1. Introduction

On April 15, 1912, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.
While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others.
I will build a predictive model that answers the question: “what sorts of people were more likely to survive?” using passenger data (ie name, age, gender, socio-economic class, etc).

2. Analyzing data and Data import

1) Data importing and brief analyzing

Import training data, test data and basic packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

```
train = pd.read_csv('/home/xiangliu/桌面/data challenge/titanic/train.cs
test = pd.read_csv('/home/xiangliu/桌面/data challenge/titanic/test.csv'
```

Age feature has more than 150 missing values, it’s important to fill these missing value in the future. Cabin feature only has 204 data, it should be explore why most passengers didn’t have a cabin. Embarked feature(Port of Embarkation) only has 2 missing values, which has almost no influence on this dataset.

There are about 38 percent people survived. People's average age is 30(ignore missing values)

```
train.info()
```

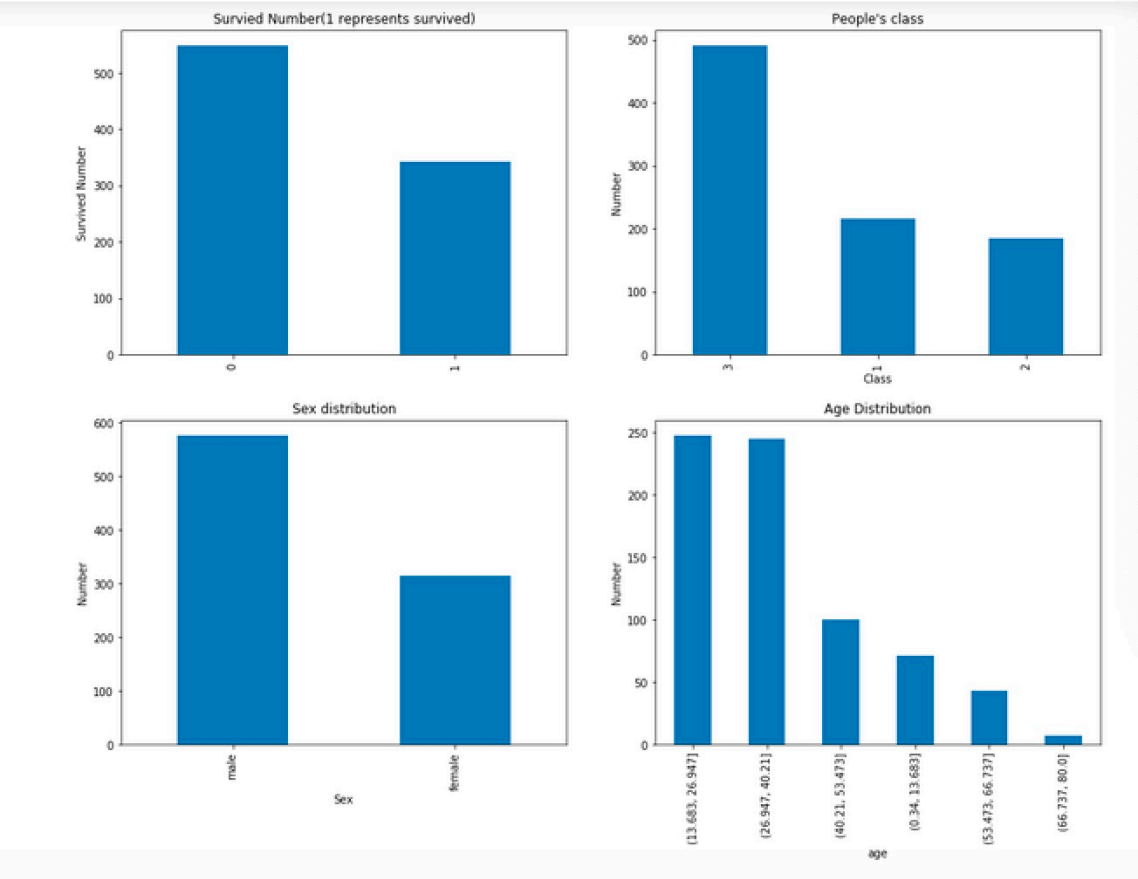
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 non-null float64
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(2), int64(5), object(5)
memory usage: 83.6+ KB
```

