Predict survival on the Titanic

In this Lab, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy

Dataset

The dataset contains 891 observations of 12 variables:

- **PassengerId**: Unique ID for each passenger
- **Survived**: Survival (0 = No; 1 = Yes)
- **Pclass**: Passenger Class (1 = 1st; 2 = 2nd; 3 = 3rd)
- **Name**: Name
- **Sex**: Sex
- **Age**: Age
- **Sibsp**: Number of Siblings/Spouses Aboard
- **Parch**: Number of Parents/Children Aboard
- **Ticket**: Ticket Number
- **Fare**: Passenger Fare
- **Cabin**: Cabin
- **Embarked** Port of Embarkation (C = Cherbourg; Q = Queenstown; S = Southampton)

```
import os
from google.colab import drive
drive.mount('/content/drive', force_remount=False)
```

1
```
# imports
import warnings
warnings.filterwarnings('ignore')
# your code here
import pandas as pd
import numpy as np
```

2
```
titanic = pd.read_csv('titanic.csv')  # your code here
titanic.head()
```

2

	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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450

4
```
# print some info about the dataframe
# your code here
print(titanic.shape)
print(titanic.columns)
```

```
(891, 12)
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

Looks like there are some Nan values, let's see how many for each column

5
```
titanic.isnull().sum()
```

5
```
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
```

Cabin contains a lot of Nan values, we'll drop this column
We'll replace the Nan values in **Age** with the age's median, and the ones in **Embarked** with **'S'**, which is the most frequent one in this column

7
```
# your code here to drop Cabin
titanic.drop('Cabin', axis = 1, inplace = True)
```

```
# check the fillna documentation: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFr
titanic["Age"].fillna(titanic["Age"].median, inplace = True)
titanic["Embarked"].fillna(titanic["Embarked"].mode, inplace = True)
titanic.isnull().sum()
```

7

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

Visualization

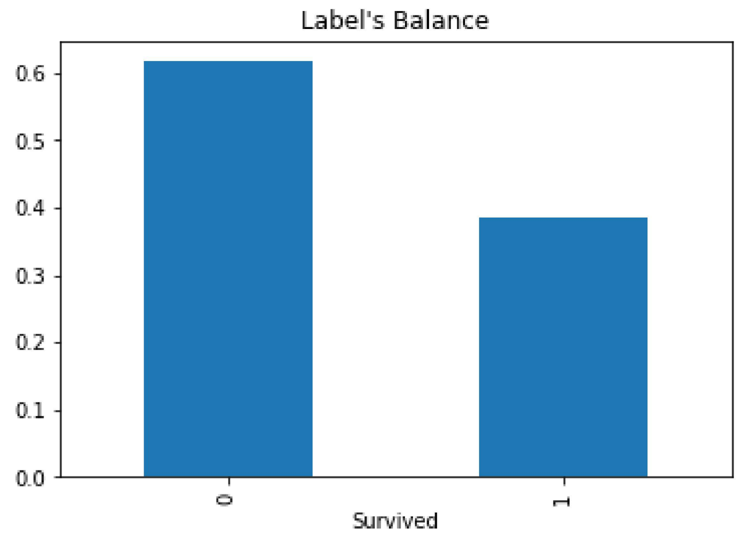
8

```
%matplotlib inline
import matplotlib.pyplot as plt
print ('survival rate =', titanic.Survived.mean())
(titanic.groupby('Survived').size()/titanic.shape[0]).plot(kind="bar",title="Label's Balance")
```

