Data science project.

Introduction.

Titanic Data Set, As the name suggests (no points for guessing), this data set provides the data on all the passengers who were aboard the RMS Titanic when it sank on 15 April 1912 after colliding with an iceberg in the North Atlantic ocean. It is the most commonly used and referred to data set for beginners in data science. With 891 rows and 12 columns, this data set provides a combination of variables based on personal characteristics such as age, class of ticket and sex, and tests one's classification skills.

Objective:

Predict the survival of the passengers aboard RMS Titanic. Link to the Data Set: https://www.kaggle.com/c/titanic/data

Program

> The whole program is written with the help of python lanugage.

In [1]:

```
#pyplot is matplotlib's plotting framework this imports the module matplotlib
import matplotlib.pyplot as plt

#library for scientific computing,np overcome conflict due to namespaces
import numpy as np

#providing fast, flexible, and expressive data structures designed to make working with structured
import pandas as pd

#dict of Series objects
from pandas import Series, DataFrame

#seaborn help us see the different stylistic parameters
import seaborn as sns

#To ignore warnings
import warnings
warnings.filterwarnings("ignore")

#its a directiory helps Python to display matplotlib plots
%matplotlib inline
```

General Survival rate by Categories

In [2]:

```
#function will be evaluated against the column names,returning names where the callable function e
valuates to True.
titanic=pd.read_csv('train.csv')

#creating the table of number of people died and number of people survived
titanic_Survived=titanic.groupby(['Survived'])

#describe the table and  Remove missing values
titanic_Survived.describe().dropna()
```

Out[2]:

	Age								Fare		...	Pclass		SibSp		

	Age								Fare		...	Pclass		SibSp		
Survived	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std
0	424.0	30.626179	14.172110	1.00	21.0	28.0	39.0	74.0	549.0	22.117887	...	3.0	3.0	549.0	0.553734	1.28839
Survived																
1	290.0	28.343690	14.950952	0.42	19.0	28.0	36.0	80.0	342.0	48.395408	...	3.0	3.0	342.0	0.473684	0.70868

2 rows × 48 columns

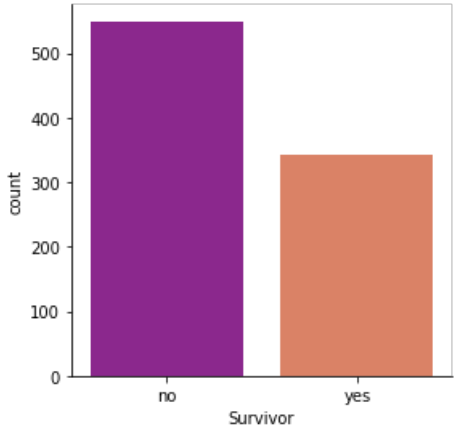
Mapping of ratio of survival

In [3]:

```
#Let's start by creating a new column for legibility purposes through mapping
titanic['Survivor'] = titanic.Survived.map({0:'no', 1:'yes'})

# Let's just get a quick overall view of survied vs died.
sns.factorplot('Survivor', data=titanic, palette='plasma', kind='count')
```

Out[3]:

```
<seaborn.axisgrid.FacetGrid at 0x17fe87ad2e8>
```



> Analysis says the rate of survival is less.

Analysis based on age.

In [4]:

```
#sort it by age and gender
titanic_Survived=titanic.groupby(['Age','Sex'])
#describe a person's age and gender in titanic_survied
titanic_Survived.describe().dropna()
```

Out[4]:

		Fare								Parch		...	Sib
		count	mean	std	min	25%	50%	75%	max	count	mean	...	75%
Age	Sex												
0.75	female	2.0	19.258300	0.000000	19.2583	19.258300	19.25830	19.258300	19.2583	2.0	1.000000	...	2.00
0.83	male	2.0	23.875000	7.247845	18.7500	21.312500	23.87500	26.437500	29.0000	2.0	1.500000	...	0.75
1.00	female	2.0	13.437500	3.258631	11.1333	12.285400	13.43750	14.589600	15.7417	2.0	1.500000	...	0.75
	male	5.0	36.633340	9.725496	20.5750	37.004200	39.00000	39.687500	46.9000	5.0	1.600000	...	4.00

