Titanic Disaster

In this work we want to build **a predictive** model that will help to understand who is more likely to survive to a titanical Shipwreck ,using passenger data (ie name, age, gender, socio-economic class, etc).

We are going to use **Deep Learning Models**, as baseline we will test a simple **Logistic Regression** to compare with a **Vanilla Neural Network**, **a Deep Neural Network** and a second **Optimized Deep Neural Network**.

You can find Data Source at https://www.kaggle.com/c/titanic/data

The data has been split into two groups:

- training set (train.csv)
- test set (test.csv)

Data Overview

Both the dataset with the following features:

PassengerId: Unique ID of the passenger

Pclass: Ticket class

Name: Full name of the passenger with salutation

Sex: Gender

Age: Age in years

SibSp: Number of siblings / spouses aboard the Titanic

Parch: Number of parents / children aboard the Titanic

Ticket: Ticketnumber

Fare: Passenger fare

Cabin: Cabinnumber

Embarked: Port of Embarkationtrain.csv has the target variable named Survived

Test dataset has one less column because there’s no target variable.

```
labelled_df.shape, unlabelled_df.shape
```

```
((891, 12), (418, 11))
```

Then we merge the 2 dataset and provide a sample

```
df.sample(5)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
820	821	1.0	1	Hays, Mrs. Charles Melville (Clara Jennings Gr...	female	52.0	1	1	12749	93.5000	B69	S
205	206	0.0	3	Strom, Miss. Telma Matilda	female	2.0	0	1	347054	10.4625	G6	S
1274	1275	NaN	3	McNamee, Mrs. Neal (Eileen O'Leary)	female	19.0	1	0	376566	16.1000	NaN	S
123	124	1.0	2	Webber, Miss. Susan	female	32.5	0	0	27267	13.0000	E101	S
687	688	0.0	3	Dakic, Mr. Branko	male	19.0	0	0	349228	10.1708	NaN	S

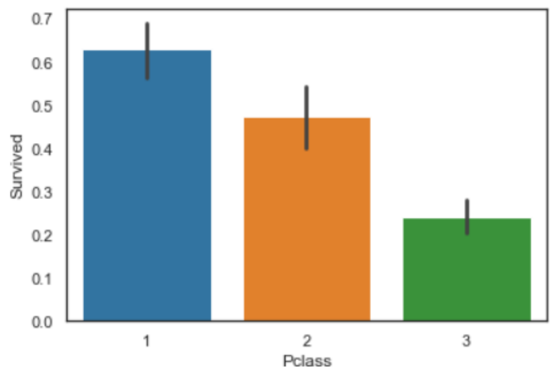
Exploratory Data Analysis

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  1309 non-null   int64
 1   Survived     891 non-null    float64
 2   Pclass       1309 non-null   int64
 3   Name         1309 non-null   object
 4   Sex          1309 non-null   object
 5   Age          1046 non-null   float64
 6   SibSp        1309 non-null   int64
 7   Parch        1309 non-null   int64
 8   Ticket       1309 non-null   object
 9   Fare         1308 non-null   float64
 10  Cabin        295 non-null    object
 11  Embarked     1307 non-null   object
dtypes: float64(3), int64(4), object(5)
memory usage: 122.8+ KB
```

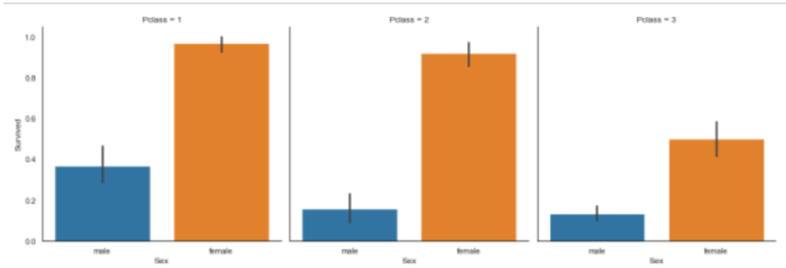
We have missing values for Age, Cabin, Fare and Embarked features, since the observations available are few, we want to fill this missing values. We have the following insights…

1) **PassengerID**: removed, no value added to model
2) **Pclass**: no missing values. First class passengers are more likely to survive followed by 2nd and 3rd