```
train.describe()
```

	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

2) Data analyzing and visualization

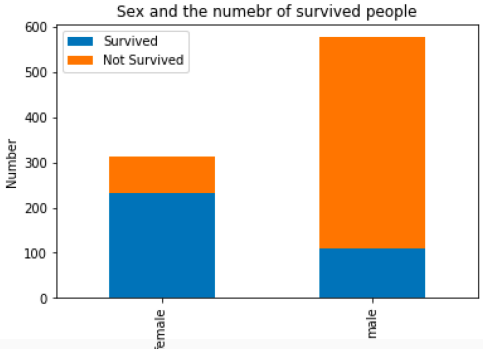
The number of dead people are more than the number of survived people.
Class 3 has a lot more people than Class 1 and Class 2.
Male are more than female.
People aged from 13 to 26 and 27 to 40 are the most.



'**Sex':** Although, the number of male is much more than the number of female, female are more likely to be survived. Sex is a important feature that should be considered.

```
Survived_0 = train.Sex[train.Survived == 0].value_counts()
Survived_1 = train.Sex[train.Survived == 1].value_counts()
data1 = pd.DataFrame({'Survived':Survived_1,'Not Survived':Survived_0})
data1.plot(kind='bar',stacked=True)
plt.title('Sex and the numebr of survived people')
plt.ylabel('Number')
```

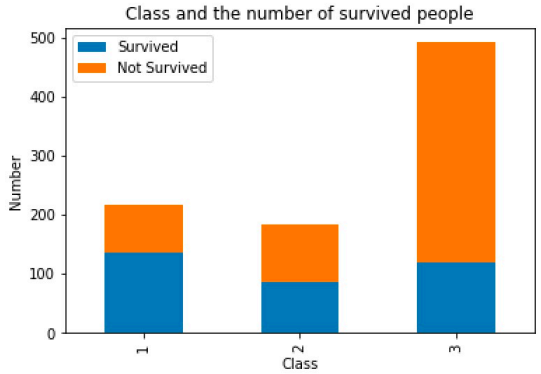
```
Text(0, 0.5, 'Number')
```



‘Pclass’: Class 3 has the most people, but least ratio of survived.
Class 1's people are more likely to be survived. Maybe because class 1's people are richer and have higher society status, which improved the probability of being rescued.