		Fare								Parch			Sib
		count	mean	std	min	25%	50%	75%	max	count	mean	...	75%
Age	Sex												
2.00	female	6.0	43.245833	53.738575	10.4625	15.715625	26.95000	30.431250	151.5500	6.0	1.500000	...	2.5
	male	4.0	28.971875	7.874870	21.0750	24.768750	27.56250	31.765625	39.6875	4.0	1.000000	...	4.0
3.00	female	2.0	31.327100	14.498659	21.0750	26.201050	31.32710	36.453150	41.5792	2.0	1.500000	...	2.5
	male	4.0	23.009375	7.019551	15.9000	18.037500	22.37500	27.346875	31.3875	4.0	1.250000	...	1.75
4.00	female	5.0	22.828340	9.853082	13.4167	16.700000	22.02500	23.000000	39.0000	5.0	1.200000	...	1.0
	male	5.0	36.258320	26.721733	11.1333	27.900000	29.12500	31.275000	81.8583	5.0	1.600000	...	4.0
5.00	female	4.0	22.717700	8.512145	12.4750	17.562475	23.50415	28.659375	31.3875	4.0	1.250000	...	2.5
6.00	female	2.0	32.137500	1.219759	31.2750	31.706250	32.13750	32.568750	33.0000	2.0	1.500000	...	3.0
7.00	male	2.0	34.406250	7.468815	29.1250	31.765625	34.40625	37.046875	39.6875	2.0	1.000000	...	4.0
8.00	female	2.0	23.662500	3.659278	21.0750	22.368750	23.66250	24.956250	26.2500	2.0	1.500000	...	2.25
	male	2.0	32.937500	5.391689	29.1250	31.031250	32.93750	34.843750	36.7500	2.0	1.000000	...	3.25
9.00	female	4.0	27.198950	8.396014	15.2458	24.736450	29.58750	32.050000	34.3750	4.0	1.750000	...	3.25
	male	4.0	28.678125	13.773542	15.9000	19.368750	25.95625	35.265625	46.9000	4.0	1.750000	...	4.25
11.00	male	3.0	61.895833	52.246041	18.7875	32.843750	46.90000	83.450000	120.0000	3.0	1.333333	...	3.0
13.00	female	2.0	13.364600	8.676766	7.2292	10.296900	13.36460	16.432300	19.5000	2.0	0.500000	...	0.0
14.00	female	4.0	42.291675	52.719301	7.8542	10.394825	20.65625	52.553100	120.0000	4.0	0.500000	...	1.0
	male	2.0	43.293750	5.100008	39.6875	41.490625	43.29375	45.096875	46.9000	2.0	1.500000	...	4.75
15.00	female	4.0	60.261475	100.769291	7.2250	7.828150	11.24170	63.675025	211.3375	4.0	0.250000	...	0.25
16.00	female	6.0	41.043750	30.358319	7.7333	15.662500	43.15000	55.209400	86.5000	6.0	0.666667	...	0.0
	male	11.0	17.400382	11.453374	7.7750	8.633350	10.50000	23.125000	39.6875	11.0	0.454545	...	1.0
17.00	female	6.0	35.130550	40.569385	7.9250	10.875000	13.22915	46.364575	108.9000	6.0	0.333333	...	1.0
	male	7.0	22.611314	38.931801	7.0542	7.177100	8.66250	8.662500	110.8833	7.0	0.428571	...	0.5
18.00	female	13.0	53.616669	87.309278	6.7500	9.350000	14.45420	23.000000	262.3750	13.0	0.692308	...	1.0
	male	13.0	22.510254	31.537173	6.4958	7.795800	8.30000	13.000000	108.9000	13.0	0.153846	...	1.0
19.00	female	7.0	30.727986	28.144091	7.8542	16.939600	26.00000	28.141650	91.0792	7.0	0.285714	...	1.0
	male	18.0	26.757861	60.245772	0.0000	7.895800	8.10415	12.375000	263.0000	18.0	0.166667	...	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
44.00	male	6.0	26.020833	32.140685	7.9250	8.050000	12.07500	23.525000	90.0000	6.0	0.166667	...	0.75
45.00	female	6.0	42.453483	60.474702	7.7500	13.738550	20.35210	27.487500	164.8667	6.0	1.166667	...	1.0
	male	6.0	31.183333	27.989934	6.9750	12.675000	26.55000	33.262500	83.4750	6.0	0.000000	...	0.0
45.50	male	2.0	17.862500	15.043697	7.2250	12.543750	17.86250	23.181250	28.5000	2.0	0.000000	...	0.0
46.00	male	3.0	55.458333	27.056796	26.0000	43.587500	61.17500	70.187500	79.2000	3.0	0.000000	...	0.5
47.00	female	2.0	33.527100	26.908383	14.5000	24.013550	33.52710	43.040650	52.5542	2.0	0.500000	...	1.0
	male	7.0	25.908329	16.624565	7.2500	12.000000	25.58750	36.260400	52.0000	7.0	0.000000	...	0.0
48.00	female	4.0	41.226050	16.820377	25.9292	32.263550	36.98750	45.950000	65.0000	4.0	1.250000	...	1.0
	male	5.0	35.226680	28.819985	7.8542	13.000000	26.55000	52.000000	76.7292	5.0	0.000000	...	1.0
49.00	female	2.0	51.329200	35.921024	25.9292	38.629200	51.32920	64.029200	76.7292	2.0	0.000000	...	0.75
	male	4.0	64.229175	48.214973	0.0000	42.696900	73.01670	94.548975	110.8833	4.0	0.250000	...	1.0
50.00	female	5.0	64.646660	102.581067	10.5000	10.500000	26.00000	28.712500	247.5208	5.0	0.400000	...	0.0
	male	5.0	63.405000	55.781371	8.0500	13.000000	55.90000	106.425000	133.6500	5.0	0.000000	...	1.0
51.00	male	6.0	20.551400	21.305637	7.0542	7.825000	10.28750	23.043750	61.3792	6.0	0.166667	...	0.0
52.00	female	2.0	85.883350	10.771570	78.2667	82.075025	85.88335	89.691675	93.5000	2.0	0.500000	...	1.0
	male	4.0	34.162500	31.397011	13.0000	13.375000	22.00000	42.787500	79.6500	4.0	0.250000	...	0.25
54.00	female	3.0	53.555567	28.093061	23.0000	41.200000	59.40000	68.833350	78.2667	3.0	1.000000	...	1.0
	male	5.0	39.030000	25.467489	14.0000	26.000000	26.00000	51.862500	77.2875	5.0	0.200000	...	0.0

