Case Study

Titanic: Machine Learning From Disaster

Description

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

In this challenge, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we ask you to apply the tools of machine learning to predict which passengers survived the tragedy.

Overview

The data has been provided as a single training file:

- training set (train.csv)

The **training set** should be used to build your machine learning models. For the training set, we provide the outcome (also known as the “ground truth”) for each passenger. Your model will be based on “features” like passengers’ gender and class.

Data Dictionary

Variable	Definition	Key
PassengerId	Unique Identifier of each passenger	
Survived	Survival	0 = No, 1 = Yes
Pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
Name	Name of Passenger	
Sex	Gender	
Age	Age in years	
SibSp	# of siblings / spouses aboard the Titanic	
Parch	# of parents / children aboard the Titanic	
Ticket	Ticket number	
Fare	Passenger fare	
Cabin	Cabin number	
Embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

- Detail of some of the variables is provided. You need to explore rest of the varibles yourself.

Import Libraries

In [1]:
```
%config IPCompleter.greedy=True
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets
from sklearn import metrics
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
```

Read Data

In [2]:
```
df=pd.read_csv("train.csv")
```

Dimensions of Data

In [3]:
```
print(df.shape)
print("num of columns="+ str(df.shape[1]))
print("num of rows="+ str(df.shape[0]))
```

```
(891, 12)
num of columns=12
num of rows=891
```

In [ ]:
```
```

Peak at the Data

In [4]:
```
df.head(10)
```

Out[4]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabi
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	Na
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C8
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	Na
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C12
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	Na
5	6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	Na
6	7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E4
7	8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	Na
8	9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	Na
9	10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	Na

In [5]:
```
df.tail(10)
```

Out[5]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
881	882	0	3	Markun, Mr. Johann	male	33.0	0	0	349257	7.8958
882	883	0	3	Dahlberg, Miss. Gerda Ulrika	female	22.0	0	0	7552	10.5167
883	884	0	2	Banfield, Mr. Frederick James	male	28.0	0	0	C.A./SOTON 34068	10.5000
884	885	0	3	Sutehall, Mr. Henry Jr	male	25.0	0	0	SOTON/OQ 392076	7.0500
885	886	0	3	Rice, Mrs. William (Margaret Norton)	female	39.0	0	5	382652	29.1250
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13.0000
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30.0000
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23.4500
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30.0000
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7500

Attributes in Data

In [6]:
```
df.columns
```

Out[6]:
```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

Data types of Attributes

In [7]:
```
print(df.dtypes)
print()
print("Discrete Variables")
print("Passenger Id, Survived, Pclass, Name, Sex, SibSp, Parch, Ticket, Cabin, Em
print()
print("Continuous Variables")
print("Age, Fare")
```

```
PassengerId      int64
Survived         int64
Pclass           int64
Name            object
Sex             object
Age            float64
SibSp            int64
Parch            int64
Ticket          object
Fare           float64
Cabin           object
Embarked        object
dtype: object

Discrete Variables
Passenger Id, Survived, Pclass, Name, Sex, SibSp, Parch, Ticket, Cabin, Embarke
d

Continuous Variables
Age, Fare
```

Describe the Data

In [8]:
```
df.describe()

###
### Print the description of data
### You should know what it means to describe a continuous attribute
### And what it means to describe a discrete attribute
### Make separate blocks for each of these
### Write your code here
###
```

Out[8]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [9]:
```
survivalPercentage=df.Survived.mean() * 100
print("percentage of passengers survived=" + str(round(survivalPercentage,2)))
```

