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)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, ca
```

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

```
titanic = pd.read_csv('titanic.csv', index_col=0)
titanic.head()
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket
PassengerId								
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599
			Heikkinen,					STON/O2.

```
print(titanic.dtypes.value_counts())
print(titanic.shape)
```

```
object     5
int64      4
float64    2
dtype: int64
(891, 11)
```

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

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

```
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

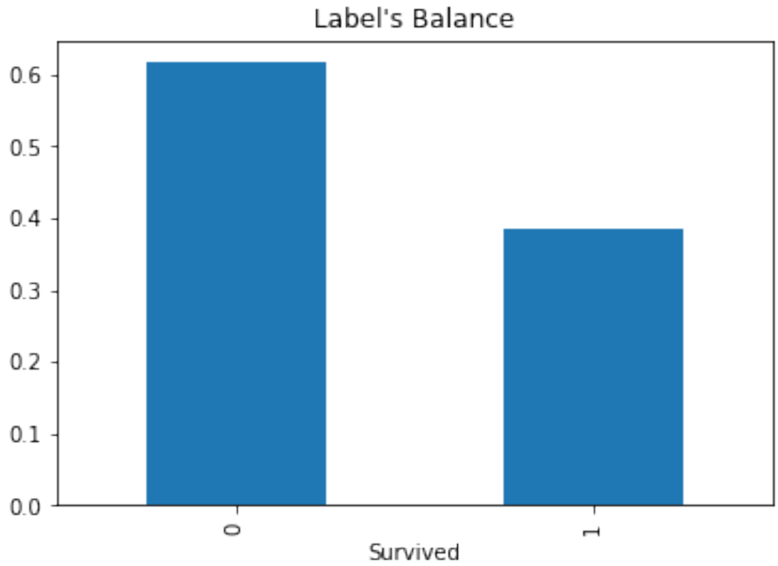
```
to_drop =['Cabin']
titanic.drop(to_drop, axis = 1, inplace = True)

# check the fillna documentation: http://pandas.pydata.org/pandas-docs/stable/ge

titanic["Age"]= titanic["Age"].fillna(titanic["Age"].median())
titanic["Embarked"].fillna("s")
print(titanic.isnull().sum())
print(titanic["Age"])
```

```
Survived    0
Pclass      0
Name        0
Sex         0
Age         0
SibSp       0
Parch       0
Ticket      0
Fare        0
Embarked    2
dtype: int64
PassengerId
1      22.0
2      38.0
3      26.0
4      35.0
5      35.0
       ...
887    27.0
888    19.0
889    28.0
890    26.0
891    32.0
Name: Age, Length: 891, dtype: float64
```

Visualization

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

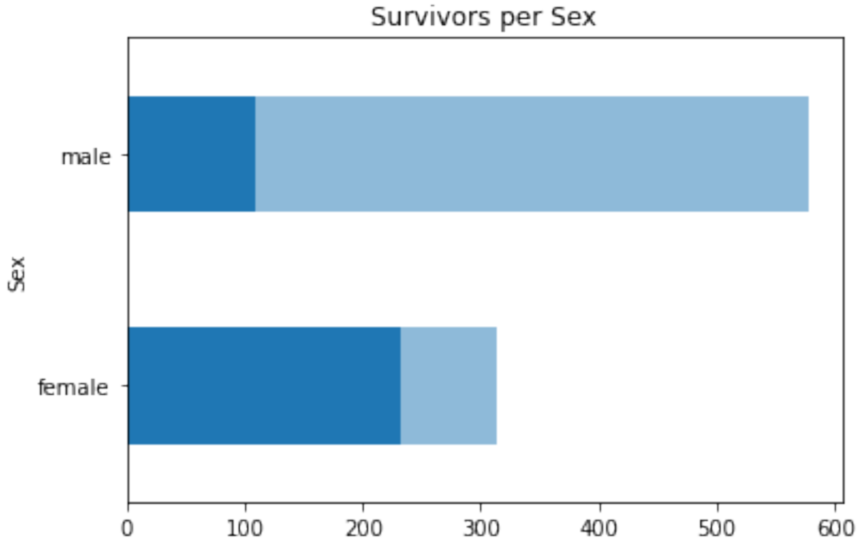
```
survival rate = 0.3838383838383838
<matplotlib.axes._subplots.AxesSubplot at 0x7fc7634bd710>
```