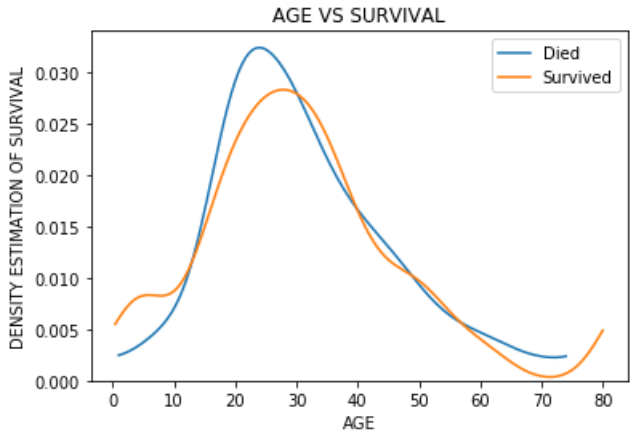
56.00	**male**	Fare	30.915267	4.479034	26.5500	28.622900	30.69580	33.097900	35.5000	Parch	0.000000	...	Sib
58.00	**female**	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%
Age	**Sex**	2.0	71.487500	59.096449	29.7000	50.593750	71.48750	92.381250	113.2750	2.0	1.000000	...	0.0
59.00	**male**	2.0	10.375000	4.419417	7.2500	8.812500	10.37500	11.937500	13.5000	2.0	0.000000	...	0.0
60.00	**male**	3.0	48.250000	27.516858	26.5500	32.775000	39.00000	59.100000	79.2000	3.0	0.666667	...	1.0
61.00	**male**	3.0	24.019433	15.410889	6.2375	19.279150	32.32080	32.910400	33.5000	3.0	0.000000	...	0.0
62.00	**male**	3.0	21.200000	9.266472	10.5000	18.525000	26.55000	26.550000	26.5500	3.0	0.000000	...	0.0
63.00	**female**	2.0	43.772900	48.345456	9.5875	26.680200	43.77290	60.865600	77.9583	2.0	0.000000	...	0.7
64.00	**male**	2.0	144.500000	167.584307	26.0000	85.250000	144.50000	203.750000	263.0000	2.0	2.000000	...	0.7
65.00	**male**	3.0	32.093067	27.536262	7.7500	17.150000	26.55000	44.264600	61.9792	3.0	0.333333	...	0.0
70.00	**male**	2.0	40.750000	42.779960	10.5000	25.625000	40.75000	55.875000	71.0000	2.0	0.500000	...	0.7
71.00	**male**	2.0	42.079200	10.500536	34.6542	38.366700	42.07920	45.791700	49.5042	2.0	0.000000	...	0.0

