Section 1: Data set description & Objective

1.1: Introduction of the dataset

On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew. While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.

Source: ***[https://www.kaggle.com/c/titanic/data (https://www.kaggle.com/c/titanic/data)](https://www.kaggle.com/c/titanic/data)***

1.2: Dataset features

We are given with two separate dataset here.

- train.csv: It is a **labelled dataset** which we need to use to train the models.
- test.csv: It is not labelled and used only for predictions.

Both the dataset have 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: Ticket number
- Fare: Passenger fare
- Cabin: Cabin number
- Embarked: Port of Embarkation

train.csv has the target variable named *Survived*.

1.3: Objective

The objective is to make survival prediction on test.csv. Then, the predicted values can be submitted to Kaggle which will calculate the model accuracy.

1.4: Glimpse of the dataset

Not to confuse with train-test splitting, we will import train.csv as labelled and test.csv as unlabelled dataset. We will merge both labelled and unlabelled dataset to run Feature engineering on the whole dataset. Before model training, we will break them down again.

In [1]:
```
# Importing essentials
import numpy as np, pandas as pd, matplotlib.pyplot as plt, seaborn as sns
```

In [2]:
```
# Setting the stage
sns.set_context('notebook')
sns.set_style('white')
```

In [3]:
```
# Supress warning
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
# Importing dataset
labelled_df = pd.read_csv('train.csv')
unlabelled_df = pd.read_csv('test.csv')
```

In [5]:
```python
# Checking the shapes
labelled_df.shape, unlabelled_df.shape
```

Out[5]:
```
((891, 12), (418, 11))
```

Unlabelled dataset has one less columns as it doesn't have target variable.

In [6]:
```python
# Checking if Labelled dataset all survival values
labelled_df.Survived.isna().sum()
```

Out[6]:
```
0
```

Good that labelled dataset has no missing target variable values

In [7]:
```python
# Merging the datasets
df = pd.concat([labelled_df, unlabelled_df], ignore_index=True)
df.shape
```

Out[7]:
```
(1309, 12)
```

Now, the mergred dataset should have 418 missing values in Survived columns.

In [8]:
```python
df.Survived.isna().sum()
```

Out[8]:
```
418
```

Looking into some samples from the dataset

In [9]:
```
df.sample(10)
```

Out[9]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
515	516	0.0	1	Walker, Mr. William Anderson	male	47.0	0	0	36967	34.0208
443	444	1.0	2	Reynaldo, Ms. Encarnacion	female	28.0	0	0	230434	13.0000
1223	1224	NaN	3	Thomas, Mr. Tannous	male	NaN	0	0	2684	7.2250
136	137	1.0	1	Newsom, Miss. Helen Monypeny	female	19.0	0	2	11752	26.2833
641	642	1.0	1	Sagesser, Mlle. Emma	female	24.0	0	0	PC 17477	69.3000
1046	1047	NaN	3	Duquemin, Mr. Joseph	male	24.0	0	0	S.O./P.P. 752	7.5500
1243	1244	NaN	2	Dibden, Mr. William	male	18.0	0	0	S.O.C. 14879	73.5000
45	46	0.0	3	Rogers, Mr. William John	male	NaN	0	0	S.C./A.4. 23567	8.0500
570	571	1.0	2	Harris, Mr. George	male	62.0	0	0	S.W./PP 752	10.5000
190	191	1.0	2	Pinsky, Mrs. (Rosa)	female	32.0	0	0	234604	13.0000

Section 2: Data Analysis

2.1: Shape and size

We have missing values for Age, Cabin, Fare and Embaked features. As we have only 891 labelled observations, we cannot afford to remove missing values. We need to find ways to deal with them.

In [10]:
```
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 will now go through each of the features except PassengerID to perform feature engineering. We will remove PassengerID as it doesn't add any value to our analysis or modelling.

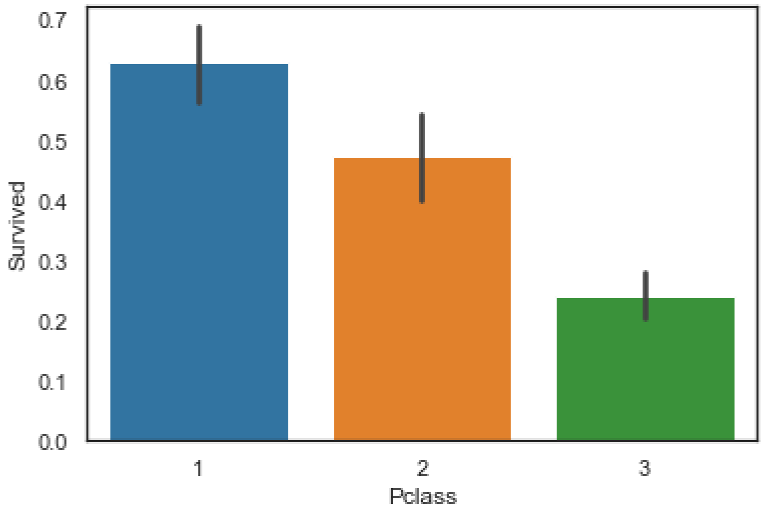
2.2: Pclass

We have no missing data in Pclass. First class passengers are more likely to survive followed by second and third class.

In [11]:
```
df.Pclass.value_counts()
```

Out[11]:
```
3    709
1    323
2    277
Name: Pclass, dtype: int64
```

```
In [12]: # Bar plot to check survival probablities of each ticket class
ax = sns.barplot(x='Pclass', y='Survived', data=df)
```



2.3: Name

Name needn't to have any correlation with survival. However, we will extract salutations from names which will help us to determine missing ages.

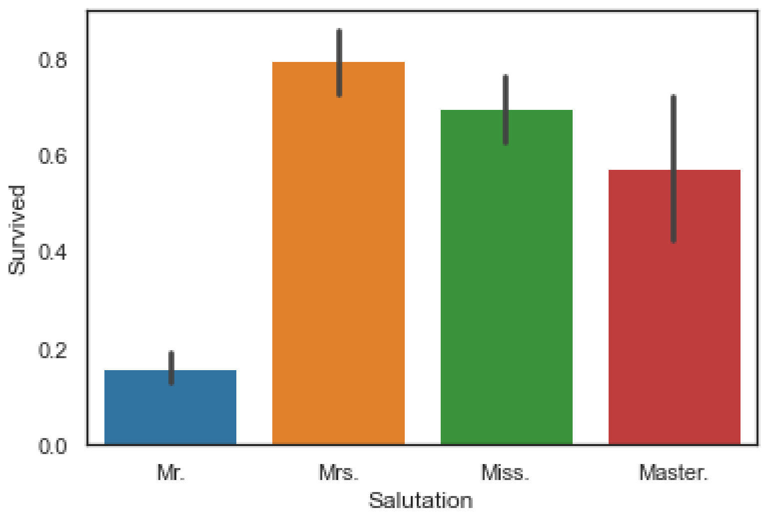
```
In [13]: df['Salutation'] = df['Name'].map(lambda name: name.split(',')[1].split(' ')[1
])
df.groupby(['Salutation', 'Sex']).size()
```

