A bit of data science work/wonk:

Let's compare the speed of "vectorization" or matrix math, using numpy arrays vs. plain vanilla Python Loops

In [1]:
```
import numpy as np
from math import log10 as log10
import random
from time import time
print("Libraries imported")
```

```
Libraries imported
```

In [2]:
```
# The Log10 function takes your input number and finds what power you need t

print(np.log10(100))
```

```
2.0
```

In [4]:
```
# Let's set up some variables to run our comparative exercise

n = 10000000 # Set the variable n to 10,000,000

l1 = list(np.random.uniform(low=1.0, high=100.0, size=n)) # Create a list of

l2 = [] # Create an empty list

a1 = np.array(l1) # Create a numpy array for list L1

print("Variables are all set.")
```

```
Variables are all set.
```

In [6]:
```
# Loop through our list (L1) to find the Log10 of each number, and keep trac

t0 = time()

for i in l1:
    l2.append(log10(i))

loopTime = time()-t0

print("Loop time: " + str(round(loopTime, 3)) + " seconds")
```

```
Loop time: 2.416 seconds
```

In [10]:
```
t0 = time() # Again, start the timer/
a2 = np.log10(a1) # Perform the function on the entire numpy array (matrix)
vectorTime = time() - t0

print("Vectorized time: " + str(round(vectorTime, 3)) + " seconds")
```

```
Vectorized time: 0.069 seconds
```

In [12]:
```
# How much faster is vectorization than the Python Loop?

ratio = (loopTime / vectorTime)
print("Ratio = " + str(round(ratio,3)))
```

```
Ratio = 34.803
```

Conclusion

- Vectorization speeds things up a lot. What used to be computed in a day can be done in an hour.

On the Titanic!

In [63]:
```
# Import data science libraries for use

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [15]:
```
# Load training and test datasets into pandas dataframs

train = pd.read_csv("titanic_train.csv")
test = pd.read_csv("titanic_test.csv")
```

In [21]:
```
# Inspect your dataset, the first 5 rows

# Some column'field definitions/explanations:
# Pclass = passenfer class
#SibSp = # of siblings/spouces on board
# Parch = # of parents/children on board

train.head()
```

Out[21]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250(
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925(
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100(
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050(

In [22]:

```
test.head()
```

Out[22]:

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

In [23]:

```
# Use pandas to display basic description statistics
train.describe(include="all")
```

Out[23]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	
count	891.000000	891.000000	891.000000	891	891	714.000000	891.000000	891.0
unique	NaN	NaN	NaN	891	2	NaN	NaN	
top	NaN	NaN	NaN	Braund, Mr. Owen Harris	male	NaN	NaN	
freq	NaN	NaN	NaN	1	577	NaN	NaN	
mean	446.000000	0.383838	2.308642	NaN	NaN	29.699118	0.523008	0.3
std	257.353842	0.486592	0.836071	NaN	NaN	14.526497	1.102743	0.8
min	1.000000	0.000000	1.000000	NaN	NaN	0.420000	0.000000	0.0
25%	223.500000	0.000000	2.000000	NaN	NaN	20.125000	0.000000	0.0
50%	446.000000	0.000000	3.000000	NaN	NaN	28.000000	0.000000	0.0
75%	668.500000	1.000000	3.000000	NaN	NaN	38.000000	1.000000	0.0
max	891.000000	1.000000	3.000000	NaN	NaN	80.000000	8.000000	6.0

In [26]:

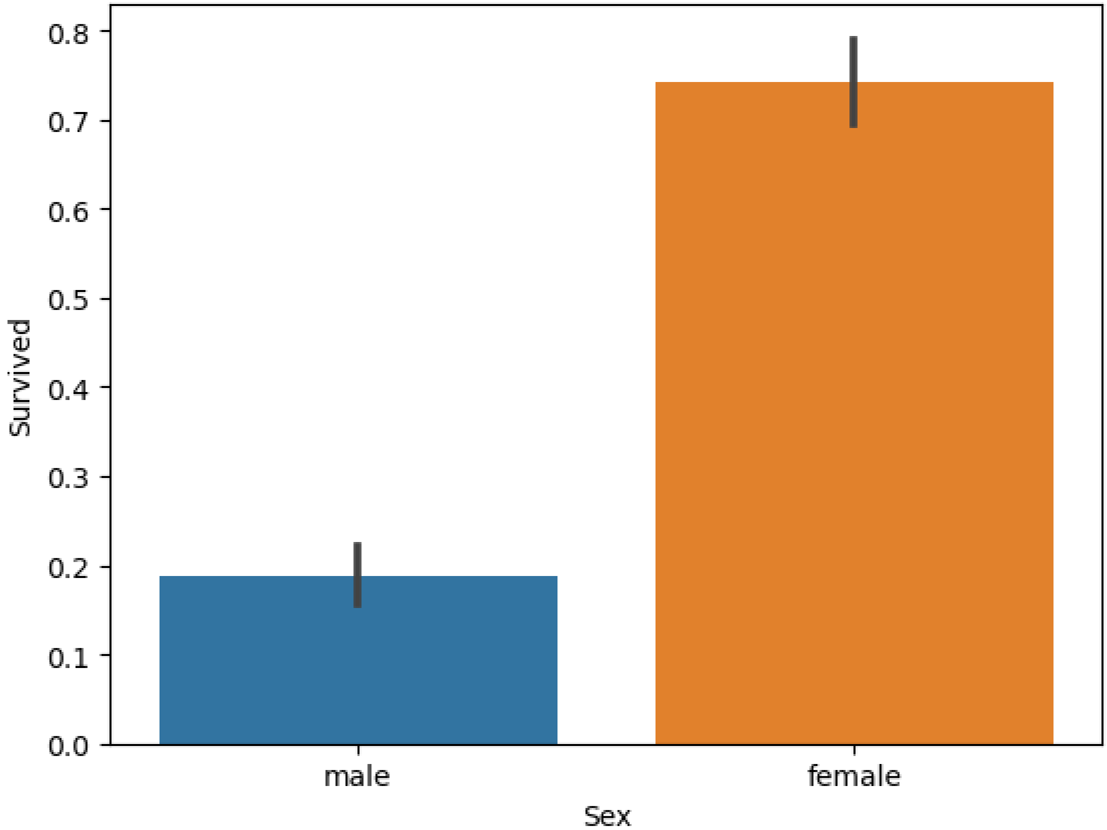
```
# Do the same with the test set
test.describe(include="all")