```
percentage of passengers survived=38.38
```

In [10]:
```
pclass1= df["Pclass"]==1
pclass2= df["Pclass"]==2
pclass3= df["Pclass"]==3
df.Pclass.isnull().any()

pclass1Percent=pclass1.sum()/len(df)
pclass1Percent*=100
print("Percentage of customers bought 1st class="+str(round(pclass1Percent,2)))

pclass2Percent=pclass2.sum()/len(df)
pclass2Percent*=100
print("Percentage of customers bought 2nd class="+str(round(pclass2Percent,2)))

pclass3Percent=pclass3.sum()/len(df)
pclass3Percent*=100
print("Percentage of customers bought 3rd class="+str(round(pclass3Percent,2)))


pclass1Sur=(df["Pclass"]==1) & (df["Survived"]==1)
pclass2Sur=(df["Pclass"]==2) & (df["Survived"]==1)
pclass3Sur=(df["Pclass"]==3) & (df["Survived"]==1)

pclass1SurPer= pclass1Sur.sum()/pclass1.sum()
pclass2SurPer= pclass2Sur.sum()/pclass2.sum()
pclass3SurPer= pclass3Sur.sum()/pclass3.sum()

print()

print("percentage of pclass1 passengers survived="+ str(round(pclass1SurPer,2)))
print("percentage of pclass2 passengers survived="+ str(round(pclass2SurPer,2)))
print("percentage of pclass3 passengers survived="+ str(round(pclass3SurPer,2)))

print()
#pclass1Sur=pclass1Sur["Survived"]>1
pclassCorr= df['Pclass'].corr(df['Survived'])
print("corelation between passenger class and survival="+ str(round(pclassCorr,2)
print("means higher the passenger class number the less likely you will survive")
```

```
Percentage of customers bought 1st class=24.24
Percentage of customers bought 2nd class=20.65
Percentage of customers bought 3rd class=55.11

percentage of pclass1 passengers survived=0.63
percentage of pclass2 passengers survived=0.47
percentage of pclass3 passengers survived=0.24

corelation between passenger class and survival=-0.34
means higher the passenger class number the less likely you will survive
```

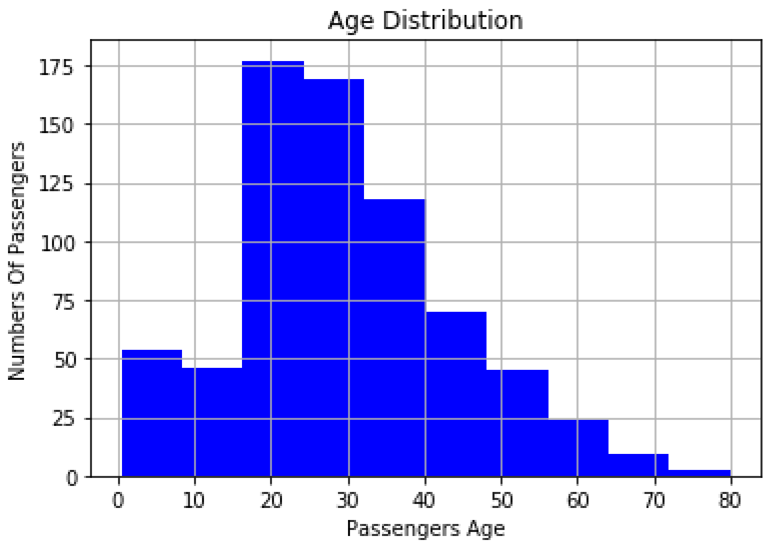
In [11]:
```
#malefilter= df['Sex']=="male"
df['Sex'] = df['Sex'].map({'female': 1, 'male': 0})

maleFilter=df['Sex']==0
femaleFilter=df['Sex']==1

print("num of male passengers=" + str(maleFilter.sum()))
print("num of male passengers=" + str(femaleFilter.sum()))



