```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#Importing the libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

import seaborn as sns
```

Loading training dataset

```
df_train = pd.read_csv("./drive/MyDrive/ML/train.csv")
```

Exploratory Data Analysis

First 5 rows of training dataset

```
df_train.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th	female	38.0	1	0	PC 17599	71.

```
df_train.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	F
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000

Information about the data in each column, and the column name.

- Non-NaN count
- Datatype

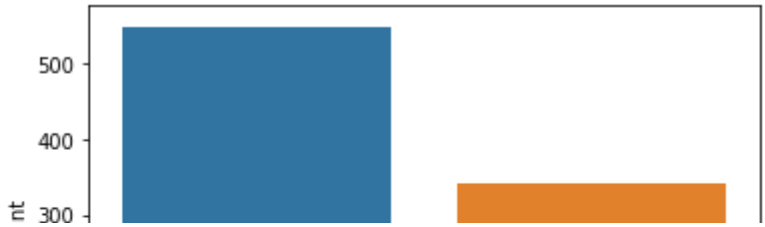
```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  891 non-null    int64  
 1   Survived     891 non-null    int64  
 2   Pclass       891 non-null    int64  
 3   Name         891 non-null    object 
 4   Sex          891 non-null    object 
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64  
 7   Parch        891 non-null    int64  
 8   Ticket       891 non-null    object 
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object 
 11  Embarked     889 non-null    object 
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

The number of people who survived and who died

```
sns.countplot(x="Survived",data=df_train)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd52c3909d0>
```

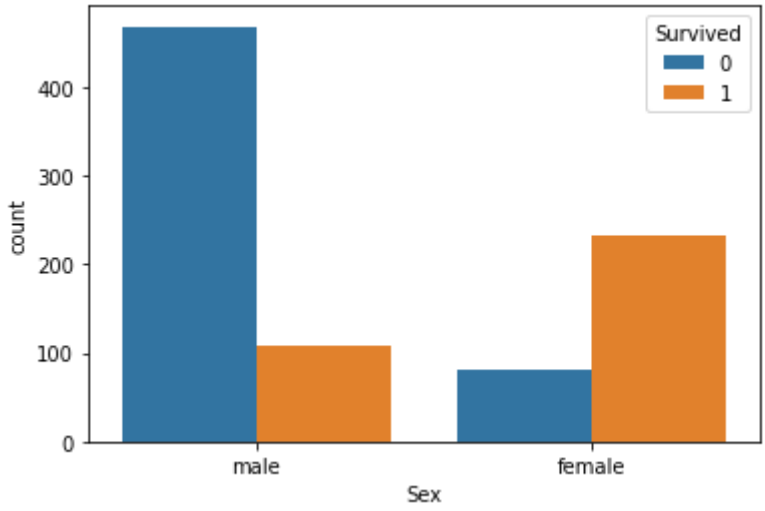


The number of people who survived and who died based on sex:

- Observation: Much more females survived than males, and much more males died than females.

```
sns.countplot(x="Sex",data=df_train,hue="Survived")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd52c27ce50>
```



The number of people who survived and who died based on which class they were in:

- First class being the most luxurious and 3rd class being kind of like economy class on an airplane.

```
sns.countplot(x="Pclass",data=df_train,hue="Survived")
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd52bd9d850>
```

The number of people who survived and who died based on how many siblings/spouses they had with them:

[ ] ↳ *1 cell hidden*

The number of people who survived and who died based on how many parents/children they had with them:

[ ] ↳ *1 cell hidden*

Box plot comparing survivor's fare and the fare of the people who died:

[ ] ↳ *1 cell hidden*

The number of people who survived or died based on which port they embarked on:

- C = Cherbourg, Q = Queenstown, S = Southampton

[ ] ↳ *1 cell hidden*

Data Preprocessing: Training Dataset

- Remove missing data
- Remove unnecessary or repetitive features
- Convert string features to dummy variables

First we need to know which columns/features have missing data.

```
df_train.isnull().sum()
```

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

Starting with Age, all NaN values are filled with the mean of all of the values in the Age column .