```
Survived_0 = train.Pclass[train.Survived == 0].value_counts()
Survived_1 = train.Pclass[train.Survived == 1].value_counts()
data2 = pd.DataFrame({'Survived':Survived_1,'Not Survived':Survived_0})
data2.plot(kind='bar',stacked=True)
plt.title('Class and the number of survived people')
plt.xlabel('Class')
plt.ylabel('Number')
```

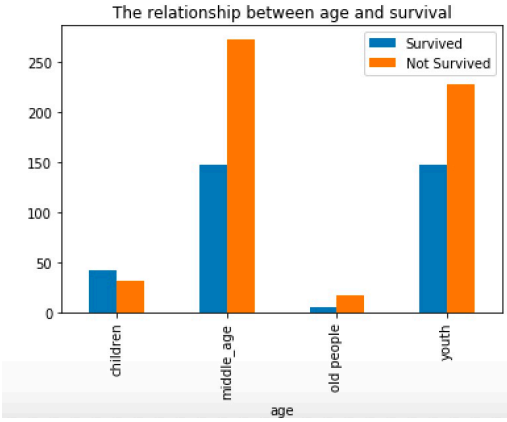
```
Text(0, 0.5, 'Number')
```



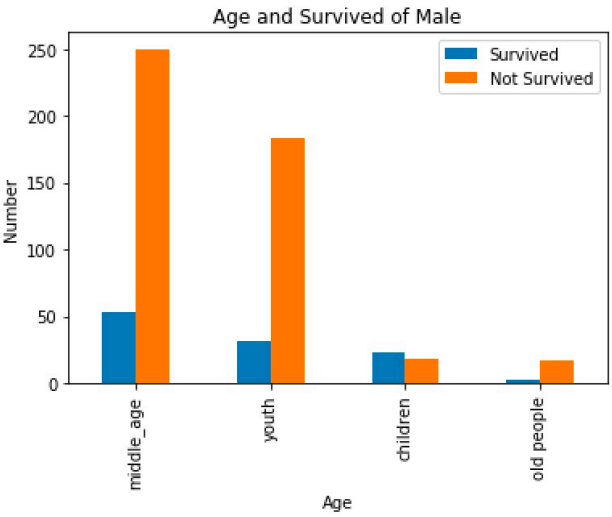
‘Age’: Divided the age into 4 groups: children, youth, middle age, old people. Children have more chances to be survived. Older people have less chances. The number of survived middle age is almost equal to the number of survived youth, but the total number of middle age people is larger, so the survival rate of youth is higher than the middle age people.

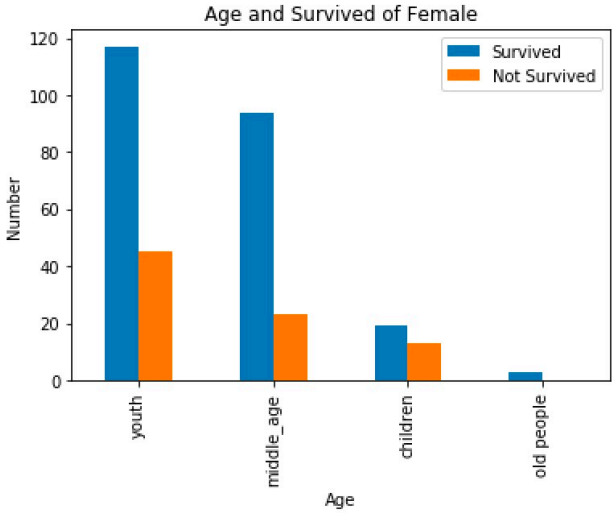
```
bins = [0,12,30,60,100]
train['age_class'] = pd.cut(train.Age,bins = bins,labels=['children','youth','middle_age',
su_1 = train.age_class[train.Survived == 1].value_counts()
su_0 = train.age_class[train.Survived == 0].value_counts()
data0 = pd.DataFrame({'Survived':su_1,'Not Survived':su_0})
data0.plot(kind = 'bar')
plt.title('The relationship between age and survival')
plt.xlabel('age')
```