110 rows × 48 columns

Plot of graph based on Age vs Survival

In [5]:

```
#Age with missing value

#using module seaborn lable as died if survived=0
sns.kdeplot(titanic.loc[(titanic['Survived']==0),'Age'], label = 'Died',cut=0)

#using module seaborn lable as survived if survived=1
sns.kdeplot(titanic.loc[(titanic['Survived']==1),'Age'], label = 'Survived',cut=0)

#plot a graph with x-axis age
plt.xlabel("AGE")

#plot a graph with y-axis DENSITY ESTIMATION OF SURVIVAL
plt.ylabel("DENSITY ESTIMATION OF SURVIVAL")

#and the title of graph as AGE VS SURVIVAL
plt.title("AGE VS SURVIVAL")
plt.show()
```



> Analysis says lesser the age, survival rate is high.

Analysis based on gender.

In [6]:

```
# group the survival in the base of survived and gender
titanic_Survived=titanic.groupby(['Survived','Sex'])
#discribe titatic_survived in the form of table
titanic_Survived.describe().dropna()
```

Out[6]:

		Age								Fare		...	Pclass		SibSp	
		count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean
Survived	**Sex**															
0	**female**	64.0	25.046875	13.618591	2.00	16.75	24.5	33.25	57.0	81.0	23.024385	...	3.0	3.0	81.0	1.20987
	male	360.0	31.618056	14.056019	1.00	21.75	29.0	39.25	74.0	468.0	21.960993	...	3.0	3.0	468.0	0.44017
1	**female**	197.0	28.847716	14.175073	0.75	19.00	28.0	38.00	63.0	233.0	51.938573	...	3.0	3.0	233.0	0.51502
	male	93.0	27.276022	16.504803	0.42	18.00	28.0	36.00	80.0	109.0	40.821484	...	3.0	3.0	109.0	0.38532

4 rows × 48 columns

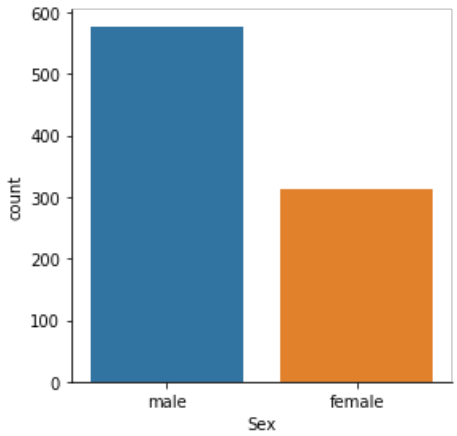
Plot based on sex ratio.

In [7]:

```
#using module seaborn ploting graph of sex versus count
sns.factorplot('Sex', data=titanic, kind="count")
```

Out[7]:

```
<seaborn.axisgrid.FacetGrid at 0x17fe8baf390>
```



> Analysis says male were more in number.

Calculation of mean.

