerId, Name, Ticket, Fare, Cabin
2. Encoded Pclass, Embarked, Sex as ordinal numeric variables, masking missing values of Embarked
3. Imputed to missing Age values the mean according to Sex



```
df.head()
```

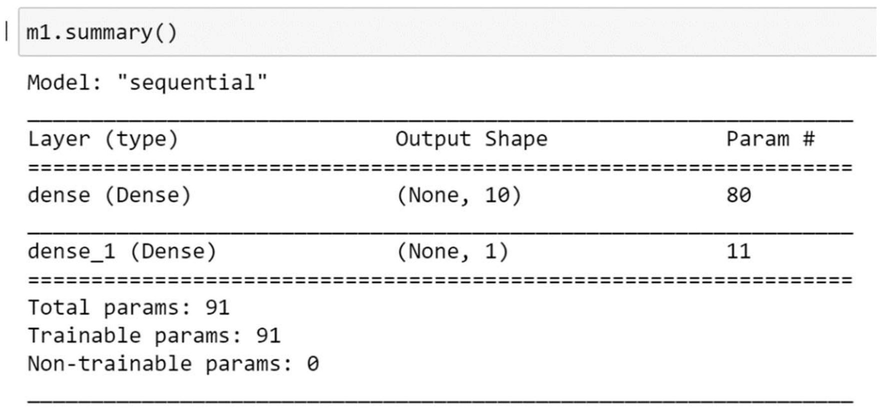
	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	0	2	1	22.0	1	0	7.2500	2
1	1	0	0	38.0	1	0	71.2833	0
2	1	2	0	26.0	0	0	7.9250	2
3	1	0	0	35.0	1	0	53.1000	2
4	0	2	1	35.0	0	0	8.0500	2

We use a train-test split with 20% test size and scale the feature using the standard scaler.

Deep learning models
For this project, we use neural networks with 1 neuron in their output layer. A sigmoid activation function will be applied to the output neuron, and its output will be the probability that the passenger under consideration survived. All classifiers will be trained on the same training set with a batch size of 100 for 10 epochs.

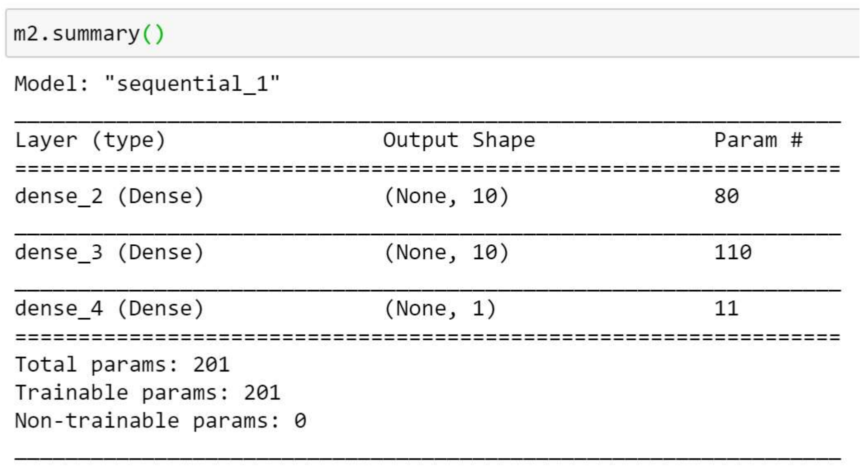
We consider three deep learning models:

1. A model with one hidden layer containing 10 neurons.
2. A model with two hidden layers, each containing 10 neurons.
3. A model with one hidden layer containing 3 neurons.