```
df_train["Age"] = df_train["Age"].fillna(df_train["Age"].mean())
```

If you haven't noticed, those who have a recorded cabin number(those who's cabin number isn't NaN) have a much larger fare than those who don't have a recorded cabin number. Below is the proof:

```
# Defining a function that returns "Cabin# Recorded" if the value is not NaN and "No Cabin
def NaN_vs_NotNaN(value):
    if pd.isnull(value) == False:
        return "Cabin# Recorded"
    else:
        return "No Cabin# Recorded"


# Creating the new pandas dataframe from two columns from the training dataset: Fare and C
new_df = pd.concat([df_train["Fare"],df_train["Cabin"]],axis=1)
new_df["Cabin"] = new_df["Cabin"].apply(NaN_vs_NotNaN) # Using df.apply() will pass every
new_df
```

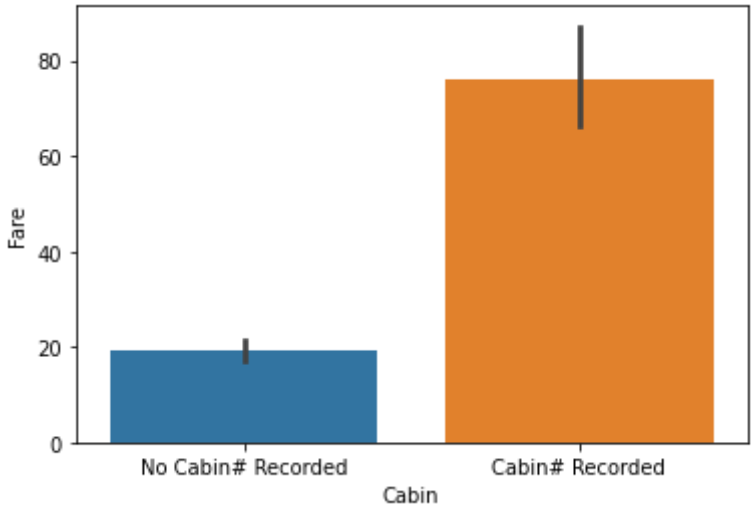
	Fare	Cabin
0	7.2500	No Cabin# Recorded
1	71.2833	Cabin# Recorded
2	7.9250	No Cabin# Recorded
3	53.1000	Cabin# Recorded

As you can see, the fares of passengers with a recorded cabin number is MUCH higher than the fares of passengers without a recorded cabin number This means that the Cabin column in the training dataset is useless because it basically resembles the Fare column.

887 30.0000 Cabin# Recorded

```
# Creating a barplot using seaborn
sns.barplot(x="Cabin", y="Fare", data=new_df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fd52baab810>
```



Therefore we need to drop the Cabin feature.

```
df_train = df_train.drop("Cabin",axis=1)
```

Now there is only the embarked column that contains missing values. There are only 2 rows that contain missing values, so dropna() is a good idea because dropping 2 columns won't make much of a difference in the training process.

```
df_train = df_train.dropna()
```

```
# Double confirm that there are no more missing values left in the dataset
df_train.isnull().sum()
```

```
PassengerId    0
Survived       0
Pclass         0
Name           0
```

```
Sex           0
Age           0
SibSp         0
Parch         0
Ticket        0
Fare          0
Embarked      0
dtype: int64
```

Removing the remaining useless columns

```
df_train = df_train.drop(["Name","PassengerId","Ticket"],axis=1)
```

Since there are only 2 categories in the Sex feature, it can be converted to binary.

```
df_train["Sex"] = df_train["Sex"].map({"male":1,"female":0})
```

Converting the Embarked column to dummy variables:

```
embarked_dummies = pd.get_dummies(df_train["Embarked"],drop_first=True)
df_train = pd.concat([df_train.drop("Embarked",axis=1),embarked_dummies],axis=1)
```

This is what the training dataset looks like now:

```
df_train
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Q	S
0	0	3	1	22.000000	1	0	7.2500	0	1
1	1	1	0	38.000000	1	0	71.2833	0	0
2	1	3	0	26.000000	0	0	7.9250	0	1
3	1	1	0	35.000000	1	0	53.1000	0	1
4	0	3	1	35.000000	0	0	8.0500	0	1
...	...	...	...	...	...	...	...	...	...
886	0	2	1	27.000000	0	0	13.0000	0	1
887	1	1	0	19.000000	0	0	30.0000	0	1
888	0	3	0	29.699118	1	2	23.4500	0	1
889	1	1	1	26.000000	0	0	30.0000	0	0
890	0	3	1	32.000000	0	0	7.7500	1	0

889 rows × 9 columns

Loading test dataset:

```
df_test = pd.read_csv("./drive/MyDrive/ML/test.csv")
df_test
```

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

Filling null values and removing useless columns. Basically doing the same thing as what was done to df_train, except removing rows is not an option because those rows are still needed to predict if the passenger has survived or not.

```
df_test["Age"] = df_test["Age"].fillna(df_test["Age"].mean())
df_test["Embarked"] = df_test["Embarked"].fillna("S")
df_test = df_test.drop(["Name","PassengerId","Ticket","Cabin"],axis=1)
```