In [8]:

```
#Female and male that survived in general
#observing the detail and taking the mean of gender and survived passenger
survived=titanic.groupby('Sex')['Survived']
survived.mean()
```

Out[8]:

```
Sex
female    0.742038
male      0.188908
Name: Survived, dtype: float64
```

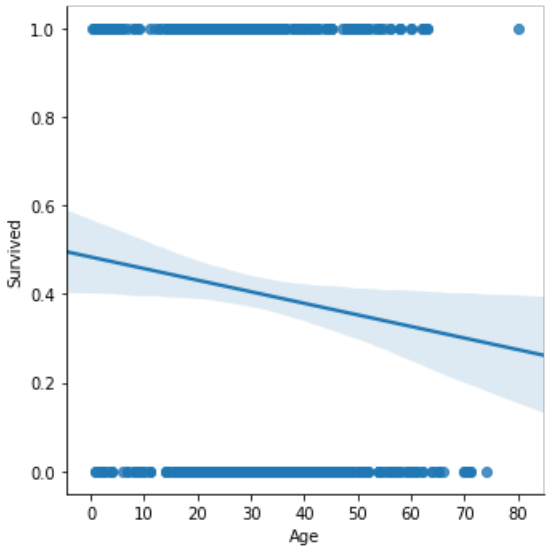
let's use a linear plot on age versus survival

In [9]:

```
#using seaborn linear plot a graph of age and survived
sns.lmplot('Age','Survived', data=titanic)
```

Out[9]:

```
<seaborn.axisgrid.FacetGrid at 0x17fe8c16320>
```



> The plot shows that the older the passenger is, the less chance he/she would survive.

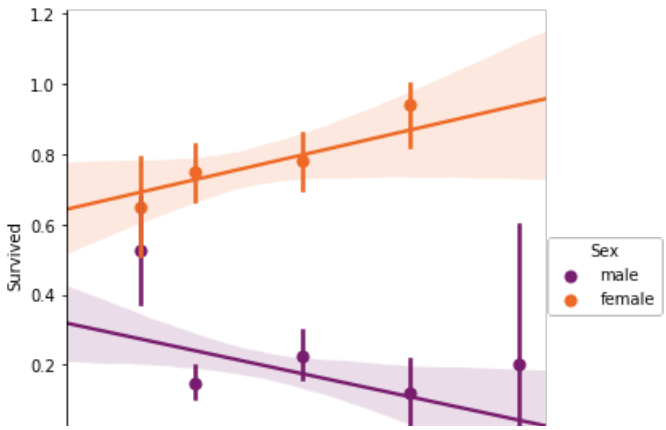
Let's use a linear plot on sex ratio

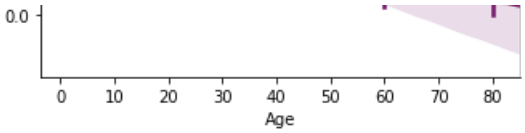
In [10]:

```
#ploting age with the difference of 10
generations=[10,20,40,60,80]
#ploting a graph of age and survived in case of male and female
sns.lmplot('Age','Survived',hue='Sex', data=titanic, palette='inferno',x_bins=generations)
```

Out[10]:

```
<seaborn.axisgrid.FacetGrid at 0x17fe8c90278>
```





We shall do the analysis by considering under 16 age as child.

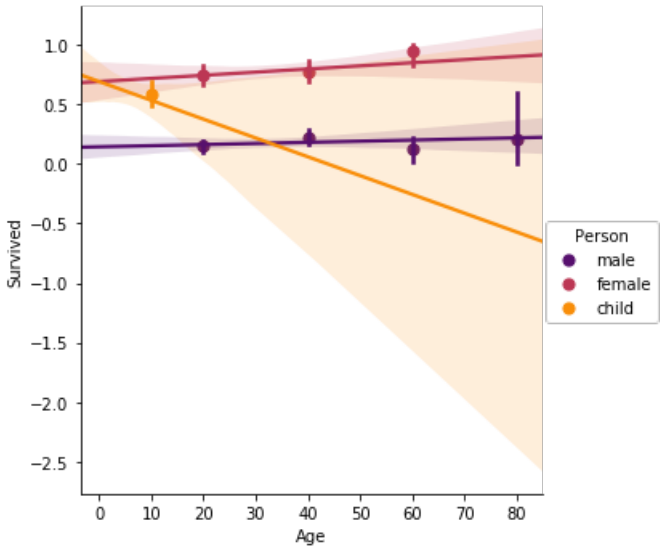
And check the survival rate

In [11]:

```
# We'll treat anyone as under 16 as a child, and then use the apply technique with a function to c
reate a new column

# First let's make a function to sort through the sex
def male_female_child(passenger):
    # Take the Age and Sex
    age,sex = passenger
    # Compare the age , otherwise leave the sex
    if age < 16:
        return 'child'
    else:
        return sex


# plot a graph of  sevived versus age with row person and coloumn male,female,child
titanic['Person']= titanic[['Age','Sex']].apply(male_female_child, axis=1)
#ploting age with the difference of 10
generations=[10,20,40,60,80]
#sns.implot-functions are the figure-level function that combines regplot() and FacetGrid
sns.lmplot('Age','Survived',hue='Person', data=titanic, palette='inferno',x_bins=generations)
```