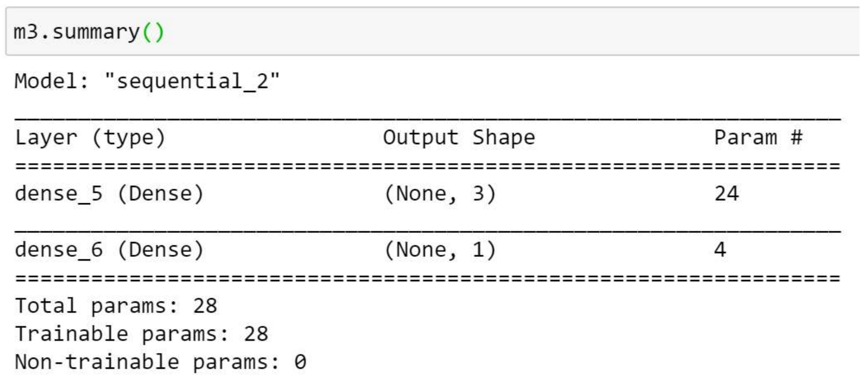
```
m1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                80
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 11
=================================================================
Total params: 91
Trainable params: 91
Non-trainable params: 0
_________________________________________________________________
```



```
m2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 10)                80
_________________________________________________________________
dense_3 (Dense)              (None, 10)                110
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 201
Trainable params: 201
Non-trainable params: 0
_________________________________________________________________
```

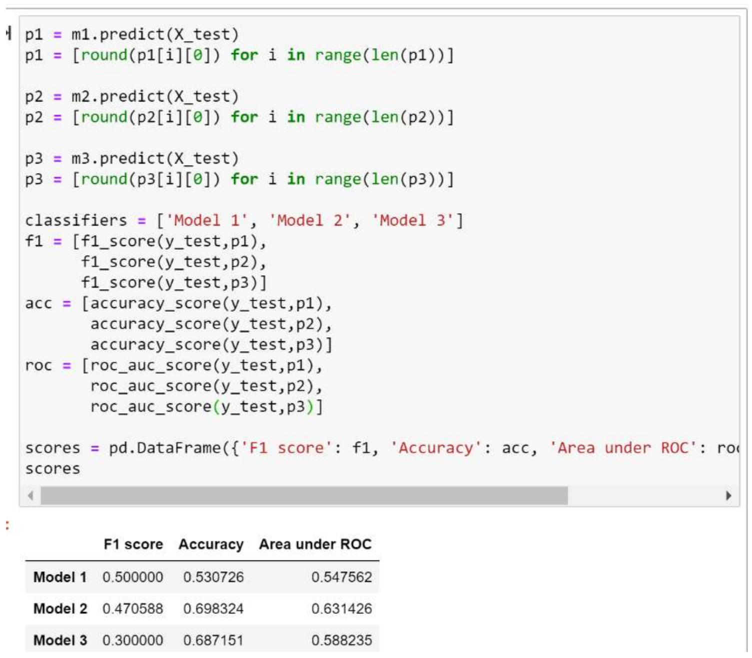


```
m3.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 3)                 24
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 4
=================================================================
Total params: 28
Trainable params: 28
Non-trainable params: 0
```

Evaluation

We compare the f1_scores, accuracy, and area under ROC curve for each model



```
p1 = m1.predict(X_test)
p1 = [round(p1[i][0]) for i in range(len(p1))]

p2 = m2.predict(X_test)
p2 = [round(p2[i][0]) for i in range(len(p2))]

p3 = m3.predict(X_test)
p3 = [round(p3[i][0]) for i in range(len(p3))]

classifiers = ['Model 1', 'Model 2', 'Model 3']
f1 = [f1_score(y_test,p1),
      f1_score(y_test,p2),
      f1_score(y_test,p3)]
acc = [accuracy_score(y_test,p1),
       accuracy_score(y_test,p2),
       accuracy_score(y_test,p3)]
roc = [roc_auc_score(y_test,p1),
       roc_auc_score(y_test,p2),
       roc_auc_score(y_test,p3)]

scores = pd.DataFrame({'F1 score': f1, 'Accuracy': acc, 'Area under ROC': ro
scores
```

	F1 score	Accuracy	Area under ROC
Model 1	0.500000	0.530726	0.547562
Model 2	0.470588	0.698324	0.631426
Model 3	0.300000	0.687151	0.588235

All three models did rather poorly on the testing set, compared to the decision trees. A natural question then arises as to whether the models are underfit or overfit. One way to answer this question is to evaluate the models on the training set: if the models do just as poorly on the training set, then they are likely underfit. Another way to answer this question is to observe that model 2 did best on accuracy and area under ROC. Since model 2 is the most complex, this suggests underfitting. To improve the performance of the models, we might either add more complexity, or increase the number of epochs.