```
Out[13]: Salutation  Sex
Capt.       male        1
Col.        male        4
Don.        male        1
Dona.       female      1
Dr.         female      1
            male        7
Jonkheer.   male        1
Lady.       female      1
Major.      male        2
Master.     male       61
Miss.       female    260
Mlle.       female      2
Mme.        female      1
Mr.         male      757
Mrs.        female    197
Ms.         female      2
Rev.        male        8
Sir.        male        1
the         female      1
dtype: int64
```

In [14]:
```
# Replace salutations for unmarried women with Miss
df['Salutation'].replace('Mlle.', 'Miss.', inplace=True)
# Replace salutations for married women with Mrs
df['Salutation'].replace(['Dona.', 'Lady.', 'Ms.', 'Mme.', 'the'], 'Mrs.', inp
lace=True)
# Replace female doctor with Mrs
df.loc[(df.Salutation == 'Dr.') & (df.Sex == 'female'), 'Salutation'] = 'Mrs.'
# Replace other salutations with Mr. as they are men
salts = df['Salutation'].value_counts()
male_salts = list(salts[salts < 10].index)
df['Salutation'].replace(male_salts, 'Mr.', inplace=True)
# Final count
df['Salutation'].value_counts()
```

Out[14]:
```
Mr.        782
Miss.      262
Mrs.       204
Master.     61
Name: Salutation, dtype: int64
```

In [15]:
```
# Bar plot to check survival probablities of each salutation
ax = sns.barplot(x='Salutation', y='Survived', data=df)
```

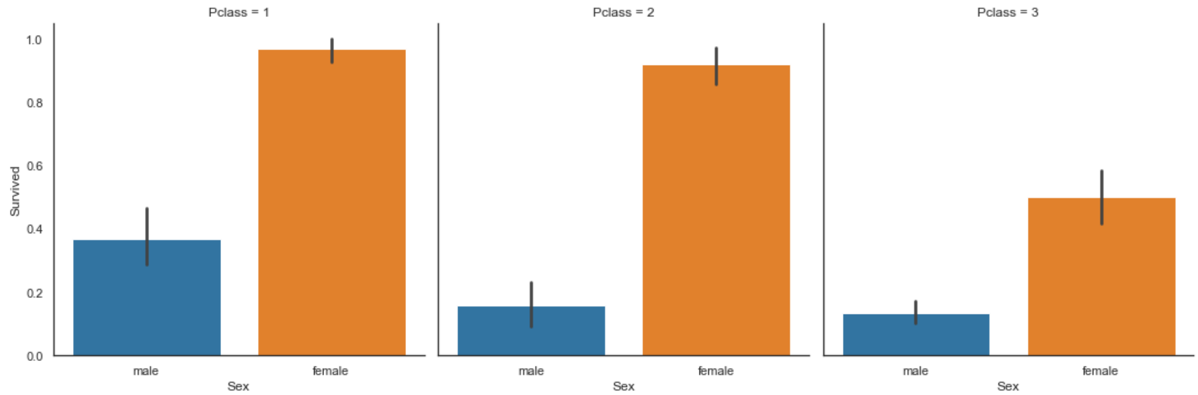


Women and kids have very high probablity of surival than male. We will further check Sex and Age columns.

2.4: Sex

As we have seen above, female are much more likely to survive than male in all classes. However, first class male have more survival rate than others.

In [16]:
```
ax = sns.catplot(x='Sex', y='Survived', col='Pclass', kind='bar', data=df)
```



2.5: Age

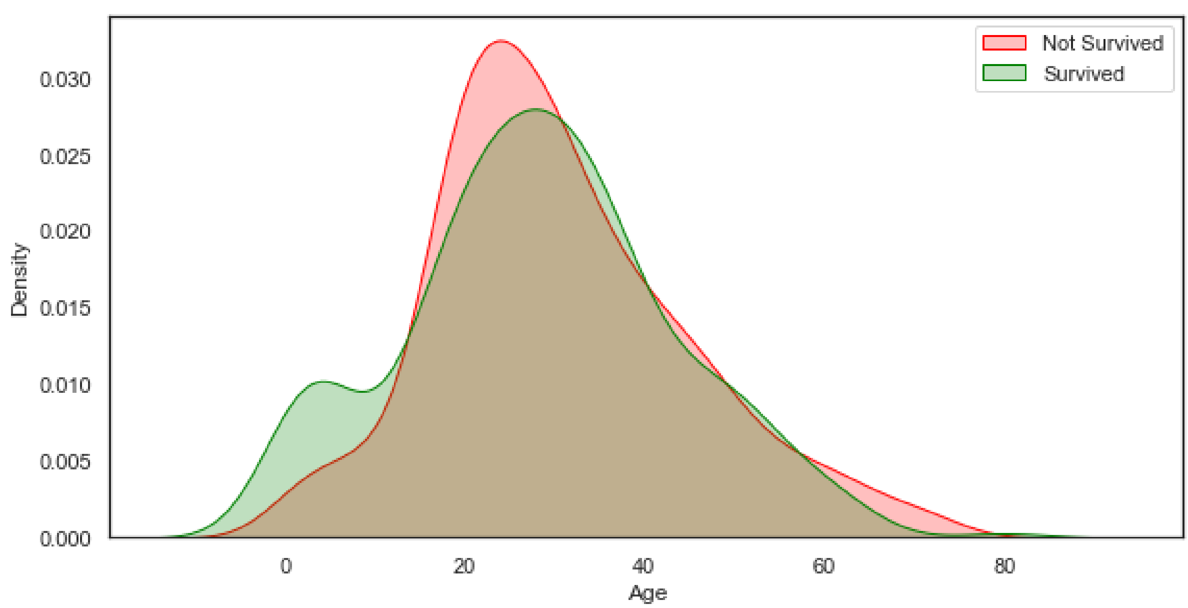
In [17]:
```
df.Age.isna().sum()
```

Out[17]:
```
263
```

We need to fill 263 missing values for Ages. Before that, we will see how Age is correlated with Survival probablity, just for available Ages.

In [18]:
```
fig, ax = plt.subplots(figsize=(10, 5))
ax = sns.kdeplot(data = df.loc[(df.Age.notnull()) & (df.Survived == 0), 'Age'
], shade = True, color = 'red')
ax = sns.kdeplot(data = df.loc[(df.Age.notnull()) & (df.Survived == 1), 'Age'
], shade = True, color = 'green')
ax.legend(['Not Survived', 'Survived'])
```

Out[18]:
```
<matplotlib.legend.Legend at 0x1de9634c388>
```



The hump between 0 to 10 shows young children are more likely to survive. Also, people aged over 70 have almost no chance to survive.

Now, let fill the missing ages by taking the mean of ages from the group of passengers with same Ticket Class, Sex, Embarked and Salutation.

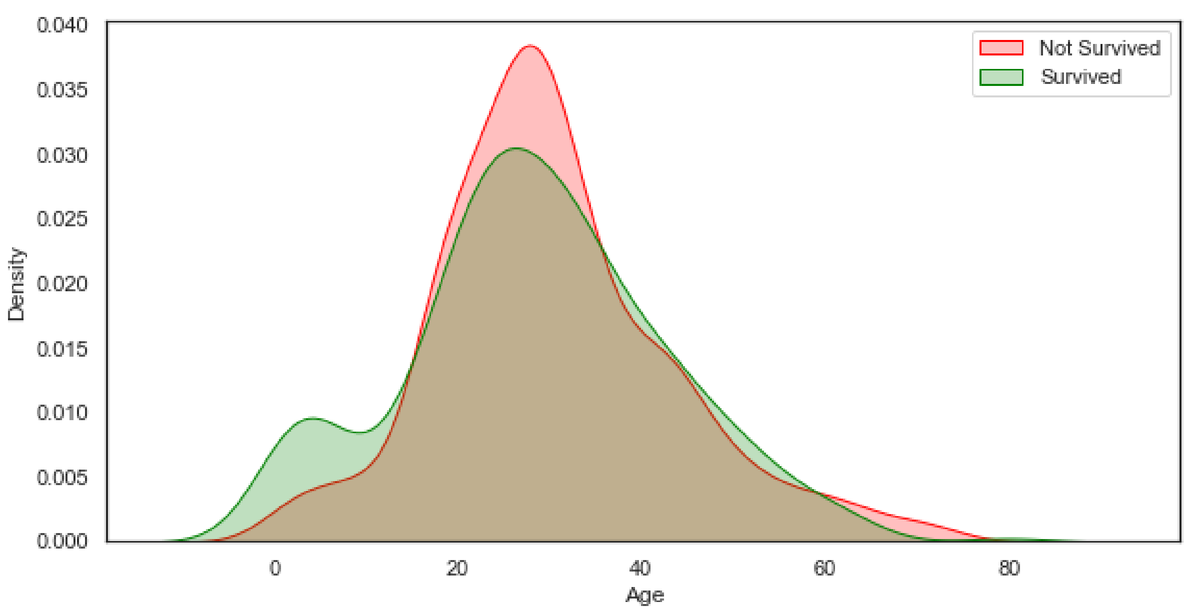
In [19]:
```
df_missing_age = df.loc[df.Age.isna()]