Out[11]:

```
<seaborn.axisgrid.FacetGrid at 0x17fe8d26fd0>
```



> **Data Analysis by visiualizatin with respect to age and gender is been analized**

Final data preparation for prediction

Checking the null values & non-null

In [12]:

```
#Re-initializing the files
```

```
#Re-intializing the files
#function will be evaluated against the column names,returning names where the callable function e
valuates to True.
train = pd.read_csv("train.csv")
test = pd.read_csv("test.csv")

#Grouping & Droping the null values
train_y = train['Survived']
#Drop a variable where axis=1 denotes that we are referring to a column, not a row
train_x = train.drop(['Survived','Age',"Sex"],axis=1)

#Drop a variable where axis=1 denotes that we are referring to a column, not a row
test_x = test.drop(['Age','Sex'],axis=1)

train_y.head()
train_x.head()
test_x.head()
train_x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 9 columns):
PassengerId    891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
SibSp          891 non-null int64
Parch          891 non-null int64
Ticket         891 non-null object
Fare           891 non-null float64
Cabin          204 non-null object
Embarked       889 non-null object
dtypes: float64(1), int64(4), object(4)
memory usage: 62.7+ KB
```

Prediction Algorithm

Predicting using Randome forest regressor

Definiton:-

Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees.

In [13]:

```
#sklearn is a library which  is used in mll.
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

## Add these lines to turn off the warnings
import warnings
warnings.filterwarnings("ignore")

'''To instantiate a DataFrame from data with element order preserved use pd.read_csv,
IO Tools docs for more information on iterator and chunksize'''
train = pd.io.parsers.read_csv("train.csv")
test = pd.io.parsers.read_csv('test.csv')

#Removing missing values
train = train.dropna(axis=0)

train_Y = train.Survived
train_predictor_columns = ['Pclass', 'Sex', 'Age', 'Fare']
train_X = train[train_predictor_columns]
test_X = test[train_predictor_columns]

### label encode the categorical values and convert them to numbers
le = LabelEncoder()
```

```
le.fit(train_X['Sex'].astype(str))
train_X['Sex'] = le.transform(train_X['Sex'].astype(str))
test_X['Sex'] = le.transform(test_X['Sex'].astype(str))

#astype always returns a newly allocated array
le.fit(train_X['Pclass'].astype(str))
train_X['Pclass'] = le.transform(train_X['Pclass'].astype(str))
#A transform is a Python callable that, when called with a displayable, returns another displayabl
e
test_X['Pclass'] = le.transform(test_X['Pclass'].astype(str))

### train the model
my_model = RandomForestRegressor()
#Fit function is generic term which is used to best match the curvature of given data points
my_model.fit(train_X, train_Y)


## fill the missing values in test data
for col in train_predictor_columns:
    #for the calculation of the mean using sum and length of train_x
    mean_col = sum(train_X[col]) / len(train_X[col])