```
Text(0.5, 0, 'age')
```



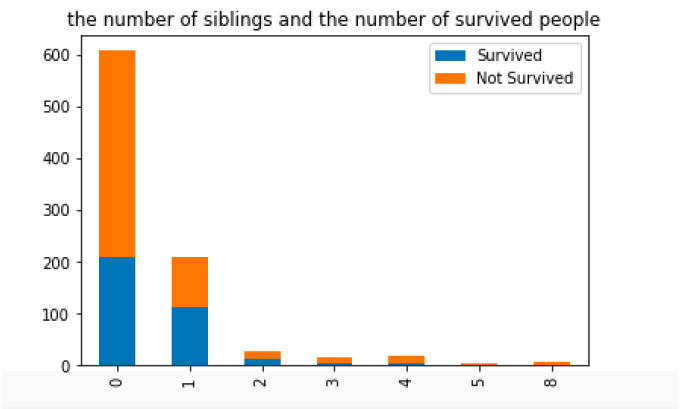
If I combine the age and sex, I found the number of survived female are more than the number of dead female, especially in middle age female and youth female. For male, the number of survived children are more than others.

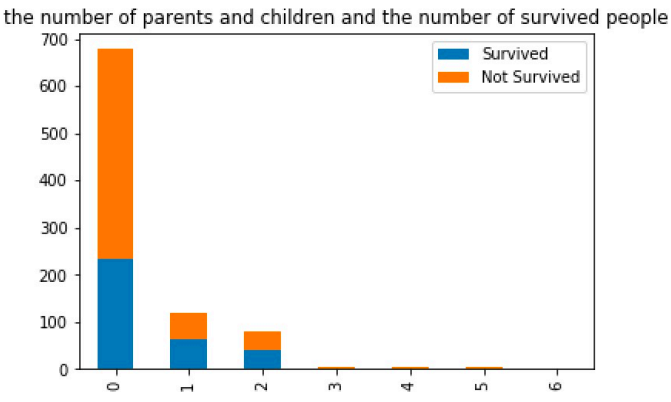




‘Sibsp’: Explore the relationship between the number of survived and the number of their siblings, I found people with more than 3 siblings had less opportunity to be survived. But it's not obvious to say that more siblings, less survival, because sample is small. Meanwhile, the relationship among people who has less than 3 siblings is not obvious, too.

‘Parch’: Analyze the relationship between the number of survived people and the number of parents and children, I found the result is like the relationship between the number of survived people and the number of siblings. The relationship is not obvious.

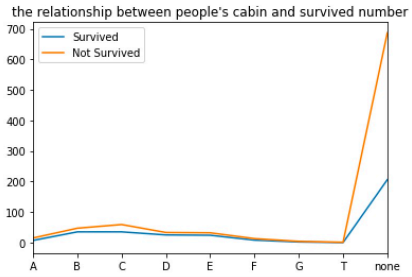




‘Cabin’: ‘Cabin’ feature is composed by letter and number. I assumed the letter represents cabin number and the number represents the seat number.Then I extract the cabin number from this feature. I found there are 687 people don't have a cabin number, so I fill this missing value with 'none'.

```
surviv_1 = train.cabin_title[train.Survived == 1].value_counts()
surviv_0 = train.cabin_title[train.Survived == 0].value_counts()
data8 = pd.DataFrame({'Survived':surviv_1,'Not Survived':surviv_0})
data8.plot(kind = 'line',stacked=True)
plt.title("the relationship between people's cabin and survived number")
```

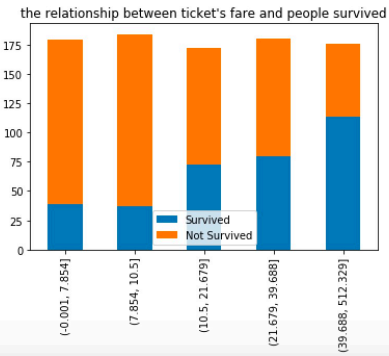
```
Text(0.5, 1.0, "the relationship between people's cabin and survived number")
```



‘Fare’: Divide the passengers into four group evenly according to the fare. People with high price's ticket are more likely to be survived.

```
train['fare_class'] = pd.qcut(train.Fare,q=5)
s_1 = train.fare_class[train.Survived == 1].value_counts()
s_0 = train.fare_class[train.Survived == 0].value_counts()
data10 = pd.DataFrame({'Survived':s_1,'Not Survived':s_0})
data10.plot(kind = 'bar',stacked=True)
plt.title("the relationship between ticket's fare and people survived")
```

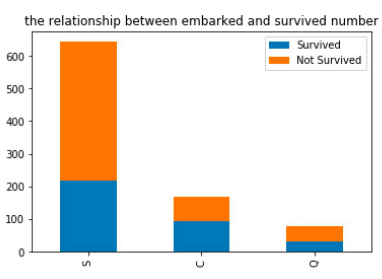
```
Text(0.5, 1.0, "the relationship between ticket's fare and people survived")
```



‘Embarked’: The number of survived people embarked at S is less than dead people. In port C and port Q.