for i, ovn_missing_age in df_missing_age.iterrows():
    pclass = ovn_missing_age.Pclass
    sex = ovn_missing_age.Sex
    embkd = ovn_missing_age.Embarked
    salt = ovn_missing_age.Salutation
    mean_age = df.loc[(df.Pclass == pclass) &
                      (df.Sex == sex) &
                      (df.Embarked == embkd) &
                      (df.Salutation == salt) &
                      (df.Age.notna()), 'Age'].mean()
    df.loc[i, 'Age'] = mean_age
```

Survival density will be impacted after Age approximation, but not hugely. Plotting the graph again to prove.

In [20]:
```
fig, ax = plt.subplots(figsize=(10, 5))
ax = sns.kdeplot(data = df.loc[df.Survived == 0, 'Age'], shade = True, color =
'red')
ax = sns.kdeplot(data = df.loc[df.Survived == 1, 'Age'], shade = True, color =
'green')
ax.legend(['Not Survived', 'Survived'])
```

Out[20]: <matplotlib.legend.Legend at 0x1de963caac8>



2.6: SibSp

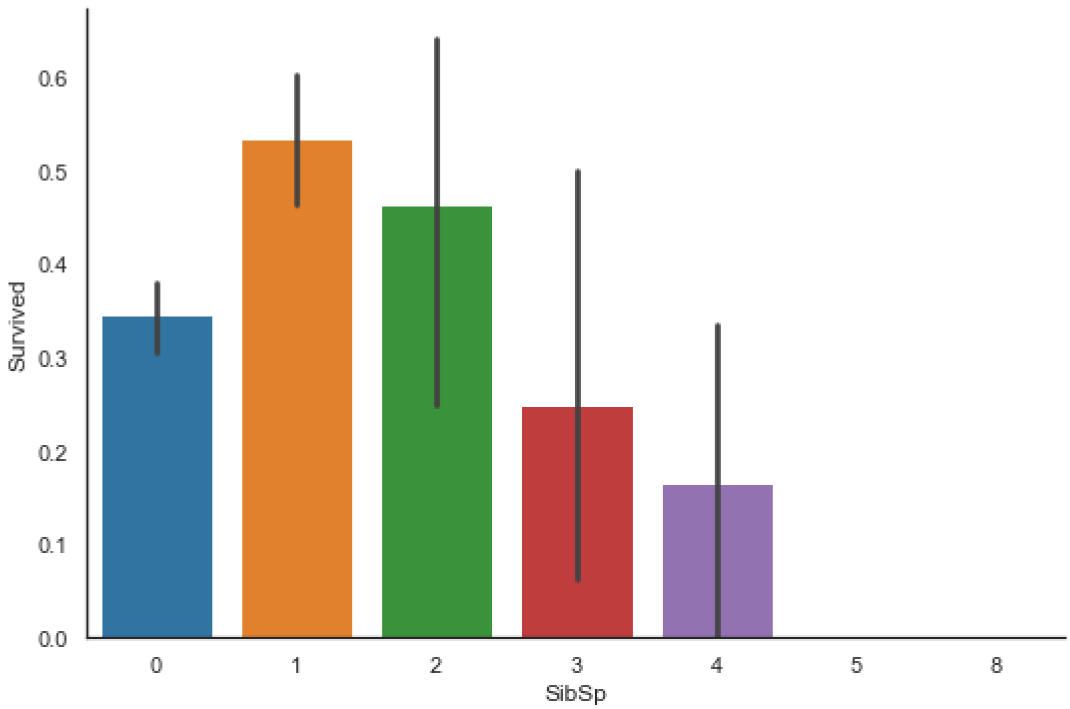
In [21]:
```
df.SibSp.value_counts()
```

Out[21]:
```
0    891
1    319
2     42
4     22
3     20
8      9
5      6
Name: SibSp, dtype: int64
```

Passengers travelling with small family of one or two members are more likely to survive. Passengers travelling alone also have higher probablity to survive than large families.

In [22]:
```
sns.catplot(x='SibSp', y='Survived', kind='bar', data=df, aspect=1.5)
```

Out[22]: <seaborn.axisgrid.FacetGrid at 0x1de963f9dc8>



2.7: Parch

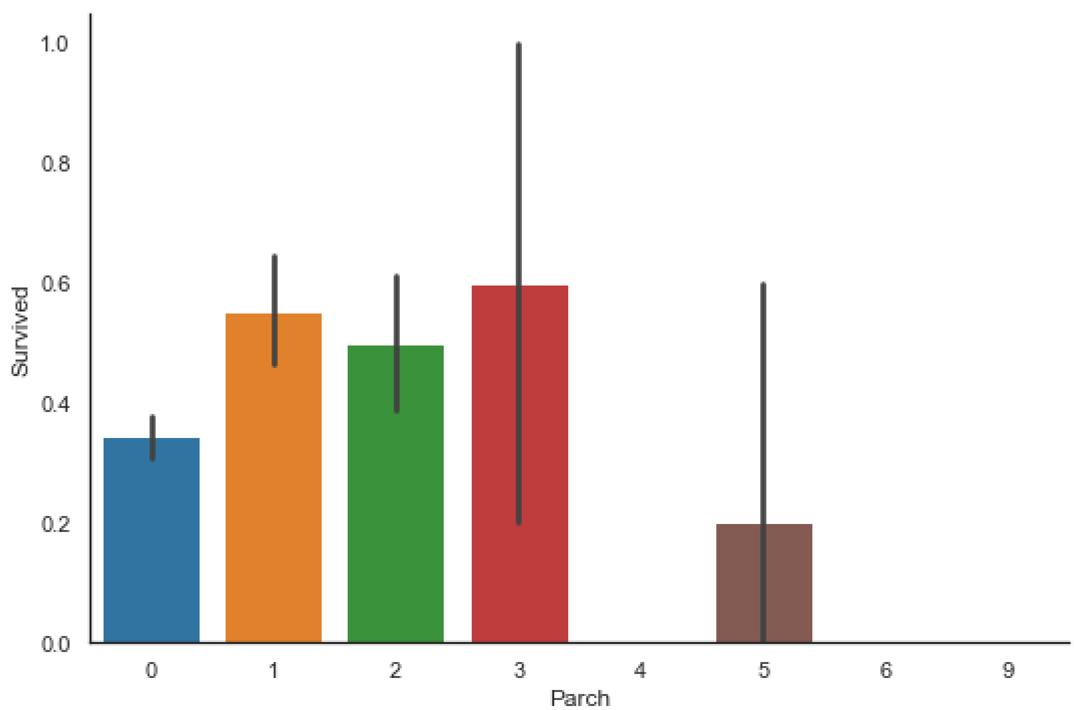
In [23]:
```
df.Parch.value_counts()
```

Out[23]:
```
0    1002
1     170
2     113
3       8
5       6
4       6
9       2
6       2
Name: Parch, dtype: int64
```

Passengers travelling with up to 3 children have slightly higher possibility to survive. However, Parch=3 has a very large standard error.

In [24]:
```
sns.catplot(x='Parch', y='Survived', kind='bar', data=df, aspect=1.5)
```

Out[24]: <seaborn.axisgrid.FacetGrid at 0x1de96481248>



2.8: Embarked

In [25]:
```
df.Embarked.value_counts()
```

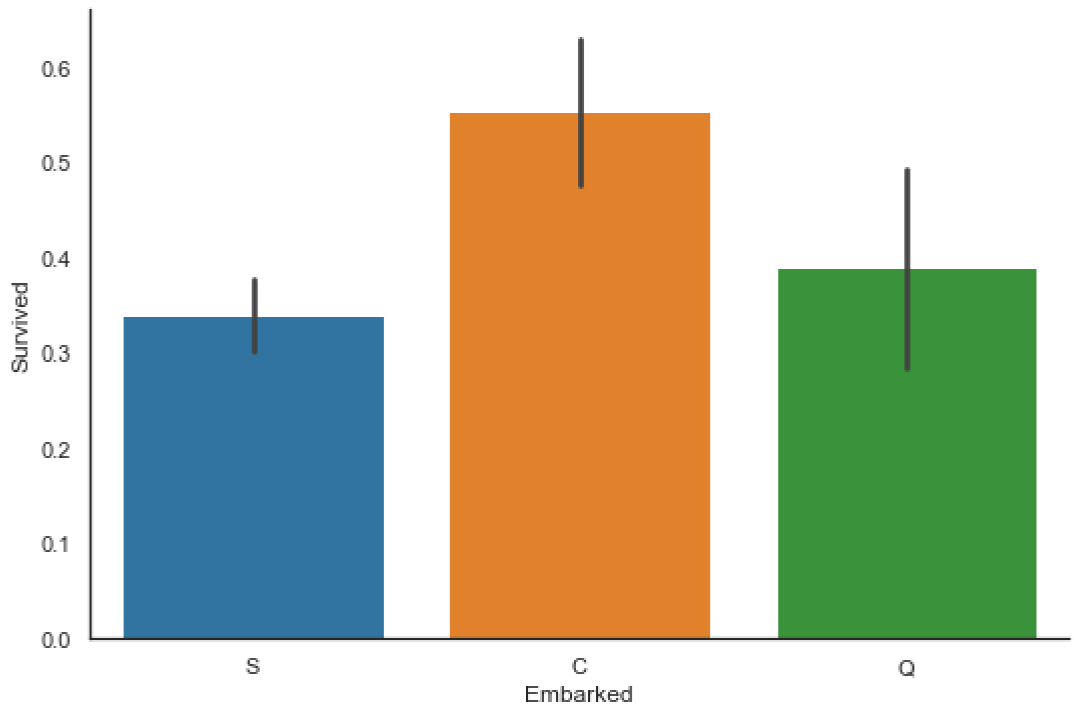
Out[25]:
```
S    914
C    270
Q    123
Name: Embarked, dtype: int64
```

There are only two missing values in Embarked feature. We will repace those with S, the most frequent value.

In [26]:
```
df.Embarked.fillna('S', inplace=True)
df.Embarked.isna().sum()
```
Out[26]:
```
0
```

Passengers from Cherbourg have highest probablity of surviving.

In [27]:
```
sns.catplot(x='Embarked', y='Survived', kind='bar', data=df, aspect=1.5)
```
Out[27]:
```
<seaborn.axisgrid.FacetGrid at 0x1de96139fc8>
```



2.9: Ticket

We will categorise all "number only" tickets as Numeric.

In [28]:
```
df.loc[df.Ticket.str.isnumeric(), 'Ticket'] = 'NUMERIC'
```

Then, remove . and / from the ticket numbers.

In [29]:
```