#df['Sex'].corr(df['Survived'])
```

```
num of male passengers=577
num of male passengers=314
```

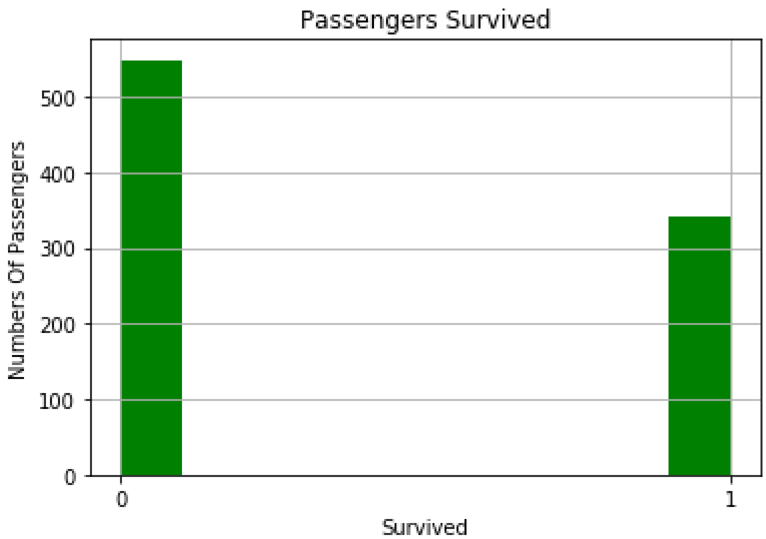
Exploratory Data Analysis

In [12]:
```
###
### For this section you should plot 2 graphs for each attribute
### First graph should display the distribution of data in each attribute
### Second graph should display how the attribute changes with respect to target
###
### Take advantage of matplotlib and any other plotting library
### Remember your graph should be meaningful and properly labeled
### You should understand that when the type of variable changes - it's represent
###
### In some of the attributes you would discover that plotting them is not possib
### Could you tell why plotting these attributes is not possible?
### Is there any way to resolve this issue?
###
### Please remember not to make any changes in the original dataframe
###
### Write your code here
###
```

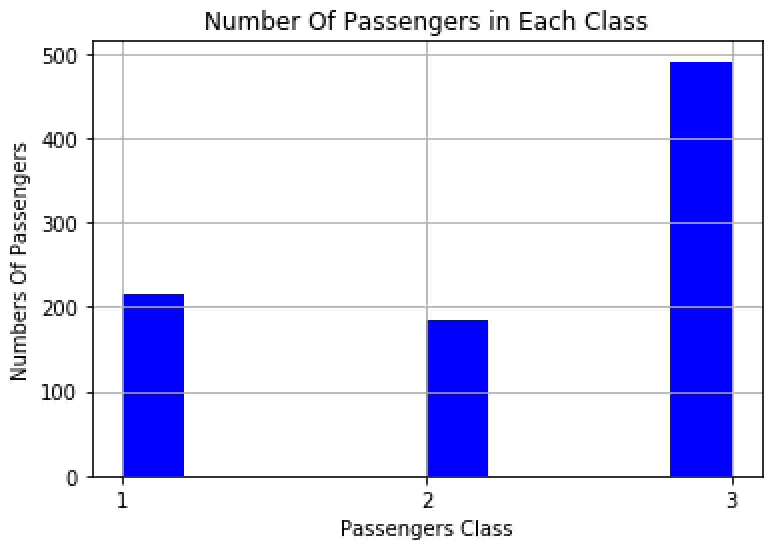
In [13]:
```
age=df["Age"]
age=age.dropna()
plt.hist(age,10, facecolor='blue')

plt.title('Age Distribution')

plt.xlabel('Passengers Age')
plt.ylabel('Numbers Of Passengers')
plt.grid(True)