    #fillna()will FILL these NaN or NULL values with some value provided to the function
    test_X[col] = test_X[col].fillna(mean_col)

predictions = my_model.predict(test_X)
print(predictions)
```

```
[0.2 0.  0.1 0.5 1.  0.7 0.1 0.7 0.2 0.3 0.2 0.3 1.  0.1 1.  0.9 0.5 0.1
 0.2 0.1 0.2 0.8 1.  0.3 1.  0.4 1.  0.4 0.8 0.6 0.5 0.7 1.  1.  0.3 0.1
 0.1 0.2 0.4 0.2 0.1 0.9 0.  1.  1.  0.4 0.2 0.1 1.  1.  0.7 0.7 1.  1.
 0.4 0.9 0.5 0.1 0.4 1.  0.5 0.8 0.2 0.2 0.6 1.  0.2 0.  0.1 1.  0.2 0.3
 0.1 0.7 1.  0.6 0.2 0.8 0.2 0.2 1.  0.1 0.6 0.2 0.2 0.4 0.2 0.2 0.1 1.
 1.  0.1 1.  0.2 0.8 0.1 1.  0.3 0.2 0.5 1.  0.9 0.1 0.1 0.2 0.7 0.1 0.1
 0.2 0.3 0.1 0.1 1.  0.2 1.  0.4 0.  0.8 0.3 1.  1.  0.1 1.  0.2 0.1 1.
 0.1 1.  0.2 0.4 0.7 0.4 1.  0.  0.1 0.4 0.2 0.6 0.2 0.6 1.  0.3 0.2 0.8
 0.7 0.3 0.4 0.3 0.9 0.3 1.  0.2 0.6 1.  0.7 0.2 1.  0.2 0.7 1.  0.1 1.
 1.  0.1 0.1 1.  0.5 0.4 0.9 0.2 0.1 0.2 0.7 0.1 0.7 1.  1.  0.1 1.  1.
 0.2 0.2 1.  0.1 1.  0.1 1.  0.5 1.  0.1 0.9 0.8 0.8 0.1 1.  0.5 1.  0.2
 0.7 0.1 1.  1.  0.  1.  0.6 0.7 0.1 0.6 0.9 0.4 0.8 0.1 0.3 1.  0.1 0.6
 0.1 0.1 0.9 0.2 1.  0.3 1.  0.1 0.8 1.  0.4 0.1 0.1 0.4 0.6 1.  0.1 0.2
 0.4 0.3 0.3 0.1 1.  1.  0.7 1.  0.  0.2 0.6 0.7 1.  0.6 1.  1.  0.7 0.3
 0.9 0.4 0.7 0.1 0.1 0.5 1.  0.1 0.  0.3 1.  0.8 0.2 0.2 0.1 0.1 0.1 0.5
 0.3 0.1 1.  1.  0.1 1.  0.6 0.2 0.2 0.3 0.2 1.  0.1 1.  0.8 0.1 0.1 0.4
 0.1 0.2 0.4 0.1 0.1 0.  0.2 0.4 0.7 0.6 0.  0.3 0.4 0.4 0.2 0.4 0.1 1.
 0.3 1.  0.2 1.  0.4 0.1 0.1 0.1 1.  0.2 0.1 0.3 0.5 0.4 0.1 0.1 0.6 0.9
 1.  0.4 1.  0.  0.4 0.3 1.  0.4 0.1 1.  0.5 0.3 0.8 0.  0.9 0.1 0.4 0.4
 0.2 1.  0.4 0.2 0.6 0.1 0.6 1.  1.  0.6 0.2 0.5 0.8 0.6 1.  0.2 0.1 1.
 0.6 1.  1.  0.5 1.  1.  0.4 1.  1.  0.3 0.3 1.  0.4 0.1 1.  1.  0.2 0.3
 0.2 1.  0.1 0.4 1.  1.  0.2 1.  0.2 0.1 0.1 1.  0.6 0.9 0.7 0.1 0.3 1.
 0.2 1.  0.1 0.2 1.  0.  1.  0.5 0.6 0.3 0.7 0.  0.1 0.8 0.1 1.  0.2 0.2
 1.  0.  0.2 0.6]
```

In [14]:

```
'''A random forest is a meta estimator that fits a number of classifying decision trees on various
sub-samples of the dataset
and use averaging to improve the predictive accuracy and control over-fitting'''
model_tree = RandomForestRegressor()
#Fit function is generic term which is used to best match the curvature of given data points
model_tree = model_tree.fit(train_X, train_Y)
#predict
a = model_tree.predict(test_X)

test_x["Survived"] = a
```

> ***Lets create a new file that stores the predicted data.***

In [16]:

```
'''integrated data alignment features of the pandas data structures set pandas apart from
```

```
the majority of related tools for working with labeled data'''

submission_data = pd.DataFrame({'PassengerId':test['PassengerId'],'Survived':test_x['Survived']})
'''A CSV file is a human readable text file where each line has a
number of fields, separated by commas or some other delimiter'''

submission_data.to_csv("submission_data.csv",index=0)
```

Conclusion

Based on the analysis and prediticon alogrithm the respective output's are shown.