df['Ticket'] = df['Ticket'].map(lambda tkt: tkt.replace('/', '').replace('.', 
'').replace(' ', ''))
```

Then, just keep first two characters of the ticket number.

In [30]:
```
df['Ticket'] = df['Ticket'].map(lambda tkt: tkt[0:2])
```

Replacing low frequency ticket numbers with Other

In [31]:
```
tkt_counts = df['Ticket'].value_counts()
low_tkt_counts = list(tkt_counts[tkt_counts < 20].index)
df['Ticket'].replace(low_tkt_counts, 'OTH', inplace=True)
```

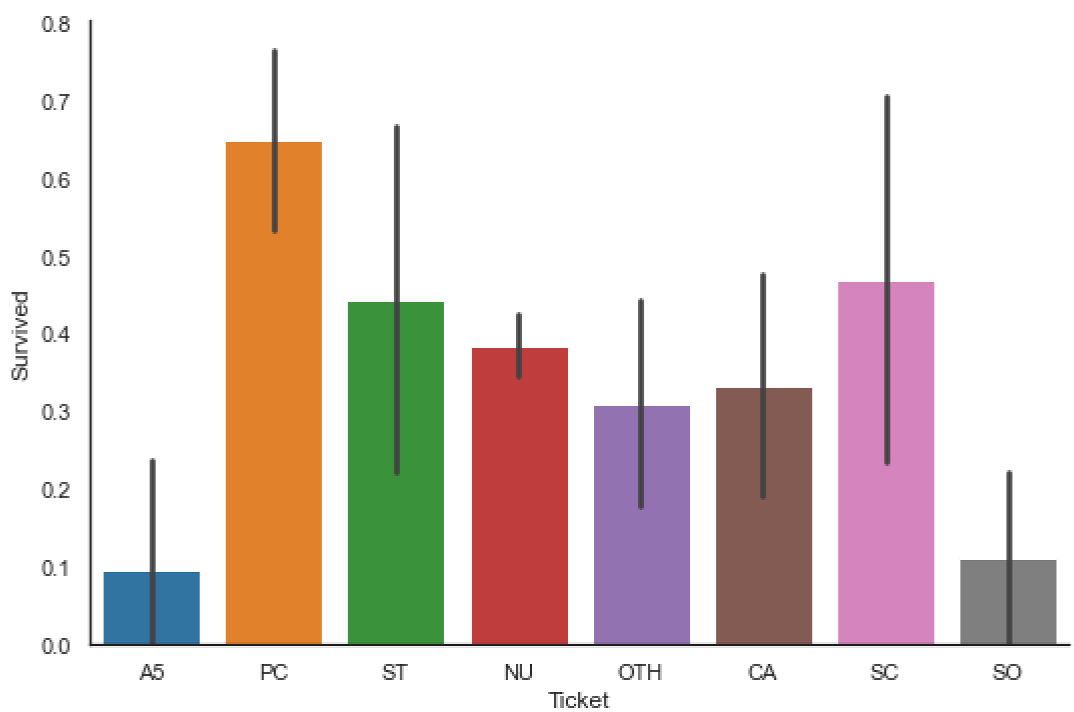
In [32]:
```
df['Ticket'].value_counts()
```

Out[32]:
```
NU     957
PC      92
CA      69
OTH     68
SO      43
SC      30
A5      28
ST      22
Name: Ticket, dtype: int64
```

PC Ticket holders have highest probablity of survival.

In [33]:
```
sns.catplot(x='Ticket', y='Survived', kind='bar', data=df, aspect=1.5)
```

Out[33]: <seaborn.axisgrid.FacetGrid at 0x1de95e67d88>



2.10: Fare

We have only one missing fare which we will replace with mean for same Pclass, Sex, Age

In [34]:
```
df[df.Fare.isna()]
```

Out[34]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	E
1043	1044	NaN	3	Storey, Mr. Thomas	male	60.5	0	0	NU	NaN	NaN	

In [35]:
```
df.groupby(by=['Pclass', 'Sex', 'Age'])['Fare'].mean()
```

Out[35]:
```
Pclass  Sex     Age
1       female  2.0     151.550000
                14.0    120.000000
                15.0    211.337500
                16.0     61.293067
                17.0     82.950000
                           ...
3       male    60.5           NaN
                61.0      6.237500
                65.0      7.750000
                70.5      7.750000
                74.0      7.775000
Name: Fare, Length: 369, dtype: float64
```

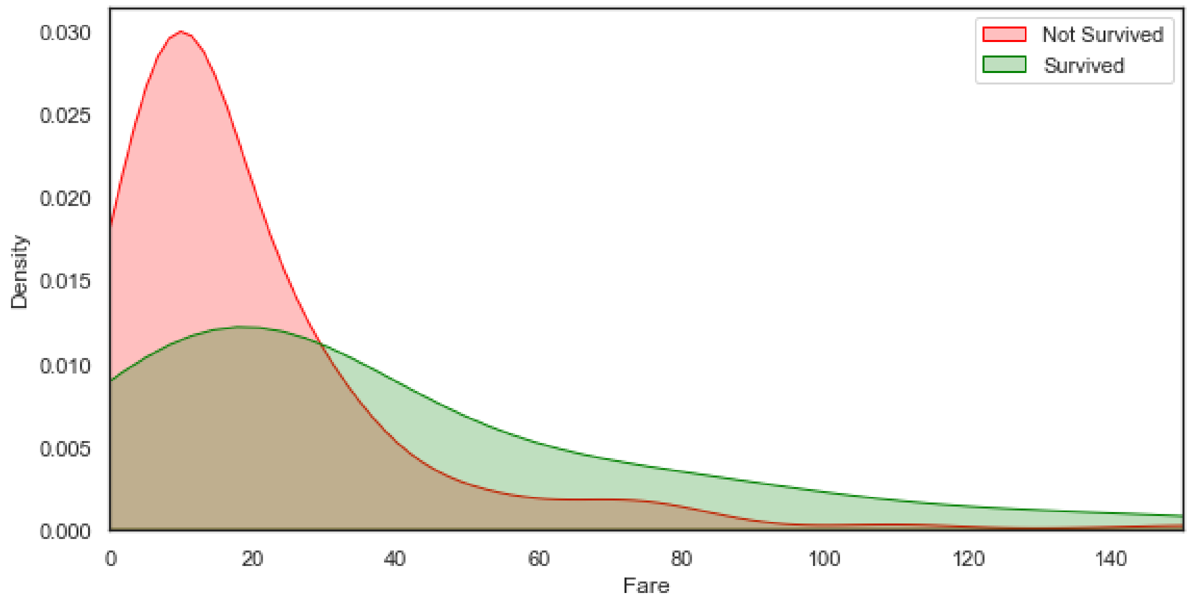
Closest fare is in between 6 and 7. So, replacing with 6.5

In [36]:
```
df.Fare.fillna(6.5, inplace=True)
```

Passengers paying less than 40 ticket fare have very high probablity of not surviving.

In [37]:
```
fig, ax = plt.subplots(figsize=(10, 5))
ax = sns.kdeplot(data = df.loc[(df.Fare.notnull()) & (df.Survived == 0), 'Far
e'], shade = True, color = 'red')
ax = sns.kdeplot(data = df.loc[(df.Fare.notnull()) & (df.Survived == 1), 'Far
e'], shade = True, color = 'green')
ax.legend(['Not Survived', 'Survived'])
ax.set_xlim(0, 150)
```

Out[37]:
```
(0.0, 150.0)
```



2.11: Cabin

Cabin has many missing values. We will keep first character from the cabin number and replace missing values with U as in Unknown.

In [38]:
```
df.loc[df.Cabin.isnull(), 'Cabin'] = 'U'
```

In [39]:
```
df['Cabin'] = df['Cabin'].map(lambda cab: cab[0:1])
```

In [40]:
```
df.Cabin.value_counts()
```

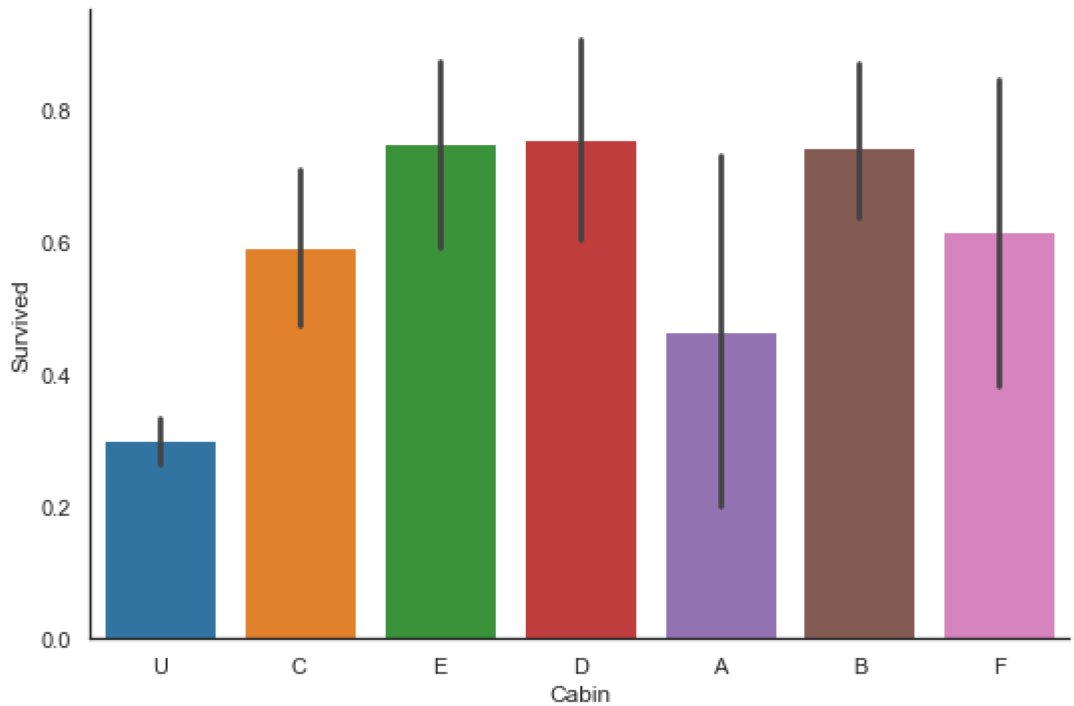
Out[40]:
```
U    1014
C      94
B      65
D      46
E      41
A      22
F      21
G       5
T       1
Name: Cabin, dtype: int64
```