plt.show()
```



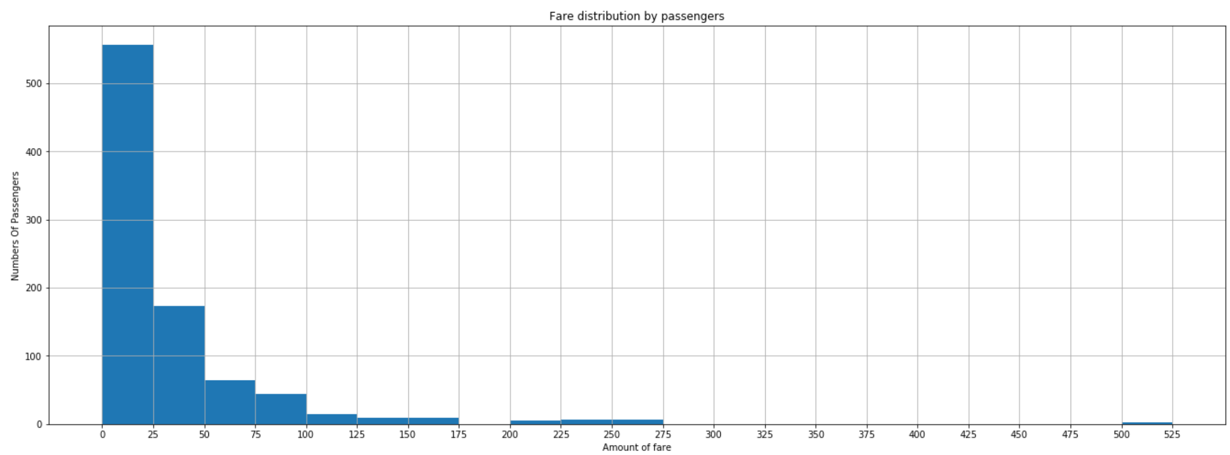
In [14]:
```
survived=df["Survived"]
survived=survived.dropna()
plt.hist(survived, facecolor='green')

plt.title('Passengers Survived')

plt.xticks(range(0, 2))
plt.xlabel('Survived')
plt.ylabel('Numbers Of Passengers')
plt.grid(True)

plt.grid(True)

plt.show()
```



In [15]:
```
pclassPass = df['Pclass']
plt.xlabel('Passengers Class')
plt.ylabel('Numbers Of Passengers')
plt.grid(True)
plt.title('Number Of Passengers in Each Class')
plt.xticks(range(0, 4))
plt.hist(pclassPass, facecolor='blue');
plt.show()
```

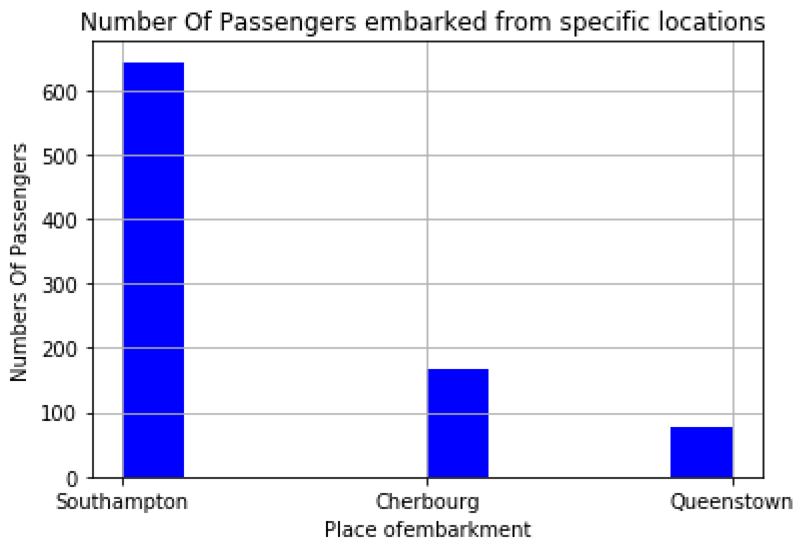


In [16]:
```
fares=df["Fare"]

fig = plt.figure(figsize=(23,8))
ax = fig.add_subplot(1, 1, 1)

# Major ticks every 20, minor ticks every 5
major_ticks = np.arange(0, 550, 25)


ax.set_xticks(major_ticks)





plt.xlabel('Amount of fare')
plt.ylabel('Numbers Of Passengers')
plt.grid(True)
plt.title('Fare distribution by passengers')




binwidth= int(25)

plt.hist(fares, bins=range( int(min(fares)), int(max(fares)) + binwidth, binwidth