```
survival rate = 0.3838383838383838
```

8

```
<AxesSubplot:title={'center':"Label's Balance"}, xlabel='Survived'>
```



9

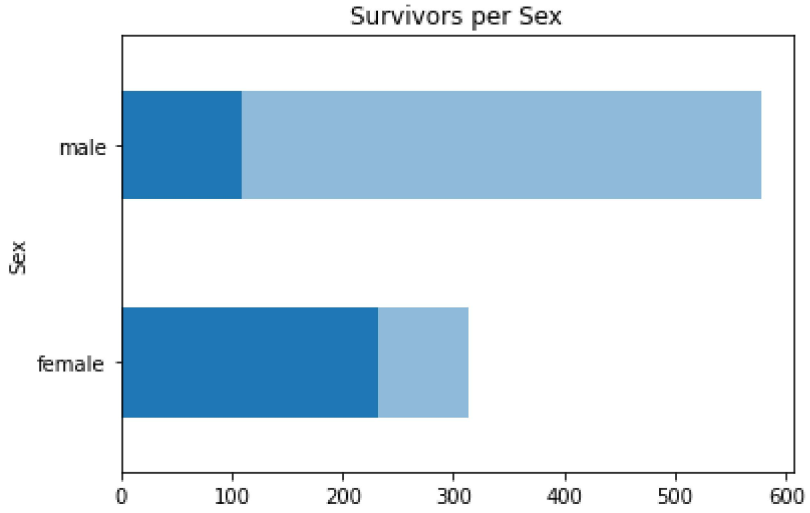
```
# make a function to plot survival against passenger attribute
def survival_rate(column,t):
    df=pd.DataFrame()
    df['total']=titanic.groupby(column).size()
    df['survived'] = titanic.groupby(column).sum()['Survived']
    df['percentage'] = round(df['survived']/df['total']*100,2)
    print(df)

    df['survived'].plot(kind=t)
    df['total'].plot(kind=t,alpha=0.5,title="Survivors per "+str(column))
    plt.show()
```

10

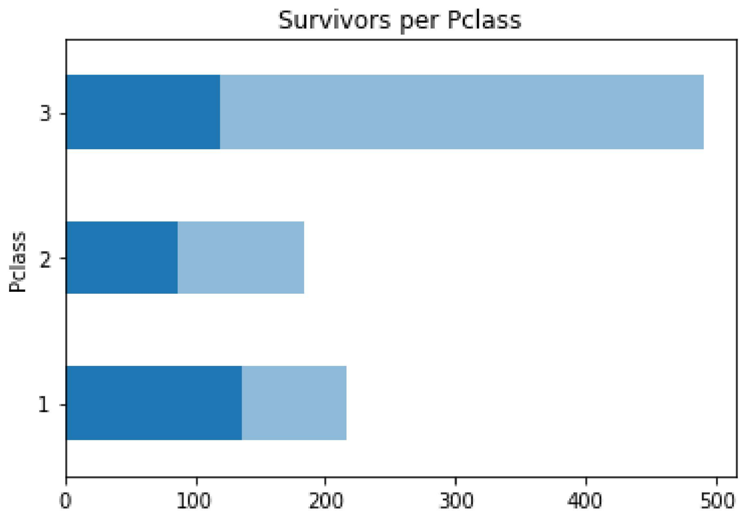
```
# Draw survival per Sex
survival_rate("Sex","barh")
```