```
In [41]: # As G & T have very low numbers, we will classify those as Unknown as well
         df['Cabin'].replace(['G', 'T'], 'U', inplace=True)
```

Cabin doesn't show much correlation with survival. Unknown cabin is showing low survival probablity only because of high frequency.

```
In [42]: sns.catplot(x='Cabin', y='Survived', kind='bar', data=df, aspect=1.5)
```

```
Out[42]: <seaborn.axisgrid.FacetGrid at 0x1de95edfa08>
```



Section 3: Feature Engineering

3.1: Encoding Categorical & Binary Columns

Before the encoding, we will drop the columns which are not required for modelling.

```
In [43]: df.drop(['PassengerId', 'Name'], axis=1, inplace=True)
         df.shape
```

```
Out[43]: (1309, 11)
```

Let's identify the different type of columns first.

```
In [44]: numeric_cols = list(df.columns[df.dtypes == 'int64']) + list(df.columns[df.dty
         pes == 'float64'])
         numeric_cols.remove('Survived')
         binary_cols = list(['Sex'])
         cat_cols = list(set(df.columns) - set(numeric_cols) - set(binary_cols))
         cat_cols.remove('Survived')
```

```
In [45]: numeric_cols, binary_cols, cat_cols
```

```
Out[45]: (['Pclass', 'SibSp', 'Parch', 'Age', 'Fare'],
          ['Sex'],
          ['Ticket', 'Embarked', 'Salutation', 'Cabin'])
```

Encoding Binary and Categorical Columns

```
In [46]: 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, it's time to separate the labelled and unlabelled data, and create train-test splits.

```
In [47]: # Separate labelled and unlabelled data
         unlabelled_df = df.loc[df.Survived.isna()]
         X = df.loc[df.Survived.notna()].drop('Survived', axis=1)
         y = df.loc[df.Survived.notna()]['Survived']
```

```
In [48]: # 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)
```

3.2: Scaling Numerical columns

```
In [50]: from sklearn.preprocessing import StandardScaler

         sc = StandardScaler()

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

Section 4: Deep Learning Models

In this section, we will train multiple neural network models and compare performances using accuracy. Once we find the best model, we can check other metrics.

4.1: Base model

We will generate Logistic Regression model as a base model. It will be used to compare the accuracies of the deep learning models.

In [51]:
```
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)
```

Out[51]:
```
GridSearchCV(cv=KFold(n_splits=4, random_state=None, shuffle=True),
             estimator=LogisticRegressionCV(), n_jobs=-1,
             param_grid={'Cs': [2, 5, 10], 'penalty': ['l1', 'l2'],
                         'solver': ['newton-cg', 'lbfgs', 'liblinear']},
             scoring='accuracy')
```

In [52]:
```
print('Logistic Regression score: ', round(grid.best_score_, 4))
```

```
Logistic Regression score:  0.8329
```

4.2: Basic Neural Network

A neural network with no hidden layer.

In [53]:
```python
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
_________________________________________________________________
```

In [54]:

```
nn1.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = 'ac
curacy')
nn1.fit(X_train_scaled, y_train, batch_size = 32, epochs = 20, validation_data
= (X_test_scaled, y_test))
```