# plt.hist(fares,bins=[0,50,100,150,200,250,300,350,400,450,500,550], facecolor='

plt.show()
```

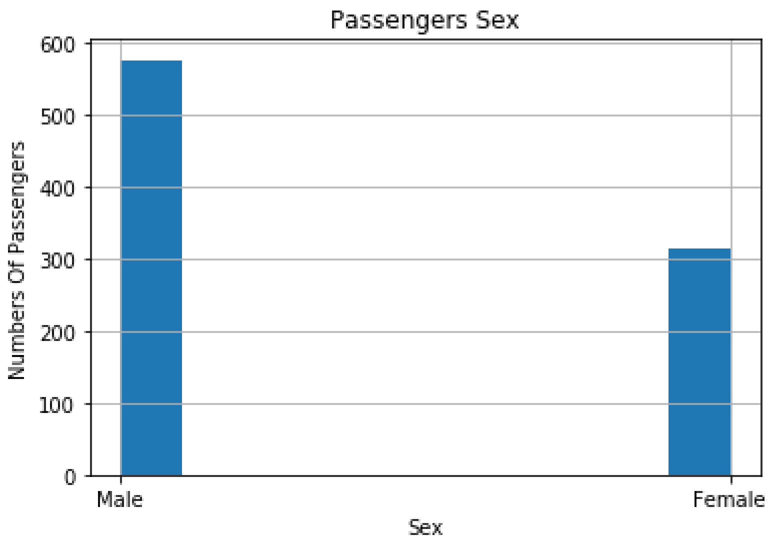


In [17]:

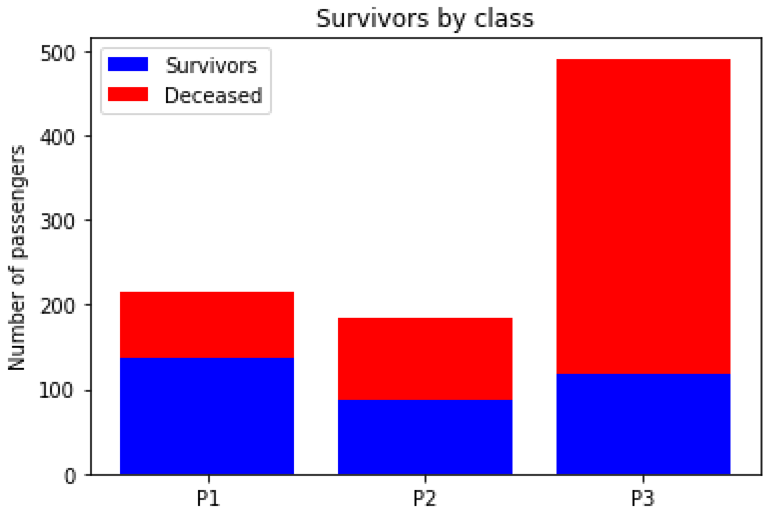
```
Embarks = df['Embarked'].map({'S': 0, 'C': 1,'Q':2})


Embarks= Embarks.dropna()
plt.xticks(range(0, 3),('Southampton', 'Cherbourg','Queenstown'))

plt.xlabel('Place ofembarkment')
plt.ylabel('Numbers Of Passengers')
plt.grid(True)
plt.title('Number Of Passengers embarked from specific locations ')