Converting categorical features to integers/dummy variables

```
df_test["Sex"] = df_test["Sex"].map({"male":1,"female":0})
embarked_dummies = pd.get_dummies(df_test["Embarked"],drop_first=True)
df_test = pd.concat([df_test.drop("Embarked",axis=1),embarked_dummies],axis=1)
```

This is what the test dataset looks like now:

```
df_test
```

	Pclass	Sex	Age	SibSp	Parch	Fare	Q	S
0	3	1	34.50000	0	0	7.8292	1	0
1	3	0	47.00000	1	0	7.0000	0	1
2	2	1	62.00000	0	0	9.6875	1	0
3	3	1	27.00000	0	0	8.6625	0	1
4	3	0	22.00000	1	1	12.2875	0	1
...	...	...	...	...	...	...	...	...
413	3	1	30.27259	0	0	8.0500	0	1
414	1	0	39.00000	0	0	108.9000	0	0
415	3	1	38.50000	0	0	7.2500	0	1
416	3	1	30.27259	0	0	8.0500	0	1
417	3	1	30.27259	1	1	22.3583	0	0

418 rows × 8 columns

Normalizing the data

Creating separate pandas dataframes. X is the features, y is the survived column, the "answer sheet".

```
X = df_train.drop('Survived',axis=1).values
y = df_train['Survived'].values


x_eval = df_test.values
```

Importing the libraries

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

Train test split

```
X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=101, test_size=0.3)
```

Using MinMaxScaler()

```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
```

```
X_test = scaler.transform(X_test)
x_eval = scaler.transform(x_eval)
```

Creating the model

1. SVM

```
from sklearn.svm import SVC
from sklearn.model_selection import cross_val_score
from sklearn import metrics
svc = SVC(kernel = 'rbf')
scores = cross_val_score(svc, X, y, scoring='accuracy', error_score='raise') #cv is cross
print(scores)
```

```
[0.58988764 0.71348315 0.68539326 0.68539326 0.68926554]
```

```
# Accuracy on train set based on cross validation
svc.fit(X,y)
y_pred = svc.predict(X)
print("Accuracy on train set is",sum(y_pred==y)/len(y))
```

```
Accuracy on train set is 0.6850393700787402
```

```
# Accuracy on test set
y_test_p= svc.predict(X_test)
print("Accuracy on test set is",sum(y_test_p==y_test)/len(y_test))
```

```
Accuracy on test set is 0.6104868913857678
```

2. Logistic Regression

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression(random_state = 0).fit(X, y)
y_pred = clf.predict(X)
print(sum(y == y_pred))
```

```
716
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: Converg
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
```

```
y_test_pred = clf.predict(X_test)
print("Accuracy =", sum(y_test_pred==y_test)/len(y_test))
```

```
Accuracy = 0.3895131086142322
```

3. Neural Network

```
import os
import torch
from torch import nn
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
from torch.optim import Adam
from torch.utils.data import Dataset, DataLoader
import tqdm
import torch.nn.functional as F
```

```
device = "cuda" if torch.cuda.is_available() else "cpu"
print(f"Using {device} device")
```

```
Using cpu device
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net,self).__init__()
        self.linear1=nn.Linear(8, 6)
        self.linear2=nn.Linear(6,3)
        self.linear3=nn.Linear(3,1)
        self.sigmoid=nn.Sigmoid()

    def forward(self,x):
        x=F.relu(self.linear1(x))
        x=F.relu(self.linear2(x))
        x=self.sigmoid(self.linear3(x))
        return x

criterion = nn.MSELoss()
EPOCHS = 300
net= Net()
optm = Adam(net.parameters(), lr = 0.005)

def train(model, x, y, optimizer, criterion):
    model.zero_grad()
    output = model(x)
    # print(output.shape)
    loss =criterion(output,y.view(-1,1))
    loss.backward()
    optimizer.step()

    return loss, output
```

```
    return loss, output

xt = torch.from_numpy(X_train).float()
yt = torch.from_numpy(y_train).float()

prevloss=0
currloss=0

for epoch in range(EPOCHS):
    epoch_loss = 0
    correct = 0
    loss, predictions = train(net,xt ,yt , optm, criterion)
    for idx, i in enumerate(predictions):
      i = torch.round(i)
      if i == yt[idx]:
          correct += 1
    acc = (correct/len(X_train))
    epoch_loss+=loss
    print('Epoch {} Accuracy : {}'.format(epoch+1, acc*100))
    print('Epoch {} Loss : {}'.format((epoch+1),epoch_loss))

xtest = torch.from_numpy(X_test).float()
ytest = torch.from_numpy(y_test).float()
pred = net(xtest)
pred = torch.round(pred)
predi= pred.detach().numpy()
predi= predi.reshape(-1)
print()
print("Final test accuracy-")
print(sum(predi==y_test)/len(y_test))
```