```
         total  survived  percentage
Sex
female     314       233       74.20
male       577       109       18.89
```



11
```
# Draw survival per Class
survival_rate("Pclass","barh")
```

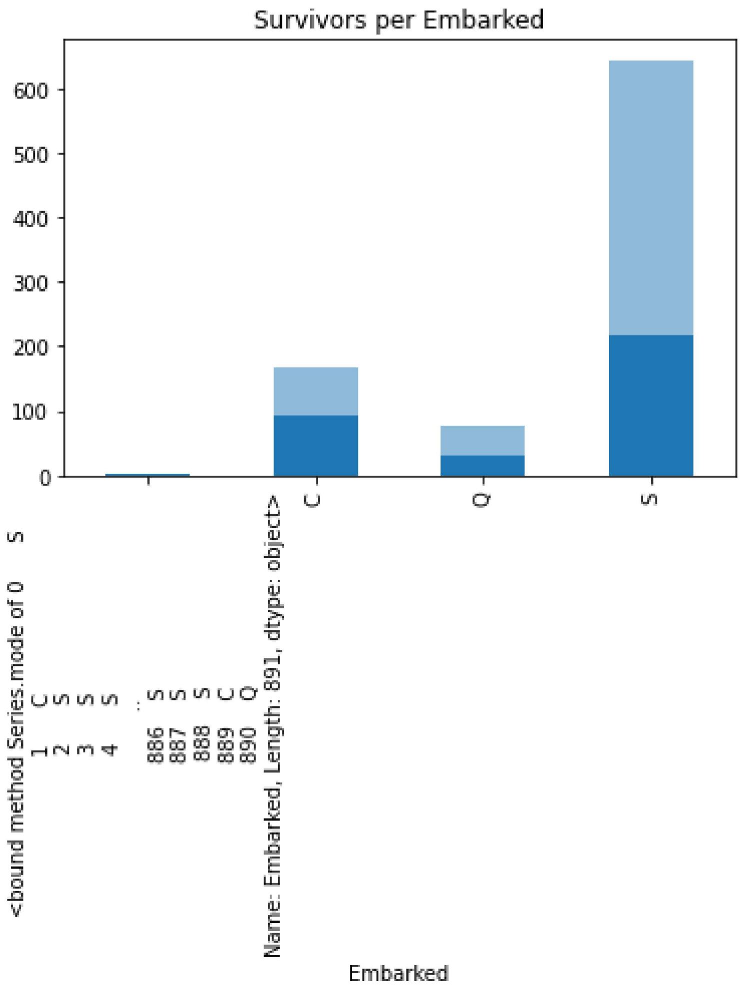
```
         total  survived  percentage
Pclass
1          216       136       62.96
2          184        87       47.28
3          491       119       24.24
```



12
```
# Graph survived per port of embarkation
survival_rate("Embarked","bar")
```

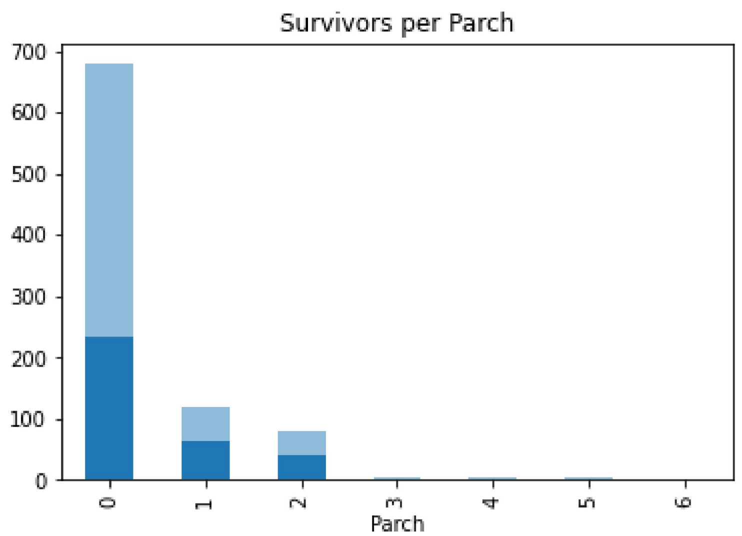
```
                                                     total  survived  \
Embarked
<bound method Series.mode of 0      S\n1      C...       2         2
C                                                      168        93
Q                                                       77        30
S                                                      644       217