```
survive_1 = train.Embarked[train.Survived == 1].value_counts()
survive_0 = train.Embarked[train.Survived == 0].value_counts()
data9 = pd.DataFrame({'Survived':survive_1,'Not Survived':survive_0})
data9.plot(kind = 'bar',stacked=True)
plt.title("the relationship between embarked and survived number")
```

```
Text(0.5, 1.0, 'the relationship between embarked and survived number')
```



‘Ticket’: There are 681 unique tickets. With more details, 547 tickets are used by one people and 94 tickets are used by two people. We have no idea why they use same tickets, maybe because there are family.

```
a,b = np.unique(data6.number,return_counts=True)
data7 = pd.DataFrame(zip(a,b))
data7.columns = [['ticket','count']]
data7
```

	ticket	count
0	1	547
1	2	94
2	3	21
3	4	11
4	5	2
5	6	3
6	7	3

3. Feature Engineering

Did same operation on both train set and test set.

1)Dealing with missing values

Age has more than 150 missing values and I can't ignore the missing value. It is not a good idea to fill them with the mean of age. By observing the dataset, I found the attribute 'name' contains some title for people, like 'Miss','Mr'. Extract the title and calculate each title's average age, then fill the missing values with corresponding average age. The train set has 891 passengers’ information and the test set has 418 passengers’ info. I concat the train and test set together to get all age and name. By using more data, we will have more accuracy.

```
df1 = pd.DataFrame({'Name':train['Name'],'Age':train['Age']})
df2 = pd.DataFrame({'Name':test['Name'],'Age':test['Age']})
df3 = pd.concat([df1,df2])
df3['title'] = df3['Name'].str.extract(' ([A-Za-z]+)\.',expand=False)
mean_ages = pd.DataFrame(df3.groupby(df3.title)['Age'].mean())
mean_ages.rename(columns = {"Age": "mean_ages"},  inplace=True)
train = pd.merge(train,mean_ages,left_on='title',right_index=True,how='left')
train.loc[ (train.Age.isnull()), 'Age' ] = train['mean_ages']
```

People without cabin number have less chance to survived. I assume that these people are poor and has worse cabin, so it's more difficult for them to escape. People with cabin numbers have no obvious difference between survived and

not survived. Thus, we can consider this feather into 2 situations: with cabin number(1) and without(0).

```
train.loc[ (train.cabin_title =='none'), 'cabin_status' ] = 0
train.loc[ (train.cabin_title != 'none'), 'cabin_status' ] = 1
```

2) Create new feature

I created a new feature ‘family’, which is the sum of ‘Parch’(of parents / children aboard the Titanic) and ‘SibSp’(of siblings / spouses aboard the Titanic). It's not obvious to say the relationship between the number of family and the number of survived people. So I temporarily ignored this feature.

3) Change feature’s datatype.

Some features may be string, so we should transform these features’ type into numeric.

```
sex_dummies  = pd.get_dummies(train_set['Sex'])
train_set = train_set.join(sex_dummies)
train_set.drop(['Sex'], axis=1,inplace=True)
```

```
embarked_dummies  = pd.get_dummies(train['Embarked'])
train = train.join(embarked_dummies)
train.drop(['Embarked'], axis=1,inplace=True)
```

4) Feature scaled

The range of age and the range of fare is very different. In order to improve accuracy, we use feature scaling to standard the data.

```
std = StandardScaler()
train_set['Age_scaled'] = std.fit_transform(train_set.Age.values.reshape(-1,1))
train_set['Fare_scaled'] = std.fit_transform(train_set.Fare.values.reshape(-1,1))
```

4.modeling

I split the train set. I used part of the train set(20% in this case) to test the model.