```
# 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()
```

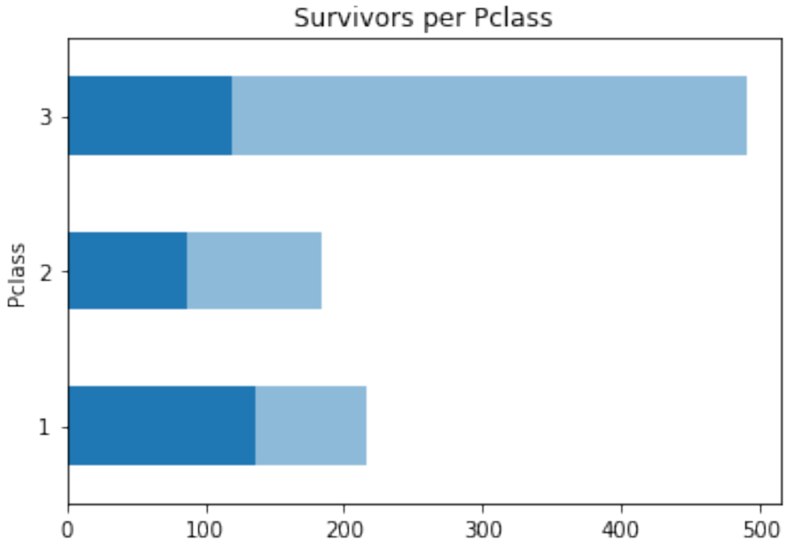
```
# Draw survival per Sex
survival_rate("Sex","barh")
```

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



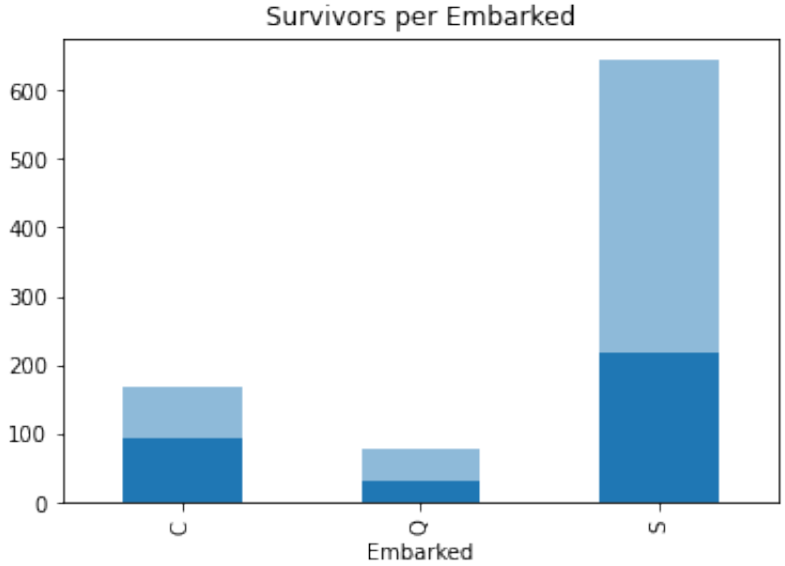