                                                     percentage
Embarked
<bound method Series.mode of 0      S\n1      C...      100.00
C                                                        55.36
Q                                                        38.96
S                                                        33.70
```



13
```
# Draw survived per Number of Parents/Children Aboard (Parch)
# your code here
survival_rate('Parch',"bar")
```

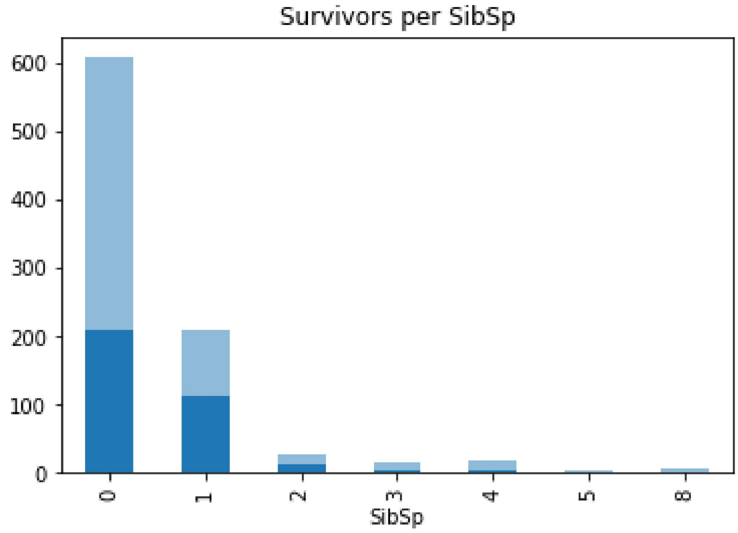
	total	survived	percentage
Parch			
0	678	233	34.37
1	118	65	55.08
2	80	40	50.00
3	5	3	60.00
4	4	0	0.00
5	5	1	20.00
6	1	0	0.00



14
```
# Draw survived per Number of Siblings/Spouses Aboard (SibSp)
```

```
# your code here
survival_rate('SibSp', "bar")
```

```
       total  survived  percentage
SibSp
0        608       210       34.54
1        209       112       53.59
2         28        13       46.43
3         16         4       25.00
4         18         3       16.67
5          5         0        0.00
8          7         0        0.00
```



Model training

Some of the columns don't have predictive power, so let's specify which ones are included for prediction

15
```
predictors = ["Pclass", "Sex", "Age", 'SibSp' ,'Parch', "Fare", "Embarked"]
```

We need now to convert text columns in **predictors** to numerical ones

16
```
for col in predictors: # Loop through all columns in predictors
    if titanic[col].dtype == 'object':  # check if column's type is object (text)
        titanic[col] = pd.Categorical(titanic[col]).codes  # convert text to numerical

titanic.head()
```

16

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

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
2	3	1	3	Heikkinen, Miss. Laina	0	2	0	0	STON/O2. 3101282	7
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	0	3	1	0	113803	5
4	5	0	3	Allen, Mr. William Henry	1	3	0	0	373450	8

17
```
# Split the data into a training set and a testing set. Set: test_size=0.3, random_state=1
# your code here
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(titanic[predictors], titanic['Survived'], test_si

print ("train shape", X_train.shape, y_train.shape)
print ("test shape", X_test.shape, y_test.shape)
```

```
train shape (623, 7) (623,)
test shape (268, 7) (268,)
```

18
```
# import LogisticRegression from: http://scikit-learn.org/stable/modules/generated/sklearn.linear_mod

# your code here
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=1)
# your code here
clf.fit(X_train, y_train)
train_score = clf.score(X_train, y_train)
test_score = clf.score(X_test, y_test)
print ('train accuracy =', train_score)
print ('test accuracy =', test_score)
```

```
train accuracy = 0.8138041733547352
test accuracy = 0.7611940298507462
```

Let's print the model's parameters

19
```
coeff = pd.DataFrame()
coeff['Feature'] = X_train.columns
coeff['Coefficient Estimate'] = pd.Series(clf.coef_[0])
coeff.loc[len(coeff)]=['Intercept',clf.intercept_[0]]
print (coeff)
```

```
     Feature  Coefficient Estimate
0     Pclass             -0.891619
1        Sex             -2.679929
2        Age              0.001337
3      SibSp             -0.237991
4      Parch              0.086617
5       Fare              0.000066
```

```
6   Embarked             -0.230639
7  Intercept              3.698563
```

We now need to predict class labels for the test set. We will also generate the class probabilities

25
```
# predict class labels for the test set
y_pred = clf.predict(X_test)
print (y_pred)
```