```
Epoch 1/20
23/23 [==============================] - 0s 7ms/step - loss: 0.6622 - accurac
y: 0.6067 - val_loss: 0.5906 - val_accuracy: 0.7486
Epoch 2/20
23/23 [==============================] - 0s 2ms/step - loss: 0.5742 - accurac
y: 0.7346 - val_loss: 0.5229 - val_accuracy: 0.7933
Epoch 3/20
23/23 [==============================] - 0s 2ms/step - loss: 0.5225 - accurac
y: 0.7697 - val_loss: 0.4825 - val_accuracy: 0.7933
Epoch 4/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4885 - accurac
y: 0.7893 - val_loss: 0.4520 - val_accuracy: 0.7933
Epoch 5/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4671 - accurac
y: 0.7963 - val_loss: 0.4329 - val_accuracy: 0.7933
Epoch 6/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4533 - accurac
y: 0.7992 - val_loss: 0.4197 - val_accuracy: 0.7989
Epoch 7/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4438 - accurac
y: 0.8048 - val_loss: 0.4123 - val_accuracy: 0.8101
Epoch 8/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4363 - accurac
y: 0.8104 - val_loss: 0.4054 - val_accuracy: 0.8156
Epoch 9/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4304 - accurac
y: 0.8202 - val_loss: 0.4003 - val_accuracy: 0.8156
Epoch 10/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4251 - accurac
y: 0.8174 - val_loss: 0.3973 - val_accuracy: 0.8156
Epoch 11/20
23/23 [==============================] - 0s 3ms/step - loss: 0.4203 - accurac
y: 0.8272 - val_loss: 0.3958 - val_accuracy: 0.8212
Epoch 12/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4166 - accurac
y: 0.8272 - val_loss: 0.3932 - val_accuracy: 0.8268
Epoch 13/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4127 - accurac
y: 0.8301 - val_loss: 0.3914 - val_accuracy: 0.8268
Epoch 14/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4090 - accurac
y: 0.8329 - val_loss: 0.3898 - val_accuracy: 0.8324
Epoch 15/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4056 - accurac
y: 0.8371 - val_loss: 0.3878 - val_accuracy: 0.8324
Epoch 16/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4024 - accurac
y: 0.8343 - val_loss: 0.3869 - val_accuracy: 0.8324
Epoch 17/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3999 - accurac
y: 0.8385 - val_loss: 0.3865 - val_accuracy: 0.8380
Epoch 18/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3971 - accurac
y: 0.8385 - val_loss: 0.3850 - val_accuracy: 0.8380
Epoch 19/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3947 - accurac
y: 0.8385 - val_loss: 0.3829 - val_accuracy: 0.8492
```

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

Out[54]:
```
<tensorflow.python.keras.callbacks.History at 0x1de9ccf4148>
```

We are able to match the accuracy just with a basic neural network. Let's try to improve on it.

4.3: Deep Neural Network

We will introduce one hidden layer.

In [55]:
```
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
_________________________________________________________________
```

In [56]:
```
nn2.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = 'ac
curacy')
nn2.fit(X_train_scaled, y_train, batch_size = 32, epochs = 20, validation_data
= (X_test_scaled, y_test))
```

```
Epoch 1/20
23/23 [==============================] - 0s 6ms/step - loss: 0.6345 - accurac
y: 0.6334 - val_loss: 0.5371 - val_accuracy: 0.7709
Epoch 2/20
23/23 [==============================] - 0s 2ms/step - loss: 0.5356 - accurac
y: 0.7767 - val_loss: 0.4699 - val_accuracy: 0.8268
Epoch 3/20
23/23 [==============================] - 0s 1ms/step - loss: 0.4867 - accurac
y: 0.8006 - val_loss: 0.4354 - val_accuracy: 0.8436
Epoch 4/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4611 - accurac
y: 0.8062 - val_loss: 0.4188 - val_accuracy: 0.8324
Epoch 5/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4431 - accurac
y: 0.8146 - val_loss: 0.4059 - val_accuracy: 0.8380
Epoch 6/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4296 - accurac
y: 0.8258 - val_loss: 0.4002 - val_accuracy: 0.8436
Epoch 7/20
23/23 [==============================] - 0s 1ms/step - loss: 0.4185 - accurac
y: 0.8258 - val_loss: 0.3932 - val_accuracy: 0.8436
Epoch 8/20
23/23 [==============================] - 0s 1ms/step - loss: 0.4112 - accurac
y: 0.8230 - val_loss: 0.3881 - val_accuracy: 0.8436
Epoch 9/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4037 - accurac
y: 0.8244 - val_loss: 0.3872 - val_accuracy: 0.8547
Epoch 10/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3973 - accurac
y: 0.8329 - val_loss: 0.3845 - val_accuracy: 0.8380
Epoch 11/20
23/23 [==============================] - 0s 1ms/step - loss: 0.3921 - accurac
y: 0.8329 - val_loss: 0.3815 - val_accuracy: 0.8380
Epoch 12/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3863 - accurac
y: 0.8357 - val_loss: 0.3810 - val_accuracy: 0.8380
Epoch 13/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3829 - accurac
y: 0.8399 - val_loss: 0.3807 - val_accuracy: 0.8380
Epoch 14/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3783 - accurac
y: 0.8413 - val_loss: 0.3783 - val_accuracy: 0.8380
Epoch 15/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3747 - accurac
y: 0.8469 - val_loss: 0.3787 - val_accuracy: 0.8380
Epoch 16/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3721 - accurac
y: 0.8497 - val_loss: 0.3740 - val_accuracy: 0.8436
Epoch 17/20
23/23 [==============================] - 0s 1ms/step - loss: 0.3690 - accurac
y: 0.8497 - val_loss: 0.3714 - val_accuracy: 0.8492
Epoch 18/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3662 - accurac
y: 0.8469 - val_loss: 0.3735 - val_accuracy: 0.8380
Epoch 19/20
23/23 [==============================] - 0s 3ms/step - loss: 0.3626 - accurac
y: 0.8539 - val_loss: 0.3726 - val_accuracy: 0.8436
```

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

Out[56]:

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

Let's see if accuracy improves by introducing more layers.

In [98]:

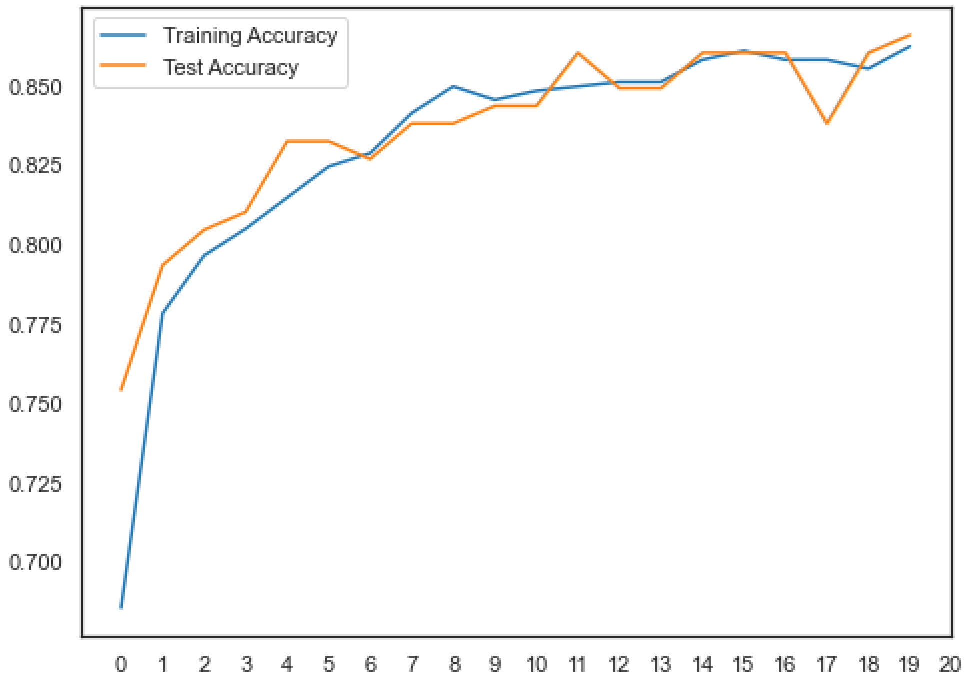
```
nn2 = Sequential()
nn2.add(Dense(units=25, input_shape = (24,), activation='relu'))
nn2.add(Dense(units=50, activation='relu'))
nn2.add(Dense(units=50, activation='relu'))
nn2.add(Dense(units=1, activation='sigmoid'))
nn2.compile(optimizer = 'rmsprop', loss = 'binary_crossentropy', metrics = 'ac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

```
Epoch 1/20
23/23 [==============================] - 0s 7ms/step - loss: 0.6231 - accurac
y: 0.6854 - val_loss: 0.5368 - val_accuracy: 0.7542
Epoch 2/20
23/23 [==============================] - 0s 2ms/step - loss: 0.5203 - accurac
y: 0.7781 - val_loss: 0.4631 - val_accuracy: 0.7933
Epoch 3/20
23/23 [==============================] - 0s 3ms/step - loss: 0.4727 - accurac
y: 0.7963 - val_loss: 0.4351 - val_accuracy: 0.8045
Epoch 4/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4492 - accurac
y: 0.8048 - val_loss: 0.4223 - val_accuracy: 0.8101
Epoch 5/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4320 - accurac
y: 0.8146 - val_loss: 0.4154 - val_accuracy: 0.8324
Epoch 6/20
23/23 [==============================] - 0s 3ms/step - loss: 0.4181 - accurac
y: 0.8244 - val_loss: 0.4070 - val_accuracy: 0.8324
Epoch 7/20
23/23 [==============================] - 0s 3ms/step - loss: 0.4074 - accurac
y: 0.8287 - val_loss: 0.4112 - val_accuracy: 0.8268
Epoch 8/20
23/23 [==============================] - 0s 2ms/step - loss: 0.4004 - accurac
y: 0.8413 - val_loss: 0.4007 - val_accuracy: 0.8380
Epoch 9/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3867 - accurac
y: 0.8497 - val_loss: 0.3935 - val_accuracy: 0.8380
Epoch 10/20
23/23 [==============================] - 0s 3ms/step - loss: 0.3799 - accurac
y: 0.8455 - val_loss: 0.3955 - val_accuracy: 0.8436
Epoch 11/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3753 - accurac
y: 0.8483 - val_loss: 0.3892 - val_accuracy: 0.8436
Epoch 12/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3677 - accurac
y: 0.8497 - val_loss: 0.3847 - val_accuracy: 0.8603
Epoch 13/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3594 - accurac
y: 0.8511 - val_loss: 0.3823 - val_accuracy: 0.8492
Epoch 14/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3580 - accurac
y: 0.8511 - val_loss: 0.3792 - val_accuracy: 0.8492
Epoch 15/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3509 - accurac
y: 0.8581 - val_loss: 0.3728 - val_accuracy: 0.8603
Epoch 16/20
23/23 [==============================] - 0s 3ms/step - loss: 0.3463 - accurac
y: 0.8610 - val_loss: 0.3793 - val_accuracy: 0.8603
Epoch 17/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3420 - accurac
y: 0.8581 - val_loss: 0.3836 - val_accuracy: 0.8603
Epoch 18/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3394 - accurac
y: 0.8581 - val_loss: 0.3840 - val_accuracy: 0.8380
Epoch 19/20
23/23 [==============================] - 0s 2ms/step - loss: 0.3351 - accurac
y: 0.8553 - val_loss: 0.3806 - val_accuracy: 0.8603
```