# Unlike the train dataset, it does not contain the "Survived" column
```

Out[26]:

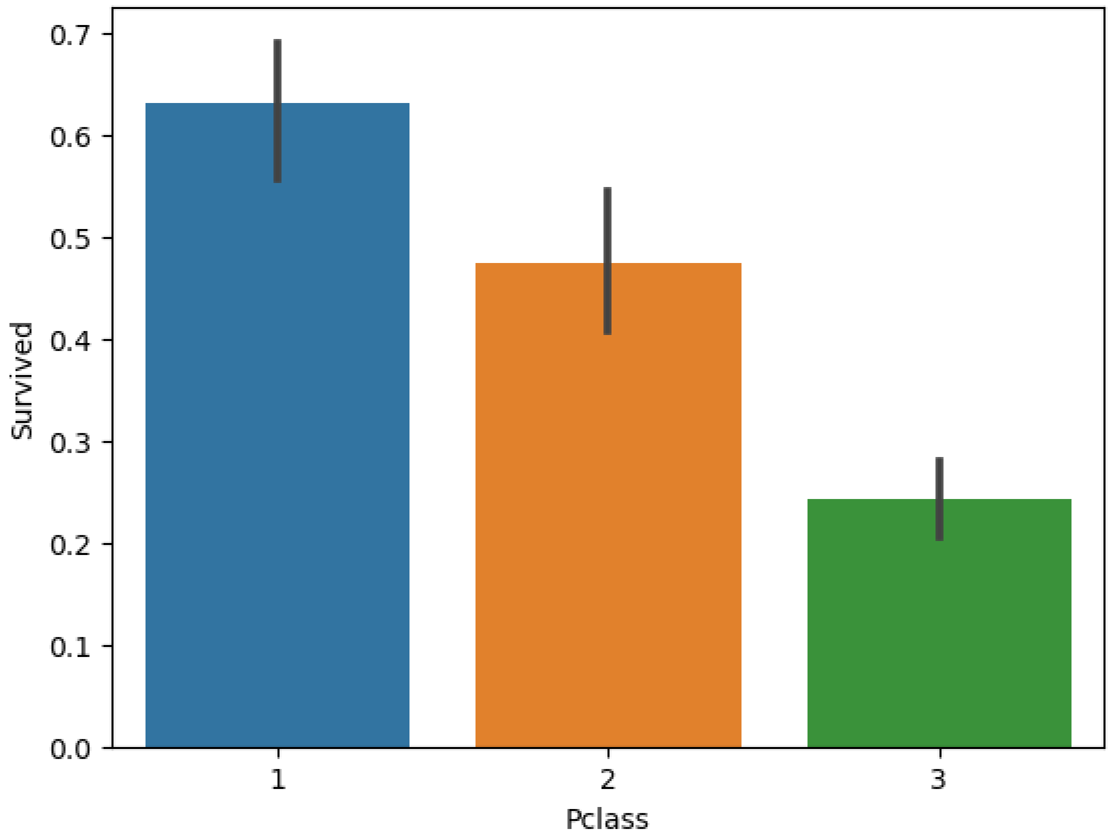
	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticke
count	418.000000	418.000000	418	418	332.000000	418.000000	418.000000	418
unique	NaN	NaN	418	2	NaN	NaN	NaN	363
top	NaN	NaN	Kelly, Mr. James	male	NaN	NaN	NaN	PC 17608
freq	NaN	NaN	1	266	NaN	NaN	NaN	5
mean	1100.500000	2.265550	NaN	NaN	30.272590	0.447368	0.392344	NaN
std	120.810458	0.841838	NaN	NaN	14.181209	0.896760	0.981429	NaN
min	892.000000	1.000000	NaN	NaN	0.170000	0.000000	0.000000	NaN
25%	996.250000	1.000000	NaN	NaN	21.000000	0.000000	0.000000	NaN
50%	1100.500000	3.000000	NaN	NaN	27.000000	0.000000	0.000000	NaN
75%	1204.750000	3.000000	NaN	NaN	39.000000	1.000000	0.000000	NaN
max	1309.000000	3.000000	NaN	NaN	76.000000	8.000000	9.000000	NaN

In [25]:

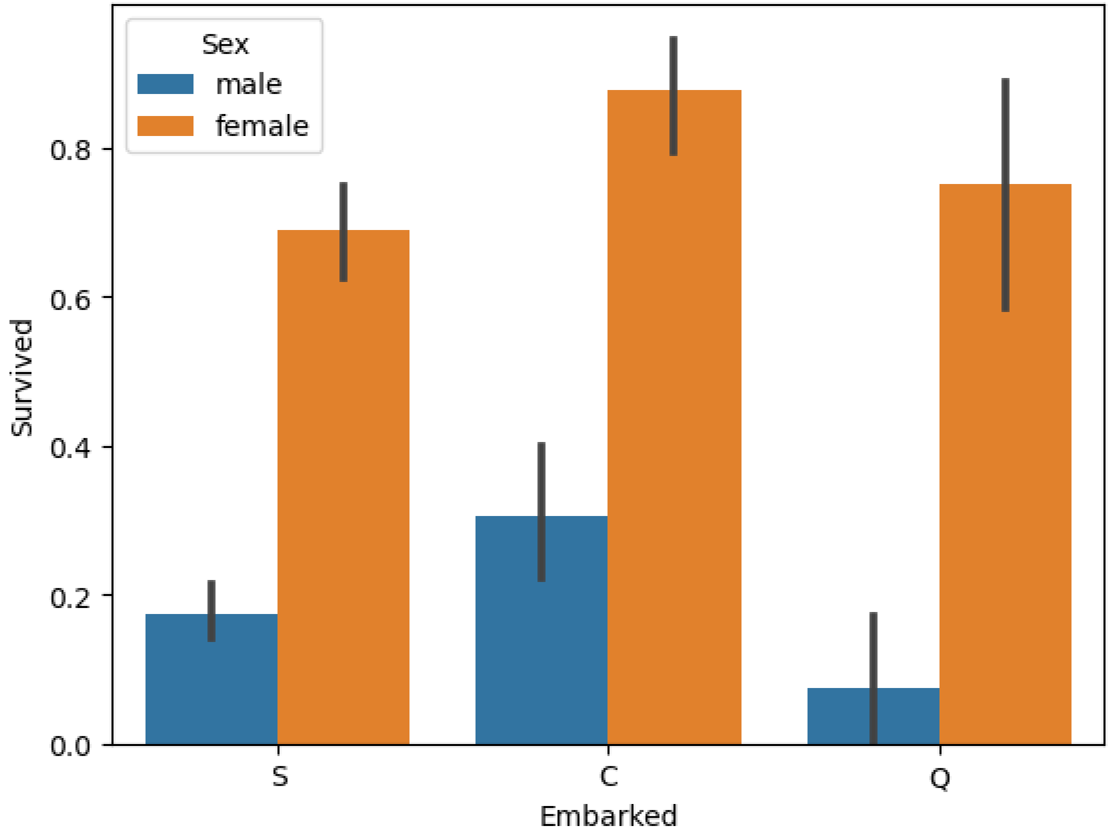
```
# Time for some visualizations
# Who was more likely to survive, males or females?
# Use a seaborn bar graph