```
[1 0 1 1 1 0 0 1 0 1 0 1 0 0 1 0 0 0 0 1 0 0 1 0 1 0 1 1 0 1 1 0 1 1 0 0 0
 0 0 0 1 1 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 0 0 1 0 0 0 0 0
 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0
 0 0 0 0 0 0 0 1 1 1 0 0 0 1 1 1 1 0 0 0 0 1 0 0 1 1 0 0 1 1 0 1 1 0 1 0 0
 1 0 1 0 0 1 0 0 0 0 1 0 0 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1
 0 0 1 0 1 0 0 1 1 1 1 0 1 0 0 0 1 0 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 1 0 0 0
 0 0 0 0 1 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 0 0 0 0 1 1 1 1 0 0 0 0 1 1 0 1 0
 0 1 0 1 1 0 0 0 0]
```

26
```
# generate class probabilities : http://scikit-learn.org/stable/modules/generated/sklearn.linear_mode
y_probs = clf.predict_proba(X_test)
print (y_probs)
```

```
[[0.09811678 0.90188322]
 [0.91231803 0.08768197]
 [0.21933444 0.78066556]
 [0.36162655 0.63837345]
 [0.19261216 0.80738784]
 [0.91026515 0.08973485]
 [0.77042761 0.22957239]
 [0.13186827 0.86813173]
 [0.57773539 0.42226461]
 [0.41804473 0.58195527]
 [0.90872448 0.09127552]
 [0.41702822 0.58297178]
 [0.63524995 0.36475005]
 [0.81004738 0.18995262]
 [0.36163243 0.63836757]
 [0.6319125  0.3680875 ]
 [0.94210346 0.05789654]
 [0.92614495 0.07385505]
 [0.90591444 0.09408556]
 [0.25642063 0.74357937]
 [0.90894583 0.09105417]
 [0.89294928 0.10705072]
 [0.06687551 0.93312449]
 [0.80527616 0.19472384]
 [0.34973759 0.65026241]
 [0.91199266 0.08800734]
 [0.08935272 0.91064728]
 [0.22206506 0.77793494]
 [0.81048368 0.18951632]
 [0.10669091 0.89330909]
 [0.41763687 0.58236313]
 [0.63588507 0.36411493]
 [0.36702898 0.63297102]
 [0.46411658 0.53588342]
 [0.88515013 0.11484987]
 [0.59951235 0.40048765]
 [0.61781714 0.38218286]
 [0.9123211  0.0876789 ]
 [0.57163057 0.42836943]
 [0.91156683 0.08843317]
 [0.41540419 0.58459581]
 [0.37399543 0.62600457]
 [0.11609521 0.88390479]
```

```
[0.61746707 0.38253293]
[0.67744212 0.32255788]
[0.91242935 0.08757065]
[0.96507476 0.03492524]
[0.80839642 0.19160358]
[0.51647164 0.48352836]
[0.8034039  0.1965961 ]
[0.72887161 0.27112839]
[0.22186261 0.77813739]
[0.80696848 0.19303152]
[0.79785643 0.20214357]
[0.90970881 0.09029119]
[0.46232504 0.53767496]
[0.80508751 0.19491249]
[0.61866996 0.38133004]
[0.89425355 0.10574645]
[0.41409452 0.58590548]
[0.80042905 0.19957095]
[0.86773033 0.13226967]
[0.79331219 0.20668781]
[0.62692375 0.37307625]
[0.68784032 0.31215968]
[0.26801155 0.73198845]
[0.88313868 0.11686132]
[0.86794976 0.13205024]
[0.39481521 0.60518479]
[0.81924741 0.18075259]
[0.94083221 0.05916779]
[0.96084102 0.03915898]
[0.51023179 0.48976821]
[0.91231803 0.08768197]
[0.20920844 0.79079156]
[0.75500133 0.24499867]
[0.13187656 0.86812344]
[0.89202967 0.10797033]
[0.52206123 0.47793877]
[0.1025837  0.8974163 ]
[0.91227614 0.08772386]
[0.9831913  0.0168087 ]
[0.91231722 0.08768278]
[0.91114208 0.08885792]
[0.14926199 0.85073801]
[0.9097089  0.0902911 ]
[0.91231722 0.08768278]
[0.86773036 0.13226964]
[0.58271519 0.41728481]
[0.36286472 0.63713528]
[0.62255116 0.37744884]
[0.86771913 0.13228087]
[0.88770702 0.11229298]
[0.36163243 0.63836757]
[0.91230959 0.08769041]
[0.51241357 0.48758643]
[0.90794892 0.09205108]
[0.31026291 0.68973709]
[0.91070073 0.08929927]
[0.91231803 0.08768197]
[0.52389966 0.47610034]
[0.637277   0.362723  ]
[0.88410428 0.11589572]
[0.2864439  0.7135561 ]
[0.06210218 0.93789782]
[0.90827272 0.09172728]
[0.90971664 0.09028336]
[0.90971759 0.09028241]
[0.95992558 0.04007442]
[0.91036994 0.08963006]
[0.57487679 0.42512321]
[0.86834283 0.13165717]
```