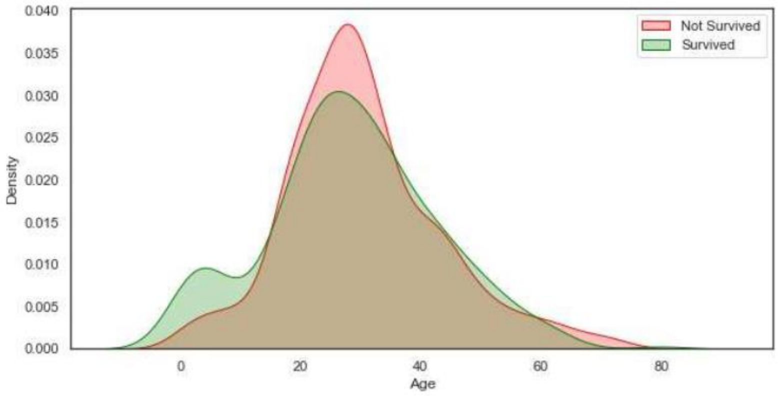


3) **Name**: no correlations with Survived

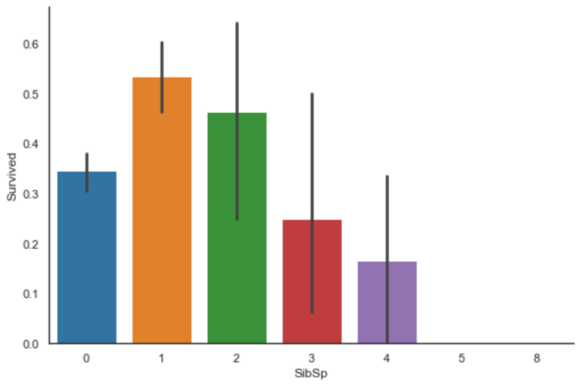
4) **Sex:** females are much more likely to survive than males in all classes.



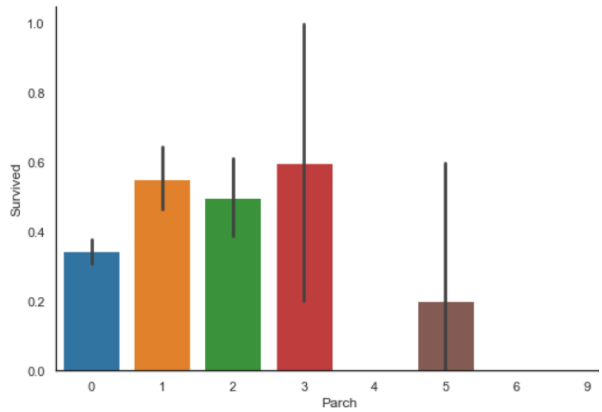
5) **Age**: We have 263 missing values for Age. We will fill the blank with Average values accordingly Ticket Class



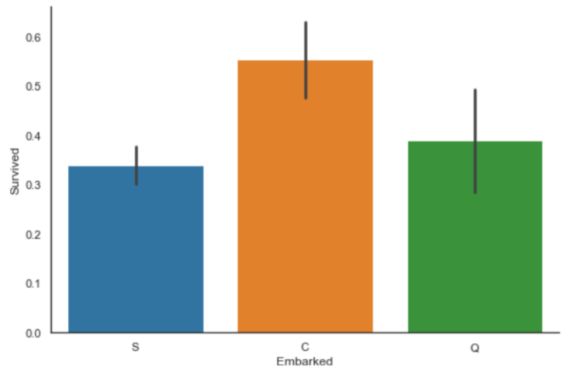
6) SibSp: Passengers with few Sibs are more likely to survive as alone passengers do as well.



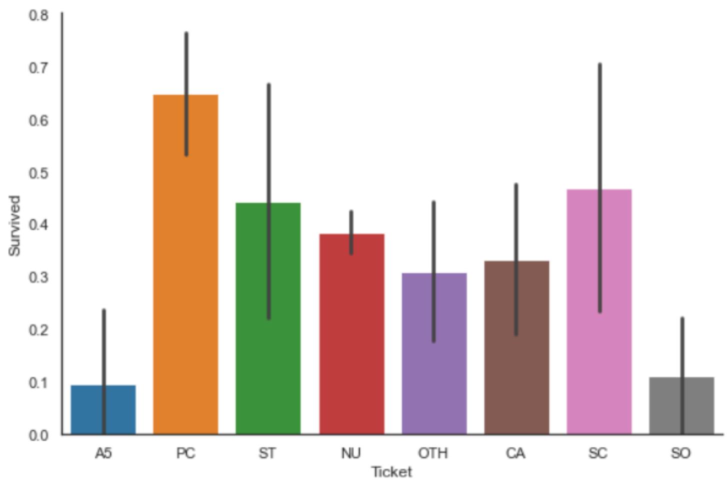
7) Parch: Passengers travelling with up to 3 children have slightly higher possibility to survive



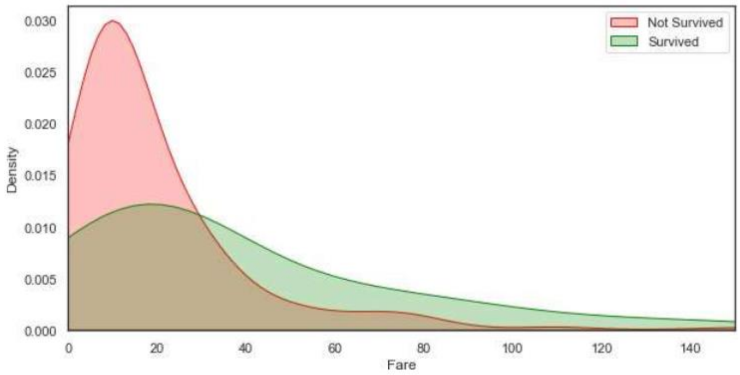
8) **Embarked**: Passengers from Cherbourg have highest probability of surviving.



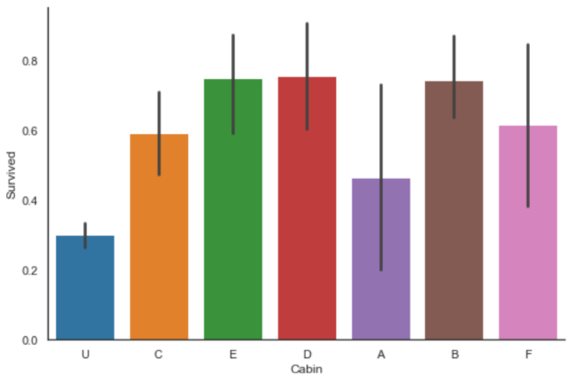
9) **Tickets** : Passengers holding PC tickets have highest probability of survival.



10) **Fare**: Only 1 Missing value replaced by mean value.



11) **Cabin**: It seems there's no correlation with Survived Feature.



Feature Engineering

First, we will drop ‘**PassengerId’** and **'Name'** features. Then, we will encode Binary and Categorical features.

```
from sklearn.preprocessing import LabelBinarizer, MinMaxScaler

lb = LabelBinarizer()
for col in binary_cols:
    df[col] = lb.fit_transform(df[col])

df = pd.get_dummies(data = df, columns=cat_cols, drop_first=True)
```

Now, we can create Train and Test Split after re-dividing our DataSet (remember we merged Label and Unlabeled csv)

```
# Create train-test splits
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, ran
dom_state=0)
```

Then a preprocess phase to properly scale our numerical columns

```
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_train_scaled = sc.fit_transform(X_train)
X_test_scaled = sc.transform(X_test)
```

Deep Learning Models

As we said, we will train multiple neural network models and compare performances using accuracy.

Let’s start with a baseline model to compare with the others.

Logistic Regression

```
from sklearn.model_selection import GridSearchCV, KFold
from sklearn.linear_model import LogisticRegressionCV

kf = KFold(n_splits = 4, shuffle = True)

params = {'Cs': [2, 5, 10],
          'penalty': ['l1', 'l2'],
          'solver': ['newton-cg', 'lbfgs', 'liblinear']}

grid = GridSearchCV(estimator = LogisticRegressionCV(),
                    param_grid = params,
                    scoring = 'accuracy',
                    cv = kf,
                    n_jobs = -1)
grid.fit(X_train_scaled, y_train)
```

We got a score of **0.832**. Not bad, simple and clean dataset. It was an expected result.

Let’s see if we can improve it.

Basic Neural Network v1.00 (No Hidden Layer so far)

So, just a Sequential Model with 1 Dense Layer, ReLu as activation function, with a Loss Function = Binary CrossEntropy and An Accuracy Score. Run for 20 Epochs and batch size = 32

```
from keras.models import Sequential
from keras.layers import Dense

nn1 = Sequential()
nn1.add(Dense(units=25, input_shape = (24,), activation='relu'))
nn1.add(Dense(units=1, activation='sigmoid'))
nn1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 25)                625
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 26
=================================================================
Total params: 651
Trainable params: 651
Non-trainable params: 0
_________________________________________________________________
```

We get:

```
Epoch 20/20
23/23 [==============================] - 0s 1ms/step - loss: 0.3914 - accurac
y: 0.8385 - val_loss: 0.3816 - val_accuracy: 0.8492

<tensorflow.python.keras.callbacks.History at 0x1de9ccf4148>
```

A slightly increase in accuracy.