plt.hist(Embarks, facecolor='blue');
plt.show()
```



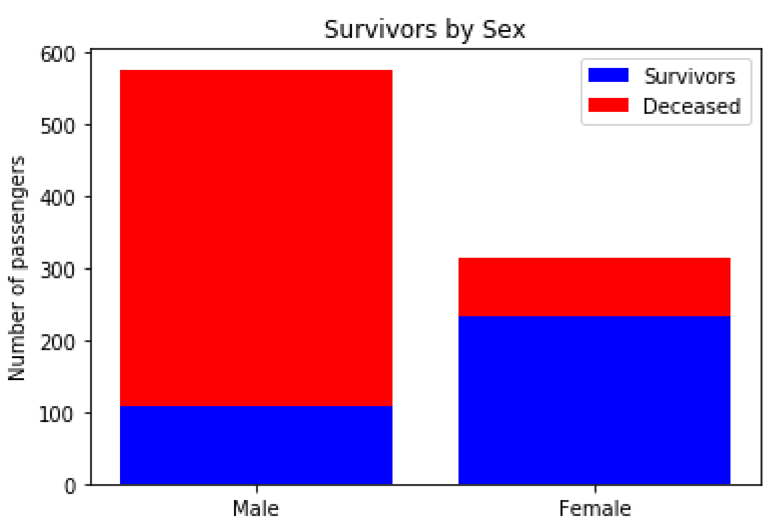
In [18]:
```
sex = df["Sex"]

plt.hist(sex)
plt.title('Passengers Sex')
plt.xticks(range(0, 2),('Male', 'Female'))
plt.xlabel('Sex')
plt.grid(True);
plt.ylabel('Numbers Of Passengers')
plt.show()
```



In [19]:
```
pclass1= df["Pclass"]==1
pclass2= df["Pclass"]==2
pclass3= df["Pclass"]==3

pclass1=pclass1.sum()
pclass2=pclass2.sum()
pclass3=pclass3.sum()



pclass1Sur=(df["Pclass"]==1) & (df["Survived"]==1)
pclass2Sur=(df["Pclass"]==2) & (df["Survived"]==1)
pclass3Sur=(df["Pclass"]==3) & (df["Survived"]==1)


pclass1Sur=pclass1Sur.sum()
pclass2Sur=pclass2Sur.sum()
pclass3Sur=pclass3Sur.sum()

pclass1NotSur=pclass1-pclass1Sur
pclass2NotSur=pclass2-pclass2Sur
pclass3NotSur=pclass3-pclass3Sur


A=[pclass1Sur,pclass2Sur,pclass3Sur]
B=[pclass1NotSur,pclass2NotSur,pclass3NotSur]

X=np.arange(3)

plt.ylabel('Number of passengers')

plt.title('Survivors by class')

plt.xticks(X, ('P1', 'P2', 'P3'))

p1= plt.bar(X, A, color = 'b')
p2=plt.bar(X, B, color = 'r', bottom = A)

plt.legend((p1[0], p2[0]), ('Survivors', 'Deceased'))

plt.show()
```



In [20]:
```
gendersurvived = df[['Sex','Survived']]
fs = len(gendersurvived[(gendersurvived['Sex'] == 1) & (gendersurvived['Survived'
fnots = len(gendersurvived[(gendersurvived['Sex'] == 1) & (gendersurvived['Surviv
ms = len(gendersurvived[(gendersurvived['Sex'] == 0) & (gendersurvived['Survived'
mnots =len(gendersurvived[(gendersurvived['Sex'] == 0) & (gendersurvived['Survive

A = [ms,fs]
B = [mnots, fnots]
X=np.arange(2)
plt.ylabel('Number of passengers')
plt.title('Survivors by Sex')
plt.xticks(X, ('Male', 'Female'))
p1= plt.bar(X, A, color = 'b')
p2=plt.bar(X, B, color = 'r', bottom = A)

plt.legend((p1[0], p2[0]), ('Survivors', 'Deceased'))

plt.show()
```



Pre-processing

In [21]:
```
###
### Check for duplicate rows. If found handle them.
### Write your code here
###



df.head()
# tickets=df['Ticket']