```
[0.79479445 0.20520555]
[0.92663807 0.07336193]
[0.80232    0.19768   ]
[0.90839197 0.09160803]
[0.83965996 0.16034004]
[0.68584322 0.31415678]
[0.25540166 0.74459834]
[0.41312155 0.58687845]
[0.26700856 0.73299144]
[0.65055338 0.34944662]
[0.91231803 0.08768197]
[0.80591492 0.19408508]
[0.08965769 0.91034231]
[0.10504364 0.89495636]
[0.22322184 0.77677816]
[0.41768074 0.58231926]
[0.9831913  0.0168087 ]
[0.89202967 0.10797033]
[0.90816858 0.09183142]
[0.90648242 0.09351758]
[0.37568559 0.62431441]
[0.52228991 0.47771009]
[0.58119203 0.41880797]
[0.06918274 0.93081726]
[0.0890262  0.9109738 ]
[0.96388418 0.03611582]
[0.92781633 0.07218367]
[0.40441709 0.59558291]
[0.10815755 0.89184245]
[0.91137246 0.08862754]
[0.1121606  0.8878394 ]
[0.41637639 0.58362361]
[0.91231867 0.08768133]
[0.4682795  0.5317205 ]
[0.80800748 0.19199252]
[0.91231722 0.08768278]
[0.26630092 0.73369908]
[0.91274482 0.08725518]
[0.39478652 0.60521348]
[0.90986981 0.09013019]
[0.95133511 0.04866489]
[0.36286773 0.63713227]
[0.91206195 0.08793805]
[0.90894345 0.09105655]
[0.91274821 0.08725179]
[0.91037591 0.08962409]
[0.18893195 0.81106805]
[0.80232    0.19768   ]
[0.83634896 0.16365104]
[0.91090935 0.08909065]
[0.41022417 0.58977583]
[0.63177554 0.36822446]
[0.94379359 0.05620641]
[0.35952727 0.64047273]
[0.29070456 0.70929544]
[0.91206195 0.08793805]
[0.90883446 0.09116554]
[0.4774079  0.5225921 ]
[0.06824165 0.93175835]
[0.60751059 0.39248941]
[0.1022151  0.8977849 ]
[0.92178953 0.07821047]
[0.10406749 0.89593251]
[0.6363495  0.3636505 ]
[0.89202978 0.10797022]
[0.88955912 0.11044088]
[0.39021453 0.60978547]
[0.92999448 0.07000552]
[0.06709562 0.93290438]
```