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

Let's plot the accuracies captured in each epochs.

In [110]:
```
fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(1,1,1)
ax.plot(nn2_steps.history['accuracy'], label = 'Training Accuracy')
ax.plot(nn2_steps.history['val_accuracy'], label = 'Test Accuracy')
plt.xticks(range(0,21,1))
plt.legend()
plt.tight_layout()
```



4.4: Second Deep Neural Network

We will now try 'adam' optimizer, smaller batch size and more epochs.

In [105]:

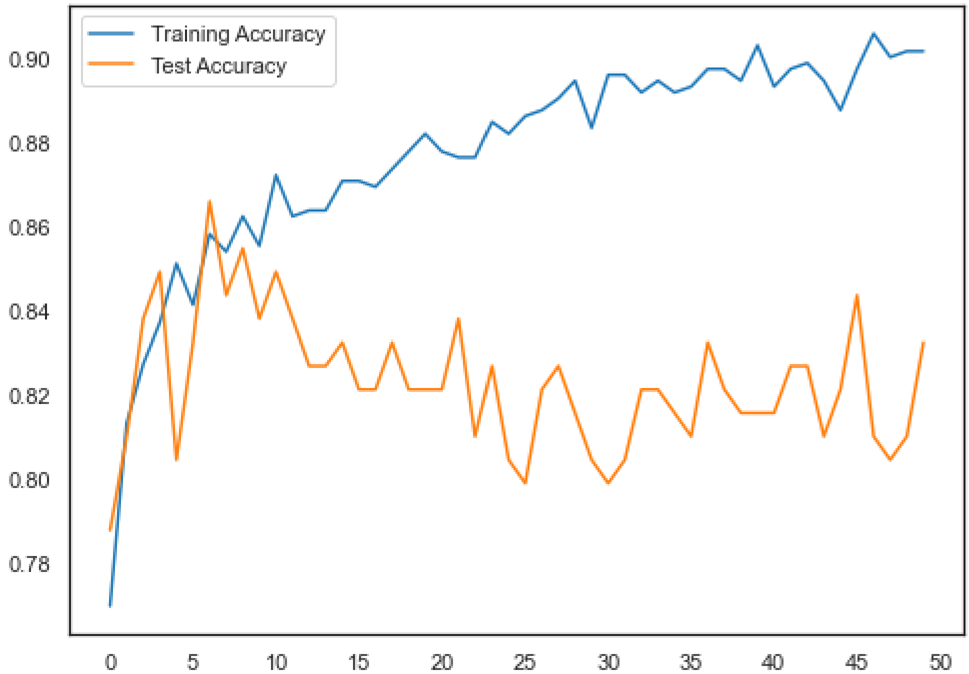
```
nn3 = Sequential()
nn3.add(Dense(units=25, input_shape = (24,), activation='relu'))
nn3.add(Dense(units=50, activation='relu'))
nn3.add(Dense(units=50, activation='relu'))
nn3.add(Dense(units=1, activation='sigmoid'))
nn3.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = 'accur
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)
```

```
Epoch 1/50
712/712 [==============================] - 1s 1ms/step - loss: 0.5127 - accur
acy: 0.7697 - val_loss: 0.4270 - val_accuracy: 0.7877
Epoch 2/50
712/712 [==============================] - 1s 1ms/step - loss: 0.4391 - accur
acy: 0.8132 - val_loss: 0.4143 - val_accuracy: 0.8101
Epoch 3/50
712/712 [==============================] - 1s 1ms/step - loss: 0.4138 - accur
acy: 0.8272 - val_loss: 0.3776 - val_accuracy: 0.8380
Epoch 4/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3906 - accur
acy: 0.8371 - val_loss: 0.4019 - val_accuracy: 0.8492
Epoch 5/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3743 - accur
acy: 0.8511 - val_loss: 0.4106 - val_accuracy: 0.8045
Epoch 6/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3700 - accur
acy: 0.8413 - val_loss: 0.3793 - val_accuracy: 0.8324
Epoch 7/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3559 - accur
acy: 0.8581 - val_loss: 0.3550 - val_accuracy: 0.8659
Epoch 8/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3529 - accur
acy: 0.8539 - val_loss: 0.3928 - val_accuracy: 0.8436
Epoch 9/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3377 - accur
acy: 0.8624 - val_loss: 0.3855 - val_accuracy: 0.8547
Epoch 10/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3445 - accur
acy: 0.8553 - val_loss: 0.3824 - val_accuracy: 0.8380
Epoch 11/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3274 - accur
acy: 0.8722 - val_loss: 0.4089 - val_accuracy: 0.8492
Epoch 12/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3274 - accur
acy: 0.8624 - val_loss: 0.4000 - val_accuracy: 0.8380
Epoch 13/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3208 - accur
acy: 0.8638 - val_loss: 0.3952 - val_accuracy: 0.8268
Epoch 14/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3196 - accur
acy: 0.8638 - val_loss: 0.3734 - val_accuracy: 0.8268
Epoch 15/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3057 - accur
acy: 0.8708 - val_loss: 0.4090 - val_accuracy: 0.8324
Epoch 16/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3146 - accur
acy: 0.8708 - val_loss: 0.3832 - val_accuracy: 0.8212
Epoch 17/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3092 - accur
acy: 0.8694 - val_loss: 0.4118 - val_accuracy: 0.8212
Epoch 18/50
712/712 [==============================] - 1s 1ms/step - loss: 0.3002 - accur
acy: 0.8736 - val_loss: 0.4084 - val_accuracy: 0.8324
Epoch 19/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2985 - accur
acy: 0.8778 - val_loss: 0.4247 - val_accuracy: 0.8212
```

```
Epoch 20/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2952 - accur
acy: 0.8820 - val_loss: 0.4186 - val_accuracy: 0.8212
Epoch 21/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2943 - accur
acy: 0.8778 - val_loss: 0.4245 - val_accuracy: 0.8212
Epoch 22/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2910 - accur
acy: 0.8764 - val_loss: 0.3888 - val_accuracy: 0.8380
Epoch 23/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2975 - accur
acy: 0.8764 - val_loss: 0.4564 - val_accuracy: 0.8101
Epoch 24/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2794 - accur
acy: 0.8848 - val_loss: 0.4562 - val_accuracy: 0.8268
Epoch 25/50
712/712 [==============================] - 1s 2ms/step - loss: 0.2819 - accur
acy: 0.8820 - val_loss: 0.4440 - val_accuracy: 0.8045
Epoch 26/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2860 - accur
acy: 0.8862 - val_loss: 0.4552 - val_accuracy: 0.7989
Epoch 27/50
712/712 [==============================] - 1s 2ms/step - loss: 0.2735 - accur
acy: 0.8876 - val_loss: 0.4640 - val_accuracy: 0.8212
Epoch 28/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2709 - accur
acy: 0.8904 - val_loss: 0.4381 - val_accuracy: 0.8268
Epoch 29/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2675 - accur
acy: 0.8947 - val_loss: 0.4499 - val_accuracy: 0.8156
Epoch 30/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2710 - accur
acy: 0.8834 - val_loss: 0.4732 - val_accuracy: 0.8045
Epoch 31/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2616 - accur
acy: 0.8961 - val_loss: 0.5229 - val_accuracy: 0.7989
Epoch 32/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2630 - accur
acy: 0.8961 - val_loss: 0.5453 - val_accuracy: 0.8045
Epoch 33/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2629 - accur
acy: 0.8919 - val_loss: 0.4945 - val_accuracy: 0.8212
Epoch 34/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2568 - accur
acy: 0.8947 - val_loss: 0.4986 - val_accuracy: 0.8212
Epoch 35/50
712/712 [==============================] - 1s 2ms/step - loss: 0.2539 - accur
acy: 0.8919 - val_loss: 0.5162 - val_accuracy: 0.8156
Epoch 36/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2493 - accur
acy: 0.8933 - val_loss: 0.5812 - val_accuracy: 0.8101
Epoch 37/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2482 - accur
acy: 0.8975 - val_loss: 0.5447 - val_accuracy: 0.8324
Epoch 38/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2517 - accur
acy: 0.8975 - val_loss: 0.5727 - val_accuracy: 0.8212
```

```
Epoch 39/50
712/712 [==============================] - 1s 2ms/step - loss: 0.2482 - accur
acy: 0.8947 - val_loss: 0.5532 - val_accuracy: 0.8156
Epoch 40/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2482 - accur
acy: 0.9031 - val_loss: 0.5444 - val_accuracy: 0.8156
Epoch 41/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2428 - accur
acy: 0.8933 - val_loss: 0.5765 - val_accuracy: 0.8156
Epoch 42/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2453 - accur
acy: 0.8975 - val_loss: 0.6245 - val_accuracy: 0.8268
Epoch 43/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2412 - accur
acy: 0.8989 - val_loss: 0.5888 - val_accuracy: 0.8268
Epoch 44/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2346 - accur
acy: 0.8947 - val_loss: 0.7620 - val_accuracy: 0.8101
Epoch 45/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2939 - accur
acy: 0.8876 - val_loss: 0.5887 - val_accuracy: 0.8212
Epoch 46/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2379 - accur
acy: 0.8975 - val_loss: 0.6382 - val_accuracy: 0.8436
Epoch 47/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2248 - accur
acy: 0.9059 - val_loss: 0.6498 - val_accuracy: 0.8101
Epoch 48/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2282 - accur
acy: 0.9003 - val_loss: 0.6508 - val_accuracy: 0.8045
Epoch 49/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2250 - accur
acy: 0.9017 - val_loss: 0.6252 - val_accuracy: 0.8101
Epoch 50/50
712/712 [==============================] - 1s 1ms/step - loss: 0.2246 - accur
acy: 0.9017 - val_loss: 0.6986 - val_accuracy: 0.8324
```