```
# Draw survival per Class
survival_rate("Pclass","barh")
```

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



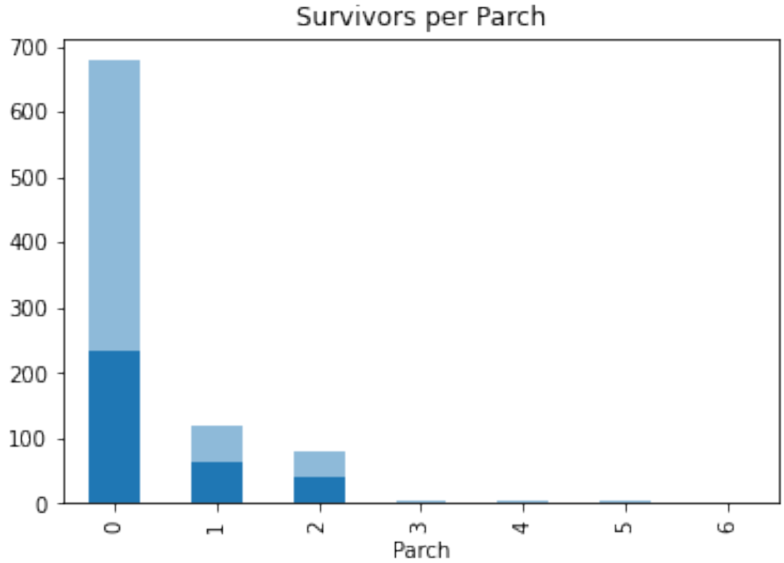
```
# Graph survived per port of embarkation
survival_rate("Embarked","bar")
```

	total	survived	percentage
Embarked			
C	168	93	55.36
Q	77	30	38.96
S	644	217	33.70



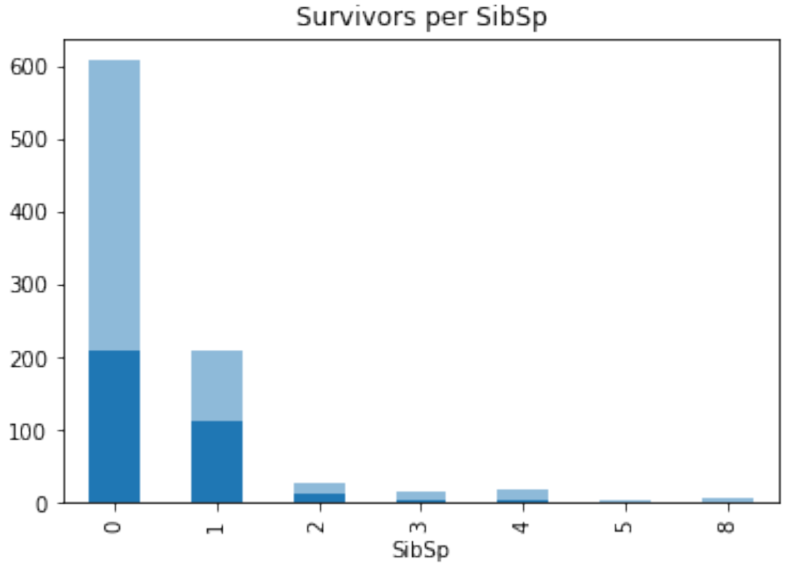
```
# Draw survived per Number of Parents/Children Aboard (Parch)

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
```



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

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

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

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

```
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 num

titanic.head()
```

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

```
# Split the data into a training set and a testing set. Set: test_size=0.3, rand
from sklearn.model_selection import train_test_split

#Y = titanic[["Survived"]]
Y = titanic.Survived
X = titanic[predictors]
print(X.describe())
print(Y.describe())

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.3, random_
print ("train shape", X_train.shape, y_train.shape)
print ("test shape", X_test.shape, y_test.shape)
```

```
           Pclass         Sex         Age  ...       Parch        Fare    E
count  891.000000  891.000000  891.000000  ...  891.000000  891.000000  891
mean     2.308642    0.647587   29.361582  ...    0.381594   32.204208    1
std      0.836071    0.477990   13.019697  ...    0.806057   49.693429    0
min      1.000000    0.000000    0.420000  ...    0.000000    0.000000   -1
25%      2.000000    0.000000   22.000000  ...    0.000000    7.910400    1
50%      3.000000    1.000000   28.000000  ...    0.000000   14.454200    2
75%      3.000000    1.000000   35.000000  ...    0.000000   31.000000    2
max      3.000000    1.000000   80.000000  ...    6.000000  512.329200    2

[8 rows x 7 columns]
count    891.000000
mean       0.383838
std        0.486592
min        0.000000
25%        0.000000
50%        0.000000
75%        1.000000
max        1.000000
Name: Survived, dtype: float64
train shape (623, 7) (623,)
test shape (268, 7) (268,)
```