```
[0.22371432 0.77628568]
[0.86954264 0.13045736]
[0.79720887 0.20279113]
[0.1028124  0.8971876 ]
[0.80589948 0.19410052]
[0.91231722 0.08768278]
[0.08924401 0.91075599]
[0.89202967 0.10797033]
[0.06572299 0.93427701]
[0.63558801 0.36441199]
[0.62678603 0.37321397]
[0.36163243 0.63836757]
[0.30684061 0.69315939]
[0.32911089 0.67088911]
[0.05659667 0.94340333]
[0.81017893 0.18982107]
[0.22261058 0.77738942]
[0.90360997 0.09639003]
[0.51005186 0.48994814]
[0.90906076 0.09093924]
[0.14846394 0.85153606]
[0.86773036 0.13226964]
[0.71898759 0.28101241]
[0.65730943 0.34269057]
[0.21748083 0.78251917]
[0.12407981 0.87592019]
[0.80316668 0.19683332]
[0.63594523 0.36405477]
[0.94059352 0.05940648]
[0.06735104 0.93264896]
[0.95741679 0.04258321]
[0.86369003 0.13630997]
[0.86328525 0.13671475]
[0.83835159 0.16164841]
[0.9124241  0.0875759 ]
[0.89203532 0.10796468]
[0.08386708 0.91613292]
[0.18972105 0.81027895]
[0.61844413 0.38155587]
[0.63589268 0.36410732]
[0.86773036 0.13226964]
[0.61617219 0.38382781]
[0.68353449 0.31646551]
[0.91286455 0.08713545]
[0.57435943 0.42564057]
[0.08551223 0.91448777]
[0.6843857  0.3156143 ]
[0.26969771 0.73030229]
[0.68904    0.31096   ]
[0.83176145 0.16823855]
[0.88654616 0.11345384]
[0.84352879 0.15647121]
[0.80777449 0.19222551]
[0.9126383  0.0873617 ]
[0.36322921 0.63677079]
[0.43194415 0.56805585]
[0.32267179 0.67732821]
[0.46944939 0.53055061]
[0.92993247 0.07006753]
[0.81892504 0.18107496]
[0.85696658 0.14303342]
[0.86226747 0.13773253]
[0.90868104 0.09131896]
[0.89882778 0.10117222]
[0.91124264 0.08875736]
[0.22071086 0.77928914]
[0.22368576 0.77631424]
[0.38757763 0.61242237]
[0.10274715 0.89725285]
```

```
 [0.90546205 0.09453795]
 [0.88890066 0.11109934]
 [0.67146669 0.32853331]
 [0.64083442 0.35916558]
 [0.09751839 0.90248161]
 [0.35549868 0.64450132]
 [0.67439434 0.32560566]
 [0.22249596 0.77750404]
 [0.83025735 0.16974265]
 [0.53434596 0.46565404]
 [0.19598724 0.80401276]
 [0.91286455 0.08713545]
 [0.36163243 0.63836757]
 [0.18942653 0.81057347]
 [0.8092976  0.1907024 ]
 [0.80253948 0.19746052]
 [0.57105641 0.42894359]
 [0.67729915 0.32270085]]
```

As you can see, the classifier outputs two probabilities for each row. It's predicting a 1 (Survived) any time the probability in the second column is greater than 0.5. Let's visualize it all together.

27
```
pred = pd.DataFrame({
        "Survived_original": y_test,
        "Survived_predicted": y_pred,
        "Survived_proba": np.transpose(y_probs)[1]
        })
pred["Comparison"]= pred.Survived_original ==pred.Survived_predicted
pred.head()
```

27

	Survived_original	Survived_predicted	Survived_proba	Comparison
862	1	1	0.901883	True
223	0	0	0.087682	True
84	1	1	0.780666	True
680	0	1	0.638373	False
535	1	1	0.807388	True

Confusion matrix

28
```
from sklearn import metrics
print (metrics.confusion_matrix(y_test, y_pred))
print (metrics.classification_report(y_test, y_pred))
```

```
[[132  21]
 [ 43  72]]
              precision    recall  f1-score   support

           0       0.75      0.86      0.80       153
           1       0.77      0.63      0.69       115

    accuracy                           0.76       268
   macro avg       0.76      0.74      0.75       268
weighted avg       0.76      0.76      0.76       268
```

As you can see, we can have the classification report for each class

K-Fold Cross Validation

31

```
# import cross_validation from: http://scikit-learn.org/stable/modules/generated/sklearn.model_select
# your code here
from sklearn.model_selection import cross_val_score
clf = LogisticRegression(random_state=1)
scores = cross_val_score(clf, titanic[predictors], titanic["Survived"], scoring='accuracy', cv=5)
## see model
print(scores)
# Take the mean of the scores (because we have one for each fold)
print(scores.mean())
```

```
[0.77653631 0.79775281 0.78089888 0.7752809  0.80337079]
0.7867679367271357
```

When you are improving a model, you want to make sur that you are really doing it and not just being lucky. This is why it's good to work with cross validation instead of one train/test split.