```
from sklearn.model_selection import train_test_split

x_train, x_val, y_train, y_val = train_test_split(predictors, target, test_size = 0.20, random_state = 0)
```

Gaussian Bayes:

```
# Gaussian Bayes
```

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

gaussian = GaussianNB()
gaussian.fit(x_train, y_train)
y_pred = gaussian.predict(x_val)
acc_gaussian = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_gaussian)
```

```
79.89
```

Logistic Regression:

```
# logistic regression
```

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_val)
acc_logreg = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_logreg)
```

```
82.12
```

Linear SVC:

```
# logistic regression
```

```
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
logreg.fit(x_train, y_train)
y_pred = logreg.predict(x_val)
acc_logreg = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_logreg)
```

```
82.12
```

Decision Tree:

```
# decision tree
```

```
from sklearn.tree import DecisionTreeClassifier

decisiontree = DecisionTreeClassifier()
decisiontree.fit(x_train, y_train)
y_pred = decisiontree.predict(x_val)
acc_decisiontree = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_decisiontree)
```

```
74.3
```

SVM:

```
#SVM
```

```
from sklearn.svm import SVC

svc = SVC()
svc.fit(x_train, y_train)
y_pred = svc.predict(x_val)
acc_svc = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_svc)
```

```
84.36
```

Random Forest:

```
# random forest
```

```
from sklearn.ensemble import RandomForestClassifier

randomforest = RandomForestClassifier()
randomforest.fit(x_train, y_train)
y_pred = randomforest.predict(x_val)
acc_randomforest = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_randomforest)
```

```
80.45
```

Gradient Boosting Classifier:

```
# Gradient Boosting Classifier
```

```
from sklearn.ensemble import GradientBoostingClassifier

gbk = GradientBoostingClassifier()
gbk.fit(x_train, y_train)
y_pred = gbk.predict(x_val)
acc_gbk = round(accuracy_score(y_pred, y_val) * 100, 2)
print(acc_gbk)
```

```
79.89
```

5.Cross Validation

I use 10-fold cross validation and the SVM model has highest mean. I decided to use SVM model.

```
from sklearn.model_selection import KFold #for K-fold cross validation
from sklearn.model_selection import cross_val_score #score evaluation
from sklearn.model_selection import cross_val_predict #prediction
```

```
kfold = KFold(n_splits=10, random_state=22) # k=10, split the data into 10 equal parts
xyz=[]
accuracy=[]
std=[]
classifiers=['Support Vector Machines', 'Logistic Regression',
             'Random Forest', 'Naive Bayes', 'Linear SVC',
             'Decision Tree',  'Gradient Boosting Classifier']
models=[SVC(), LogisticRegression(), RandomForestClassifier(), GaussianNB(),
        LinearSVC(), DecisionTreeClassifier(),
        GradientBoostingClassifier()]
for i in models:
    model = i
    cv_result = cross_val_score(model,X,y, cv = kfold,scoring = "accuracy")
    xyz.append(cv_result.mean())
    std.append(cv_result.std())
    accuracy.append(cv_result)
new_models_dataframe2=pd.DataFrame({'CV Mean':xyz,'Std':std},index=classifiers)
new_models_dataframe2
```

	CV Mean	Std
Support Vector Machines	0.806891	0.067322
Logistic Regression	0.783296	0.058821
Random Forest	0.778851	0.064099
Naive Bayes	0.775568	0.047208
Linear SVC	0.779913	0.069492
Decision Tree	0.755318	0.043520
Gradient Boosting Classifier	0.802422	0.055677

Use SVM to predict and write the result into a csv file

```
y_pred2 = svm.predict(X_test)
```

```
result = pd.DataFrame({'PassengerId':test['PassengerId'].as_matrix(), 'Survived':y_pred2.astype(np.int32)})
```

6.Result


3359
xiang liu97
0.78947
10
25d


Titanic: Machin...
Ongoing
Top 21%
3,359th
of 16467