Let's move further with a

Deep Neural Network v1.01(Basic + 1 Hidden Layer)

Same hyperparameters:

```
nn2 = Sequential()
nn2.add(Dense(units=25, input_shape = (24,), activation='relu'))
nn2.add(Dense(units=50, activation='relu'))
nn2.add(Dense(units=1, activation='sigmoid'))
nn2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_2 (Dense)              (None, 25)                625
_________________________________________________________________
dense_3 (Dense)              (None, 50)                1300
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 51
=================================================================
Total params: 1,976
Trainable params: 1,976
Non-trainable params: 0
```

We get after 20 Epochs:

```
Epoch 20/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3613 - accurac
y: 0.8455 - val_loss: 0.3733 - val_accuracy: 0.8380

<tensorflow.python.keras.callbacks.History at 0x1de9ef34ac8>
```

Another slight improvement.

Let's add more layers (1 more layer), same Hyperparameters:

```
nn2 = Sequential()
nn2.add(Dense(units=25, input_shape = (24,), activation='relu'))
nn2.add(Dense(units=50, activation='relu'))
nn2.add(Dense(units=50, activation='relu'))
nn2.add(Dense(units=1, activation='sigmoid'))
nn2.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = 'ac
curacy')
nn2_steps = nn2.fit(X_train_scaled, y_train, batch_size = 32, epochs = 20, val
idation_data = (X_test_scaled, y_test))
```

```
Epoch 20/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3314 - accurac
y: 0.8624 - val_loss: 0.3712 - val_accuracy: 0.8659
```

We keep improving accuracy!

Now we want exaggerate end implement another Deep Neural Net with Adam Optimizer and different Hyperparameters :

Deep Neural Network v1.2

```
nn3 = Sequential()
nn3.add(Dense(units=25, input_shape = (24,), activation='relu'))
nn3.add(Dense(units=50, activation='relu'))
nn3.add(Dense(units=50, activation='relu'))
nn3.add(Dense(units=1, activation='sigmoid'))
nn3.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = 'accur
acy')
nn3_steps = nn3.fit(X_train_scaled, y_train, batch_size = 1, epochs = 50, vali
dation_data = (X_test_scaled, y_test), shuffle=True)
```

Batch size =1 and run for 50 Epochs, we got :

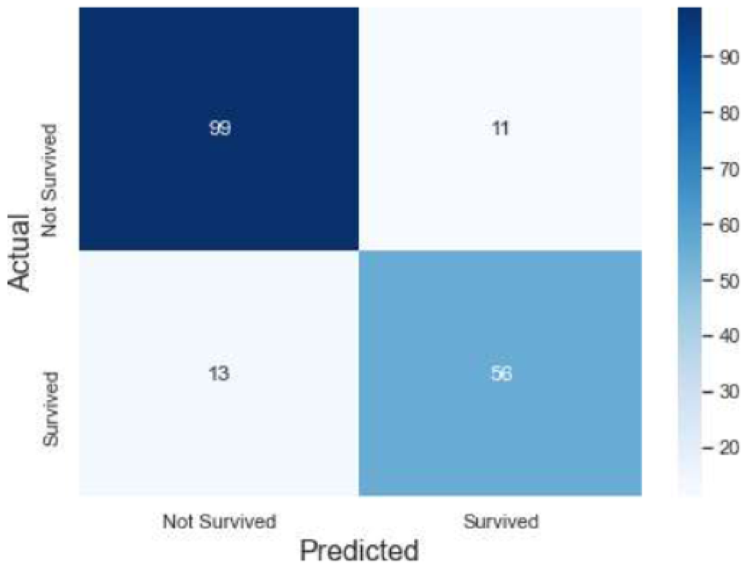
```
Epoch 50/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2246 - accur
acy: 0.9017 - val_loss: 0.6986 - val_accuracy: 0.8324
```

A very good accuracy! Maybe too good…indeed, plotting Training Accuracy Vs Test Accuracy we get:

We have Overfitted!!

CONCLUSION

Best Fit Model : We can't use **DNN v1.2** because it overfits, so only our best guess is the **DNN v1.01** with an accuracy of 0.8624 (vs Logistic Regression of 0.832) and Confusion Matrix:



```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.88      0.90      0.89       110
         1.0       0.84      0.81      0.82        69

    accuracy                           0.87       179
   macro avg       0.86      0.86      0.86       179
weighted avg       0.87      0.87      0.87       179
```

Model DNN v1.01 performed well also with the Unlabeled Dataset with an accuracy of 0.759 .

Next steps: In this work we didn't check for correlation between features and multicollinearity and probably we should have tuned hyperparameters in a better way, we hope that removing correlation and multilinearity we would be able to improve the model and get a better. We also remark the fact that the Data Set is very small, compared to the typical use of DNN.