In [109]:
```
fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(1,1,1)
ax.plot(nn3_steps.history['accuracy'], label = 'Training Accuracy')
ax.plot(nn3_steps.history['val_accuracy'], label = 'Test Accuracy')
plt.xticks(range(0,51,5))
plt.legend()
plt.tight_layout()
```



4.5: Model Selection & Performance

Our "Second Deep Neural Network" (nn3) is overfitted to the training data. So, we will stick to the first one (nn2) and measure other performance metrics.

In [130]:
```
y_pred = nn2.predict(X_test_scaled)
y_pred[y_pred < 0.5] = 0
y_pred[y_pred >= 0.5] = 1
```

In [132]:
```
from sklearn.metrics import confusion_matrix, classification_report
print(classification_report(y_test, y_pred))
```

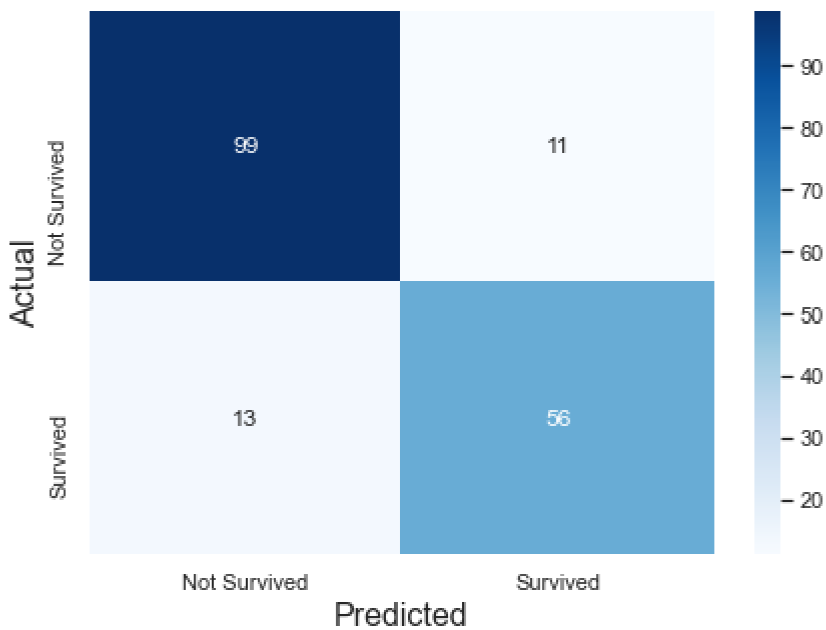
```
              precision    recall  f1-score   support

         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
```

In [133]:
```
cm = confusion_matrix(y_test, y_pred)
fig, ax = plt.subplots(figsize=(7, 5))
ax = sns.heatmap(cm, fmt='d', annot=True, cmap='Blues')
ax.set_xticklabels(['Not Survived', 'Survived'])
ax.set_yticklabels(['Not Survived', 'Survived'])
plt.xlabel('Predicted', fontsize=16)
plt.ylabel('Actual', fontsize=16)
```

Out[133]:
```
Text(39.5, 0.5, 'Actual')
```



Section 5: Predictions for Unlabelled data

As we don't know the labels, we will have to prepare and submit the prediction on Kaggle to get the accuracy score.

In [134]:
```
X_unlabelled = unlabelled_df.drop('Survived', axis=1)
X_unlabelled_scaled = sc.transform(X_unlabelled)

y_unlabelled_pred = nn2.predict(X_unlabelled_scaled)
y_unlabelled_pred[y_unlabelled_pred < 0.5] = 0
y_unlabelled_pred[y_unlabelled_pred >= 0.5] = 1
```

In [147]:
```
pid_df = pd.read_csv('test.csv')
pids = pid_df.iloc[:, 0]
results = pd.concat([pids.astype('int'), pd.Series(y_unlabelled_pred[:,0]).ast
ype('int')], axis=1)
results.columns = ['PassengerId', 'Survived']
results.to_csv('titanic_result_7Dec.csv', index=False)
```

Summary

We have noticed significant performance imrpovement with the neural network models. Even though, Age, Gender and Travel Class show very strong correlation with survival probablity, I guess, there's always a luck factor.

Note: I submitted the results in Kaggle for the unlabelled dataset and got accuracy of 0.75119.

Next steps:

What I haven't done in this notebook is to check correlation between features and multicollinearity. I also didn't check and fine-tune all the hyper-parameters. We may be able to improve the model performance by removing multicollinearity and having more precise values of model hyper-parameters.