```
Epoch 101 Accuracy : 81.35048231511254
Epoch 101 Loss : 0.16603296995162964
Epoch 102 Accuracy : 81.02893890675242
Epoch 102 Loss : 0.1657795011997223
Epoch 103 Accuracy : 81.35048231511254
Epoch 103 Loss : 0.1655302792787552
Epoch 104 Accuracy : 81.35048231511254
Epoch 104 Loss : 0.16527988016605377
Epoch 105 Accuracy : 81.35048231511254
Epoch 105 Loss : 0.16502848267555237
Epoch 106 Accuracy : 81.51125401929261
Epoch 106 Loss : 0.16477760672569275
Epoch 107 Accuracy : 81.67202572347267
Epoch 107 Loss : 0.16452568769454956
Epoch 108 Accuracy : 81.67202572347267
Epoch 108 Loss : 0.16427935659885406
Epoch 109 Accuracy : 81.67202572347267
Epoch 109 Loss : 0.1640348583459854
Epoch 110 Accuracy : 81.67202572347267
Epoch 110 Loss : 0.16379638016223907
Epoch 111 Accuracy : 81.83279742765274
Epoch 111 Loss : 0.1635635793209076
Epoch 112 Accuracy : 81.83279742765274
Epoch 112 Loss : 0.16333487629890442
Epoch 113 Accuracy : 81.51125401929261
Epoch 113 Loss : 0.16310863196849823
Epoch 114 Accuracy : 81.35048231511254
```

```
Epoch 114 Loss : 0.1628829389810562
Epoch 115 Accuracy : 81.18971061093248
Epoch 115 Loss : 0.16265663504600525
Epoch 116 Accuracy : 81.18971061093248
Epoch 116 Loss : 0.16243238747119904
Epoch 117 Accuracy : 81.18971061093248
Epoch 117 Loss : 0.16221164166927338
Epoch 118 Accuracy : 81.35048231511254
Epoch 118 Loss : 0.161993145942688
Epoch 119 Accuracy : 81.35048231511254
Epoch 119 Loss : 0.1617765724658966
Epoch 120 Accuracy : 81.35048231511254
Epoch 120 Loss : 0.1615651696920395
Epoch 121 Accuracy : 81.18971061093248
Epoch 121 Loss : 0.1613551676273346
Epoch 122 Accuracy : 81.18971061093248
Epoch 122 Loss : 0.16114738583564758
Epoch 123 Accuracy : 81.35048231511254
Epoch 123 Loss : 0.16094347834587097
Epoch 124 Accuracy : 81.35048231511254
Epoch 124 Loss : 0.16074034571647644
Epoch 125 Accuracy : 81.35048231511254
Epoch 125 Loss : 0.1605377495288849
Epoch 126 Accuracy : 81.51125401929261
Epoch 126 Loss : 0.16033807396888733
Epoch 127 Accuracy : 81.35048231511254
Epoch 127 Loss : 0.16014112532138824
Epoch 128 Accuracy : 81.35048231511254
Epoch 128 Loss : 0.1599450707435608
Epoch 129 Accuracy : 81.35048231511254
Epoch 129 Loss : 0.1597500592470169
Epoch 130 Accuracy : 81.18971061093248
```

Making the prediction with the test dataset

Predicting the values

```
xtest = torch.from_numpy(x_eval).float()
pred = net(xtest)
pred = torch.round(pred)
predictions = pred.detach().numpy()
predi= predi.reshape(-1)
predictions = pd.DataFrame(predictions.astype('int32'))
predictions = predictions.rename(columns={0:"Survived"})
predictions
```

	Survived
0	0
1	0
2	0
3	0
4	0
...	...
413	0
414	1
415	0

Because the df_test's passenger ID column was removed earlier, the df_test dataframe will have to be redefined to its original format.

```
df_test = pd.read_csv("./drive/MyDrive/ML/test.csv")
```

Final submission will be the PassengerId concatenated with the predictions

```
submission = pd.concat([df_test["PassengerId"],predictions],axis=1)
submission
```

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

Almost done! Just need to convert the submission dataframe into a csv file

```
submission.to_csv("./drive/MyDrive/ML/submission.csv",index=False)
```

✓ 0s completed at 21:53