# aa=tickets.duplicated()

dups=df.duplicated(subset=['Name'], keep=False)

df[dups]
```

Out[21]:

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked

In [22]:
```
df['Embarked']=df['Embarked'].map({'S': 0, 'C': 1,'Q':2})
# # df["Embarked"].unique()
```

In [ ]:

In [23]:
```
###
### Check for missing values. If found handle them in the best possible manner
### Write your code here
###

df["Sex"].isna().any()
df["Survived"].isna().any()

df['Pclass'].isnull().sum()


df["Age"].isna().any()
df["Age"].isna().sum()

meanAge = df['Age'].mean()
df['Age'] = df['Age'].fillna(meanAge)


df["Embarked"].isna().sum()

df = df[pd.notnull(df['Embarked'])]

df['SibSp'].isnull().sum()

df['Parch'].isnull().sum()

df['Fare'].isnull().sum()
```

Out[23]: 0

In [24]:
```
###
### Check for outliers. If found handle them in the best possible manner
### Write your code here
###
```

In [25]:
```
###
### Check for any other data quality issues
### Report these issues and also resolve them
### Write your code here
###
```

Feature Selection

In [26]:
```
###
### Do you think that all features are important in this scenario?
###
### Try to identify imoprtant features.
### Give detail of the technique you apply.
### You can search online for different ways.
###
### Write your code here
###


ModelColumns = ['Pclass', 'Sex', 'Age', 'SibSp','Parch', 'Fare', 'Embarked']
dataset = df[ModelColumns]
targetVariable = df['Survived']
# fit an Extra Trees model to the data
model = ExtraTreesClassifier()
model.fit(dataset, targetVariable)
# display the relative importance of each attribute
print(model.feature_importances_)
```

```
[0.10462907 0.28388611 0.23315184 0.05369714 0.04241023 0.24578779
 0.03643782]
```

In [27]:
```
###
### Before you build your model
### Please give a short description of attributes you wish to choose
### Also state reason for choosing them
###
### Write your answer here
###


columns = ['Pclass', 'Sex', 'Age', 'Fare', 'Embarked']
```

Building a Decision Tree Model

In [28]:
```
###
### Before you train your model, split the data in training and testing set.
### You would end up with 4 slices of data
### X_train = contains training features
### y_train = contains training labels
### Y_train = contains testing features
### y_test = contains testing labels
###
### Write your code here
###


X = df[columns].copy()
Y = y=df[['Survived']].copy()
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

In [29]:
```
###
### Using sklearn train a simple decision Tree Classifier on training data
### Use default settings and do not change any parameters of the module
###
### Write your code here
###

survivalDecissionTree = DecisionTreeClassifier()
survivalDecissionTree.fit(X_train, y_train)
```

Out[29]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

In [30]:
```
###
### Points To Ponder:
###
### Can you pass categorical(in string format) data to machine learning model.
### If Yes. Then I definetly need to learn something from you.
### If No. Explain that error and how you handled that error?
###
```

Report Accuracy For Decision Tree

In [31]:
```
###
### Report your accuracy for training set and testing set both
### i.e.
### Training Accuracy = ???
### Testing Accuracy = ???
###

predictions = survivalDecissionTree.predict(X_test)
print(accuracy_score(y_true = y_test, y_pred = predictions))
```

```
0.726457399103139
```

Building a Simple Linear Model

In [32]:
```
###
### Using sklearn
### Train a simple Logistic Regression Model on training data
### Use default settings and do not change any parameters of the module
###

X1 = df[columns].copy()
Y1 = y1=df['Survived'].copy()
X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1)

logisticRegr = LogisticRegression()
logisticRegr.fit(X1_train, y1_train)
```

Out[32]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

Report Accuracy For Logistic Regression

In [33]:
```
###
### Report your accuracy for training set and testing set both
### i.e.
### Training Accuracy = ???
### Testing Accuracy = ???
###

logisticPredictions = logisticRegr.predict(X_test)
score = logisticRegr.score(X1_test, y1_test)
print(score)
```

```
0.8116591928251121
```