```
# import LogisticRegression from: http://scikit-learn.org/stable/modules/generat
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(random_state=1)
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 = 0.8073836276083467
test accuracy = 0.7723880597014925
```

Let's print the model's parameters

```
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             -1.157723
1        Sex             -2.704421
2        Age             -0.040836
3      SibSp             -0.333083
4      Parch              0.073972
5       Fare             -0.000618
6   Embarked             -0.232788
7  Intercept              5.406444
```

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

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

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

```
# generate class probabilities : http://scikit-learn.org/stable/modules/generate
```

```
y_probs = clf.predict_proba(X_test)
print (y_probs)
```

```
 [0.96828452 0.03171548]
 [0.0524113  0.9475887 ]
 [0.93110524 0.06889476]
 [0.87429088 0.12570912]
 [0.77416116 0.22583884]
 [0.85003936 0.14996064]
 [0.93529114 0.06470886]
 [0.8958166  0.1041834 ]
 [0.04093025 0.95906975]
 [0.19628117 0.80371883]
 [0.77285589 0.22714411]
 [0.52079611 0.47920389]
 [0.87202932 0.12797068]
 [0.74337907 0.25662093]
 [0.7313774  0.2686226 ]
 [0.92343845 0.07656155]
 [0.69948351 0.30051649]
 [0.04379551 0.95620449]
 [0.51576323 0.48423677]
 [0.23563124 0.76436876]
 [0.70303091 0.29696909]
 [0.60727152 0.39272848]
 [0.83218774 0.16781226]
 [0.67251703 0.32748297]
 [0.81388601 0.18611399]
 [0.94231663 0.05768337]
 [0.38987367 0.61012633]
 [0.46887738 0.53112262]
 [0.3317613  0.6682387 ]
 [0.28303143 0.71696857]
 [0.92220426 0.07779574]
 [0.9433414  0.0566586 ]
 [0.8344545  0.1655455 ]
```

```
 [0.8344545  0.1655455 ]
 [0.88086992 0.11913008]
 [0.92945875 0.07054125]
 [0.95117657 0.04882343]
 [0.9246686  0.0753314 ]
 [0.19236969 0.80763031]
 [0.22636357 0.77363643]
 [0.65132121 0.34867879]
 [0.09282531 0.90717469]
 [0.94860267 0.05139733]
 [0.89961646 0.10038354]
 [0.6842142  0.3157858 ]
 [0.40849398 0.59150602]
 [0.19881062 0.80118938]
 [0.27662503 0.72337497]
 [0.76978265 0.23021735]
 [0.2724469  0.7275531 ]
 [0.83483053 0.16516947]
 [0.44235294 0.55764706]
 [0.10372827 0.89627173]
 [0.92343845 0.07656155]
 [0.36533412 0.63466588]
 [0.14376858 0.85623142]
 [0.76810584 0.23189416]
 [0.80755126 0.19244874]
 [0.46654909 0.53345091]
 [0.7632605  0.2367395 ]]
```

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.

```
import numpy as np

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()
```

PassengerId	Survived_original	Survived_predicted	Survived_proba	Comparis
863	1	1	0.858997	T
224	0	0	0.084323	T
85	1	1	0.872664	T
681	0	1	0.634610	Fa
536	1	1	0.922104	T

Confusion matrix

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

```
[[129  24]
 [ 37  78]]
              precision    recall  f1-score   support

           0       0.78      0.84      0.81       153
           1       0.76      0.68      0.72       115

    accuracy                           0.77       268
   macro avg       0.77      0.76      0.76       268
weighted avg       0.77      0.77      0.77       268
```

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

K-Fold Cross Validation

```
# import cross_validation from: http://scikit-learn.org/stable/modules/generated
#from sklearn.model_selection import cross_validation
from sklearn.model_selection import cross_val_score

clf = LogisticRegression(random_state=1)
scores = cross_val_score(clf, titanic[predictors], titanic["Survived"], scoring=
## see model
print(scores)
# Take the mean of the scores (because we have one for each fold)
print(scores.mean())
```

```
[0.77653631 0.78651685 0.78089888 0.76966292 0.82022472]
0.7867679367271359
```

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.

0 s terminée à 17:08