sns.barplot(x="Sex", y="Survived", data=train);
```



In [27]:
```
# Which socio-economic class was more likely to survive?

sns.barplot(x="Pclass", y="Survived", data=train);
```



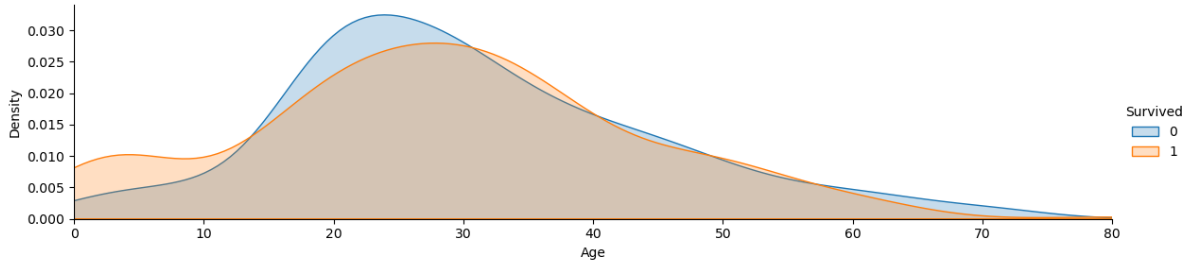
In [29]:
```
# Did the point of embarkation make a difference?
# C = Cherbourg (France), Q = Queenstown, S = Southampton

sns.barplot(x="Embarked", y="Survived", hue="Sex", data=train);
```



In [30]:
```
# What about age distribution?

a = sns.FacetGrid(train, hue="Survived", aspect=4)
a.map(sns.kdeplot, "Age", shade=True)
a.set(xlim=(0,train["Age"].max()))
a.add_legend()
```

```
/home/studio-lab-user/.conda/envs/default/lib/python3.9/site-packages/seabo
rn/axisgrid.py:848: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
/home/studio-lab-user/.conda/envs/default/lib/python3.9/site-packages/seabo
rn/axisgrid.py:848: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.
This will become an error in seaborn v0.14.0; please update your code.

  func(*plot_args, **plot_kwargs)
```

Out[30]: <seaborn.axisgrid.FacetGrid at 0x7f009407cdf0>



What are we solving for? What's our dependent y variable?

Survived (=1) or Died (=0)

In [31]:
```
y = train.Survived # This y varubale will store the "survived" data
```

In [32]:
```
y
```

Out[32]:
```
0      0
1      1
2      1
3      1
4      0
      ..
886    0
887    1
888    0
889    1
890    0
Name: Survived, Length: 891, dtype: int64
```

Now let's get the data ready

In [33]:
```
# Reformat the data into pandas dataframes to get ready for Machine Learning
# We have to know the shape fo these datasets later when we join and split t

train_shape = train.shape # Get the columns and rows of the training data
train_rows = train.shape[0] # Get number of rows from index 0
train_cols = train.shape[1] # Get number of columns from index 1

print("The shape of train is " + str(train_shape))
print("Our training set has " + str(train_rows) + " rows")
print("Our training set has " + str(train_cols) + " columns")

train.head() #Displays the first few rows
```

```
The shape of train is (891, 12)
Our training set has 891 rows
Our training set has 12 columns
```

Out[33]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.250(
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.283
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.925(
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.100(
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.050(

In [34]:

```
# DO the same thing for the test set

test_shape = test.shape
test_rows = test.shape[0]
test_cols = test.shape[1]

print("Our test set has " + str(train_rows) + " rows")
print("Our test set has " + str(train_cols) + " columns")

test.head()
```

```
Our test set has 891 rows
Our test set has 12 columns
```

Out[34]:

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

In [36]:

```
# Now concatenate the test and training sets together to make sure that they

combined = pd.concat((train,test)) # Combine the train and test dataframes t

combined_shape = combined.shape
combined_rows = combined.shape[0]
combined_cols = combined.shape[1]
print("Our concatenated set has " + str(combined_rows) + "rows")
print("Our concatednated set has " + str(combined_cols) + "columns")
combined.head()
```

```
Our concatenated set has 1309rows
Our concatednated set has 12columns
```

Out[36]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare
0	1	0.0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500
1	2	1.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1.0	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0.0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [38]:

```
# But what about the Survived column? It is in the train set but not the tes
# What are the entries in the combined dataset?

combined.sample(100)
```

Out[38]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fa
398	399	0.0	2	Pain, Dr. Alfred	male	23.0	0	0	244278	10.50
321	322	0.0	3	Danoff, Mr. Yoto	male	27.0	0	0	349219	7.89
150	1042	NaN	1	Earnshaw, Mrs. Boulton (Olive Potter)	female	23.0	0	1	11767	83.15
102	103	0.0	1	White, Mr. Richard Frasar	male	21.0	0	1	35281	77.28
271	272	1.0	3	Tornquist, Mr. William Henry	male	25.0	0	0	LINE	0.00
...	...	...	...	...	...	...	...	...	...	
137	1029	NaN	2	Schmidt, Mr. August	male	26.0	0	0	248659	13.00
278	279	0.0	3	Rice, Master. Eric	male	7.0	4	1	382652	29.12
536	537	0.0	1	Butt, Major. Archibald Willingham	male	45.0	0	0	113050	26.55
79	80	1.0	3	Dowdell, Miss. Elizabeth	female	30.0	0	0	364516	12.47
147	148	0.0	3	Ford, Miss. Robina Maggie "Ruby"	female	9.0	2	2	W./C. 6608	34.37

100 rows × 12 columns

In [43]:

```
# Now, we transform the data, take care of the nulls, and simplift it by arr
# and drop irrelevant or difficult columns/fields.
# We will eliminiate the Survived column, because we already set the Survive

def simplify_ages(df):
    df.Age = df.Age.fillna(-0.5)
    bins = (-1, 0, 5, 12, 18, 25, 35, 60, 120)
    group_names = ["Unknown", "Baby", "Child", "Teenager", "Student", "Young
    categories = pd.cut(df.Age, bins, labels=group_names)
    df.Age = categories
    return df

def simplify_cabins(df):
    df.Cabin = df.Cabin.fillna("N")
    df.Cabin = df.Cabin.apply(lambda x: x[0])
```

```
    return df

def simplify_fares(df):
    df.Fare = df.Fare.fillna(-0.5)
    bins = (-1, 0, 8, 15, 31, 1000)
    group_names = ["Unknown", "1_quartile", "2_quartile", "3_quartile", "4_q
    categories = pd.cut(df.Fare,bins,labels=group_names)
    df.Fare = categories
    return df

def drop_features(df):
    return df.drop(["Cabin", "Name", "Ticket", "PassengerId", "Survived"], a
```

In [44]:

```
# Defin a function to run those above, and run them

def transform_features(df):
    df = simplify_ages(df)
    df = simplify_cabins(df)
    df = simplify_fares(df)
    df = drop_features(df)
    return df

combined = transform_features(combined)
combined.head()
```

Out[44]:

	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	Student	1	0	1_quartile	S
1	1	female	Adult	1	0	4_quartile	C
2	3	female	Young Adult	0	0	1_quartile	S
3	1	female	Young Adult	1	0	4_quartile	S
4	3	male	Young Adult	0	0	2_quartile	S

In [45]:

```
# Now we will do one-hot encoding - essentially pivot the binned fields into
# for each bin value

combined = pd.get_dummies(combined)
combined.head()
```

Out[45]:

	Pclass	SibSp	Parch	Sex_female	Sex_male	Age_Unknown	Age_Baby	Age_Child	Age
0	3	1	0	0	1	0	0	0	
1	1	1	0	1	0	0	0	0	
2	3	0	0	1	0	0	0	0	
3	1	1	0	1	0	0	0	0	
4	3	0	0	0	1	0	0	0	

5 rows × 21 columns

In [47]:

```
# Load up the matrices, change the pandas dataframes into numpy arrays, chec
```

```
# and split the data back into training and test sets

# Create an array from "combined" that goes from the start to "train_rows"
X_train = combined[:train_rows]
print("X_train: " +str(X_train.shape))

# Create an array from "combined" that goes from "train_rows" to the end
X_test = combined[train_rows:]
print("X_test: " + str(X_test.shape))
```

```
X_train: (891, 21)
X_test: (418, 21)
```

In [67]:
```
# Load up a pile of classification models and processing toolds from Scikit-

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.gaussian_process import GaussianProcessClassifier
from sklearn.gaussian_process.kernels import RBF
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

In [75]:
```
# Create a list of names for these ML algos

names = ["Logistic regression", "k-Nearest Neighbors", "Linear SVM", "RBH SV
         "Gaussian Process", "Decision Tree", "Random Forest", "Neural Netwo

# Create a list of classifiers

classifiers = [
    LogisticRegression(),
    KNeighborsClassifier(3),
    SVC(kernel="linear", C=0.025),
    SVC(gamma=2, C=1),
    GaussianProcessClassifier(1.0 * RBF(1.0)),
    DecisionTreeClassifier(max_depth=5),
    RandomForestClassifier(max_depth=5, n_estimators=10, max_features=1),
    AdaBoostClassifier(),
    GaussianNB(),]
```

Loop through all the classifiers to see how they perform on the dataset

In [78]:
```
# Use the zip function to in a for-loop to run the ML algos with their names
# Use the fit function to match the data to the y dependent variable in each
# Measure their scores and print the results
```

```
for name, clf in zip(names, classifiers):
    clf.fit(X_train, y)
    accuracy = round(clf.score(X_train, y) * 100, 2)
    print(name, accuracy)
```

```
Logistic regression 82.15
k-Nearest Neighbors 87.21
Linear SVM 78.68
RBH SVM 90.12
Gaussian Process 85.75
Decision Tree 84.51
Random Forest 82.94
Neural Network 81.48
Adaboost 78.0
```

In [79]:
```
# Let's pick thd winner, the Support Vector Machine w/ RBF kernel function
# Store the model's predictions for each input (feature vector)

clf = SVC(gamma=2, C=1)
clf.fit(X_train, y)
accuracy = round(clf.score(X_train, y) * 100, 2)
print("Our accurancy score is: "+ str(accuracy))
predictions = clf.predict(X_test)
```

```
Our accurancy score is: 90.12
```

In [81]:
```
# Export our preiction into a file for use by the outside world

solution = pd.DataFrame({"PassengerId":test.PassengerId, "Survived":predicti
solution.to_csv("best_fit.csv", index=False)
print("Prediction file has been created")
```

```
Prediction file has been created
```

In [82]:
```
best_fit = pd.read_csv("best_fit.csv")
best_fit.sample(20)
```

Out[82]:

	PassengerId	Survived
84	976	0
272	1164	1
77	969	0
122	1014	1
189	1081	0
250	1142	1
284	1176	1
372	1264	0
377	1269	0
78	970	0
393	1285	0
246	1138	1
344	1236	0
139	1031	0
14	906	1
273	1165	1
370	1262	0
160	1052	1
326	1218	0
312	1204	0

In [ ]: