ADS 245 Project - Titanic Data

Name: WeiTing Huang

SJSU ID: 014550315

Import library

In [1]:
```
# first, import calculation, visualization library
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import seaborn as sns
from collections import Counter
```

1. Exploratory Data Analysis

In [2]:
```
train = pd.read_csv('titanic_train.csv')
test = pd.read_csv('titanic_test.csv')
# survived is the target feature
```

In [3]:
```
train.isnull().sum()
# Check if there is any null column in the dataset
# three columns have missing value: Age, Cabin, Embarked
```

Out[3]:
```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

In [4]:
```
test.isnull().sum()
# three columns have missing value: Age, Cabin, Fare
```

Out[4]:
```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

In [5]:
```
train.head()
```

Out[5]:

	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	7.2500
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500

In [6]:
```
test.head()
```

Out[6]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emb
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 [7]:
```
train.describe()
```

Out[7]:

	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 [8]:
```
test.describe()
```

Out[8]:

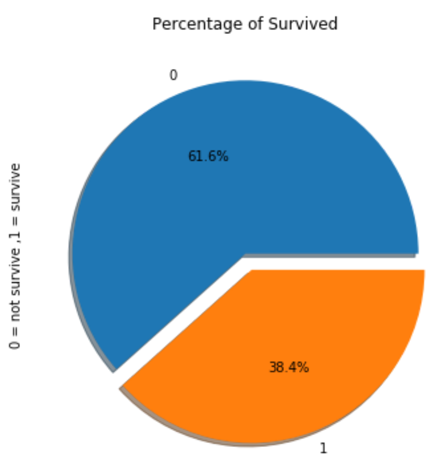
	PassengerId	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	3.000000	76.000000	8.000000	9.000000	512.329200

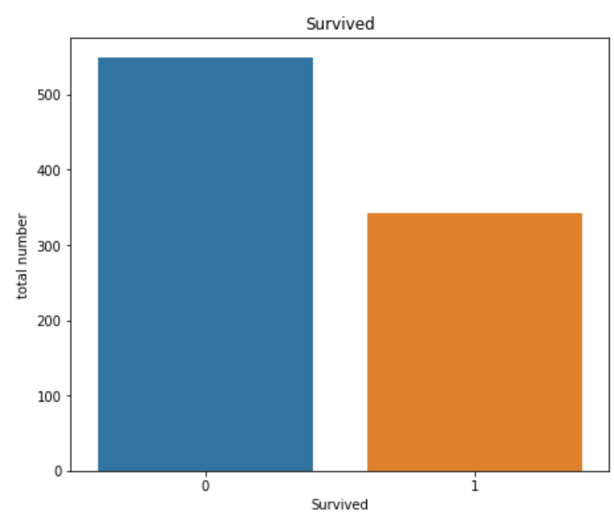
1.1 Target feature

In [9]:
```
# Target variable for this dataset is Survived. Let's check if there is any i

f,ax=plt.subplots(1,2,figsize=(16,6))

train['Survived'].value_counts().plot.pie(explode=[0,0.1],autopct='%1.1f%%',a
ax[0].set_title('Percentage of Survived')
ax[0].set_ylabel('0 = not survive ,1 = survive')
sns.countplot('Survived',data=train,ax=ax[1])
ax[1].set_title('Survived')
ax[1].set_ylabel('total number')
plt.show()
```





From total 891 passengers in training set, around 350 survived. As the pie chart showed, Only 38.4% of the total training set survived after the crash.

1.2 Feature Engineering

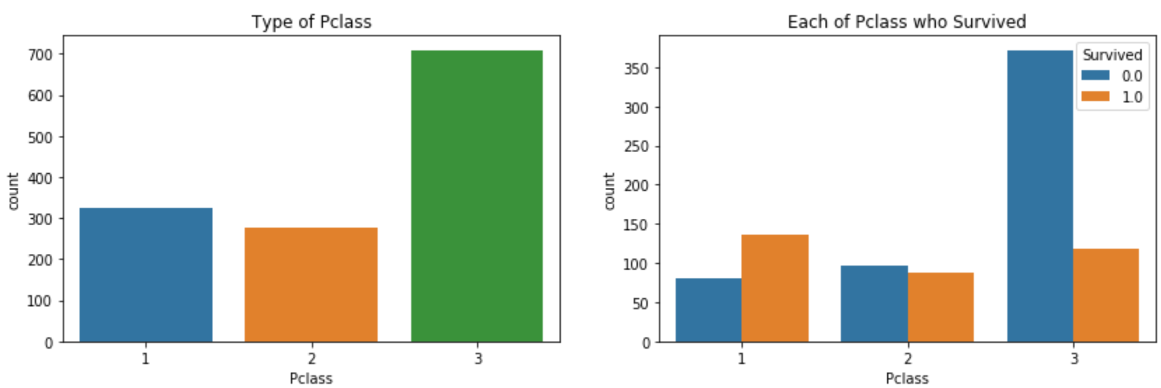
In [10]:
```
# analysis & data cleaning on this field first
# combine train and test to a dataset which will be easier to motify
ds = pd.concat([train, test] , sort=False)
```

1.2.1 PClass

In [11]:
```
f,ax = plt.subplots(1,2,figsize=(14,4))

sns.countplot('Pclass',data=ds,ax=ax[0])
ax[0].set_title('Type of Pclass')
ax[0].set_ylabel('count')

sns.countplot('Pclass',hue ='Survived',data=ds,ax=ax[1])
ax[1].set_title('Each of Pclass who Survived')
ax[1].set_ylabel('count')
plt.show()
```



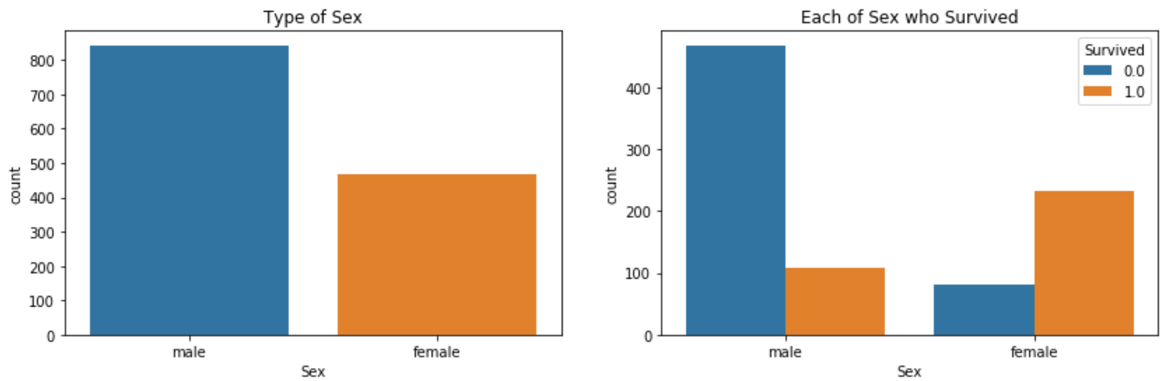
We can see that Passenegers Of Pclass 1 have higher chance to secure. Although the the number of passengers in Pclass 3 were higher, the number of survival from Pclass 3 is low.

1.2.2 Sex

In [12]:
```
f,ax = plt.subplots(1,2,figsize=(14,4))

sns.countplot('Sex',data=ds,ax=ax[0])
ax[0].set_title('Type of Sex')
ax[0].set_ylabel('count')

sns.countplot('Sex',hue ='Survived',data=ds,ax=ax[1])
ax[1].set_title('Each of Sex who Survived')
ax[1].set_ylabel('count')
plt.show()
```



Sex is a categorical Feature with two type(male/female). We can found that female survived is higher than male. Lets dive in to check survival rate with Pclass and sex together.

1.2.3 PClass and Sex

In [13]:
```
ds.loc[:,'Pclass1and2Female'] = 0
ds.loc[:,'Pcalss3Male'] = 0

ds.loc[(ds['Pclass']<=2) & (ds['Sex']=='female'), 'Pclass1and2Female'] = 1
ds.loc[(ds['Pclass']==3) & (ds['Sex']=='male'), 'Pcalss3Male'] = 1
```

Create two new column to mark female who in Pclass 1 & 2, who are indicate rich women. Also, mark make who in Pcalss 3, who refer to poor men.

1.2.4 Name

In [14]:
```
ds['LastName'] = ds['Name'].str.split(',', expand=True)[0]
```

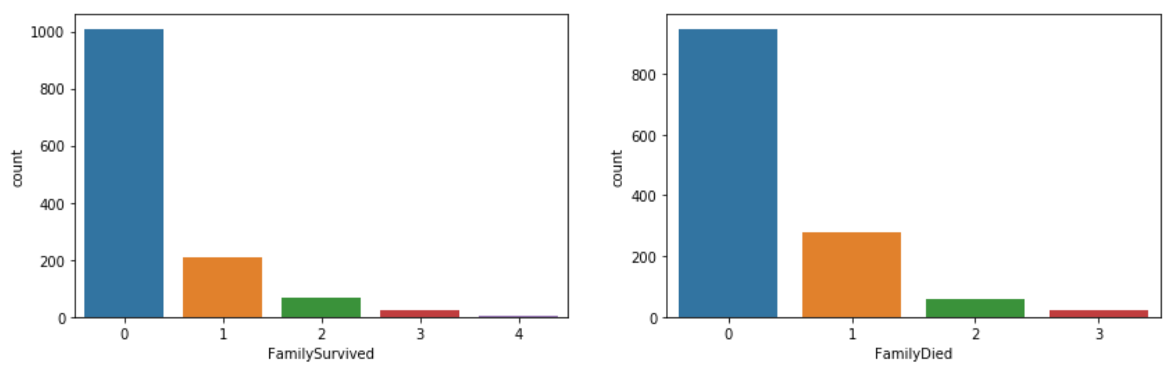
In [15]:
```
sur = []
died = []
for index, row in ds.iterrows():
    s = ds[(ds['LastName']==row['LastName']) & (ds['Survived']==1)]
    d = ds[(ds['LastName']==row['LastName']) & (ds['Survived']==0)]
    s=len(s)
    if row['Survived'] == 1:
        s-=1
    d=len(d)
    if row['Survived'] == 0:
        d-=1
    sur.append(s)
    died.append(d)

ds['FamilySurvived'] = sur
ds['FamilyDied'] = died
```

In [16]:
```
# mapping family died to 4 bins
ds.loc[ ds['FamilyDied'] == 0, 'FamilyDied'] = 0
ds.loc[(ds['FamilyDied'] > 0) & (ds['FamilyDied'] <= 2), 'FamilyDied'] = 1
ds.loc[(ds['FamilyDied'] > 2) & (ds['FamilyDied'] <= 5), 'FamilyDied'] = 2
ds.loc[(ds['FamilyDied'] > 5), 'FamilyDied'] = 3
```

Use last name of name to check whether passanger in their family have survived.

In [17]:
```
f,ax = plt.subplots(1,2,figsize=(14,4))
sns.countplot('FamilySurvived',data=ds,ax=ax[0])
sns.countplot('FamilyDied',data=ds,ax=ax[1])
plt.show()
```

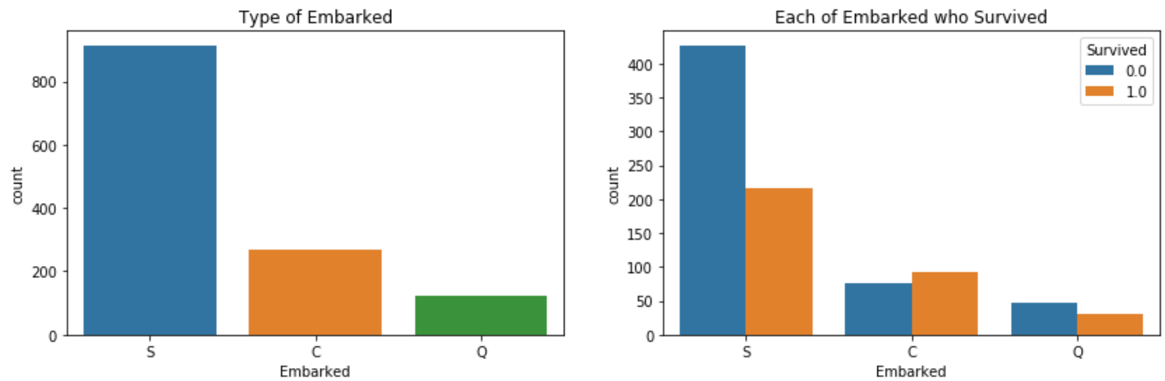


1.2.5 Embarked

In [18]:
```python
f,ax = plt.subplots(1,2,figsize=(14,4))

sns.countplot('Embarked',data=ds,ax=ax[0])
ax[0].set_title('Type of Embarked')
ax[0].set_ylabel('count')

sns.countplot('Embarked',hue ='Survived',data=ds,ax=ax[1])
ax[1].set_title('Each of Embarked who Survived')
ax[1].set_ylabel('count')
plt.show()
```



In [19]:
```python
ds['Embarked'] = ds['Embarked'].fillna('S')
```

We can see that the most common embarded type is S. Becasue there are 2 missing data in Embarded feature, we will just fill the most common one which is S type.

1.2.6 Fare

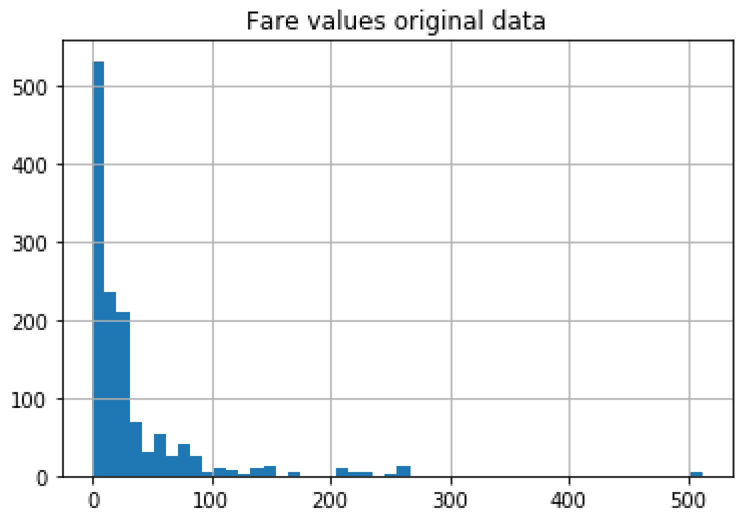
In [20]:
```python
#There is only one missing data in Fare, so fill it as a median vlaue
ds['Fare'] = ds['Fare'].fillna(train['Fare'].median())
```

In [21]:
```
print('Max of Fare',ds['Fare'].max())
print('Min of Fare',ds['Fare'].min())
print('Mean of Fare',ds['Fare'].mean())
print('Mode of Fare',ds['Fare'].mode())
print('Standard diviation of Fare',ds['Fare'].std())
```

```
Max of Fare 512.3292
Min of Fare 0.0
Mean of Fare 33.28108563789156
Mode of Fare 0    8.05
dtype: float64
Standard diviation of Fare 51.74149976752607
```

In [22]:
```
ds['Fare'].astype(int).hist(bins=50).set_title('Fare values original data')
```

Out[22]:
```
Text(0.5, 1.0, 'Fare values original data')
```



By hist graph, Fare featrue is a left skew distribution. Fare is also a continous feature that need to convert it into ordinal value.We use pandas qcut to splits to 4 bins.

In [23]:
```
ds['Fare_bin']=pd.qcut(ds['Fare'],4)
ds.groupby(['Fare_bin'])['Survived'].mean().to_frame()
```

Out[23]:

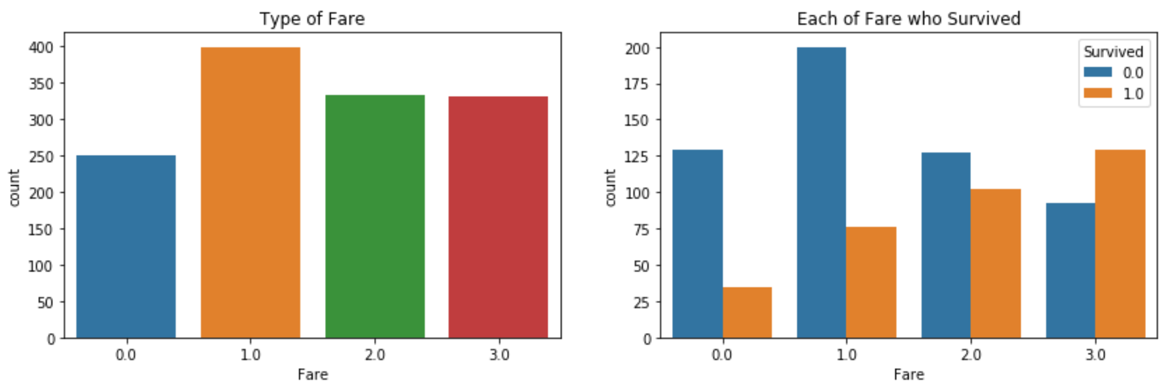
	Survived
Fare_bin	
(-0.001, 7.896]	0.197309
(7.896, 14.454]	0.303571
(14.454, 31.275]	0.441048
(31.275, 512.329]	0.600000

In [24]:
```
# Mapping Fare to fare_bin
ds.loc[(ds['Fare'] <= 7.8), 'Fare'] = 0
ds.loc[(ds['Fare'] > 7.8) & (ds['Fare'] <= 14.454), 'Fare'] = 1
ds.loc[(ds['Fare'] > 14.454) & (ds['Fare'] <= 31.27), 'Fare']   = 2
ds.loc[(ds['Fare'] > 31.27), 'Fare'] = 3
#ds['Fare'] = ds['Fare'].astype(int)
```

In [25]:
```
f,ax = plt.subplots(1,2,figsize=(14,4))

sns.countplot('Fare',data=ds,ax=ax[0])
ax[0].set_title('Type of Fare')
ax[0].set_ylabel('count')

sns.countplot('Fare',hue ='Survived',data=ds,ax=ax[1])
ax[1].set_title('Each of Fare who Survived')
ax[1].set_ylabel('count')
plt.show()
```



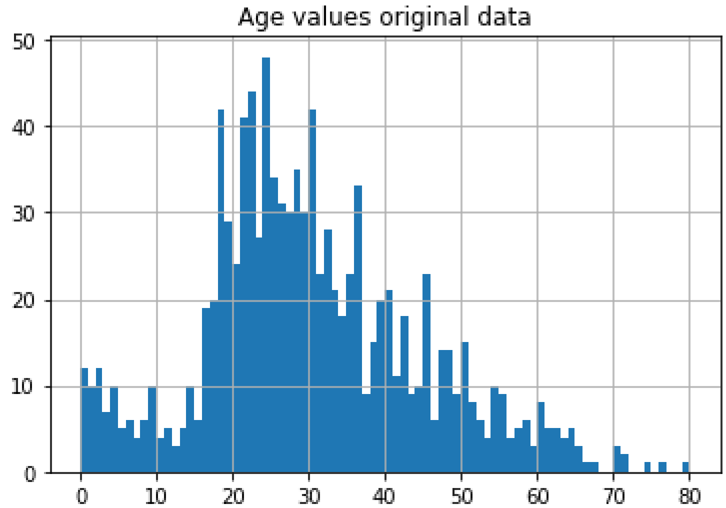
1.2.7 Age

In [26]:
```
print('Age of oldest Passenger:',ds['Age'].max())
print('Age of youngest Pasenger',ds['Age'].min())
```

```
Age of oldest Passenger: 80.0
Age of youngest Pasenger 0.17
```

In [27]:
```
f = ds['Age'].dropna().astype(int).hist(bins=80)
f.set_title('Age values original data')
```

Out[27]: `Text(0.5, 1.0, 'Age values original data')`



As we had seen earlier, Age feature has 177(train)+ 86(test) null values. To replace these NaN values, we need to observe age of max/min/mean to understand the range of age. From the histrogrm graph, we can assume age feature is normal distribution. Then, we can assgin the random value in within +1/-1 segma range. Becasue age is a continous feature, we need to use binning to catergrize.

In [28]:
```
# fill random vlaue between +1/-1 segma range.
age_avg = ds['Age'].mean()
age_std = ds['Age'].std()
age_null = ds['Age'].isnull().sum()
age_null_random_list = np.random.randint(age_avg - age_std, age_avg + age_stc
```

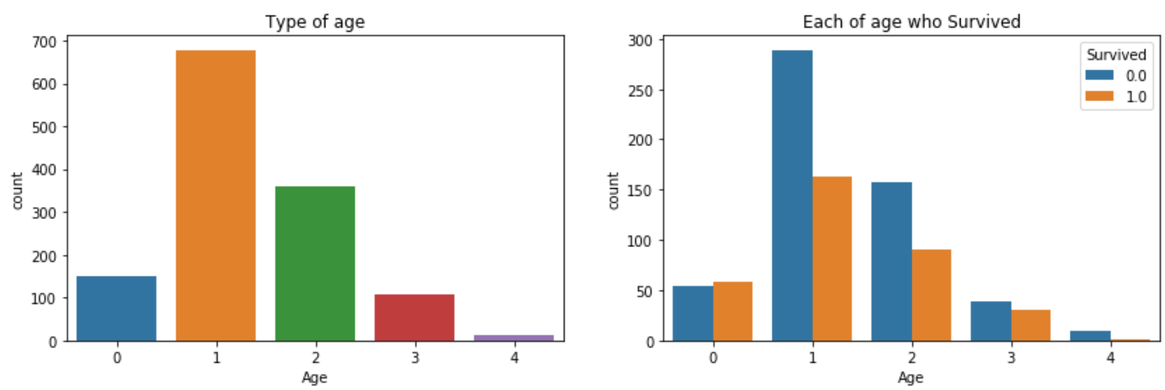
In [29]:
```
ds.loc[np.isnan(ds['Age']),'Age'] = age_null_random_list
ds.loc[:,'Age'] = ds['Age'].astype(int)

# Threshold of each bins : (80-0)/5 = 16
# Mapping Age
ds.loc[ ds['Age'] <= 16, 'Age'] = 0
ds.loc[(ds['Age'] > 16) & (ds['Age'] <= 32), 'Age'] = 1
ds.loc[(ds['Age'] > 32) & (ds['Age'] <= 48), 'Age'] = 2
ds.loc[(ds['Age'] > 48) & (ds['Age'] <= 64), 'Age'] = 3
ds.loc[ ds['Age'] > 64, 'Age'] = 4
```

In [30]:
```
f,ax = plt.subplots(1,2,figsize=(14,4))

sns.countplot('Age',data=ds,ax=ax[0])
ax[0].set_title('Type of age')
ax[0].set_ylabel('count')

sns.countplot('Age',hue ='Survived',data=ds,ax=ax[1])
ax[1].set_title('Each of age who Survived')
ax[1].set_ylabel('count')
plt.show()
```



1.2.8 Cabin

In [31]:
```
ds.loc[:,'Cabin'] = pd.Series([1 if not pd.isnull(i) else 0 for i in ds['Cabi
```

Since there are more than 60% of Cabin featrue is missing, we can't fill values by reference other. I will only mark people who have cabin in this trip.

1.2.9 SibSp and Parch

Create new columns called "Family_size" and "Alone". By calculate Parch and SibSp columns, we can know family size of the passengers.

In [32]:
```
ds['FamilySize'] = ds['SibSp'] + ds['Parch'] + 1
ds['IsAlone'] = 0
ds.loc[ds['FamilySize'] == 1, 'IsAlone'] = 1
```

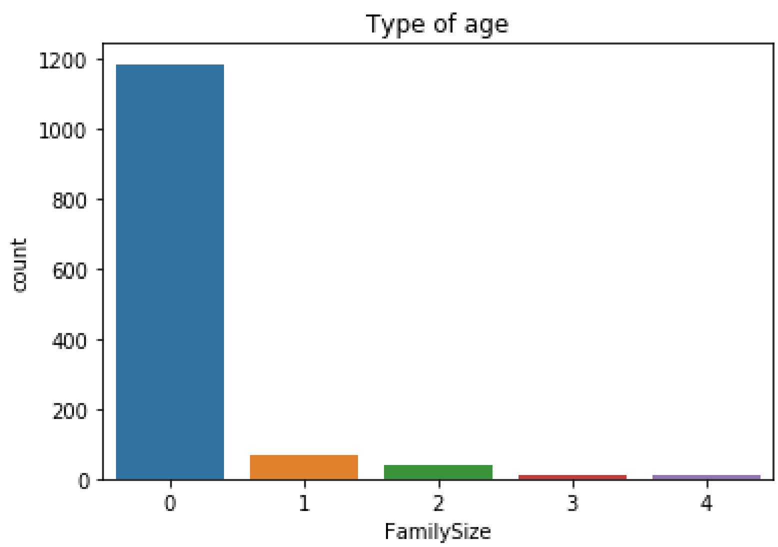
In [33]:
```
print('Max of family size',ds['FamilySize'].max())
print('Min of family size',ds['FamilySize'].min())
```

```
Max of family size 11
Min of family size 1
```

In [34]:
```
# binning of family size, use 5 bins
# (11-1) / 5 = 2.
ds.loc[ ds['FamilySize'] <= 3, 'FamilySize'] = 0
ds.loc[(ds['FamilySize'] > 3) & (ds['FamilySize'] <= 5), 'FamilySize'] = 1
ds.loc[(ds['FamilySize'] > 5) & (ds['FamilySize'] <= 7), 'FamilySize'] = 2
ds.loc[(ds['FamilySize'] > 7) & (ds['FamilySize'] <= 9), 'FamilySize'] = 3
ds.loc[ ds['FamilySize'] > 9, 'FamilySize'] = 4
```

In [35]:
```
sns.countplot('FamilySize',data=ds).set_title('Type of age')
```

Out[35]:
```
Text(0.5, 1.0, 'Type of age')
```



In [36]:
```
ds.head()
```

Out[36]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Ca
0	1	0.0	3	Braund, Mr. Owen Harris	male	1	1	0	A/5 21171	0.0	
1	2	1.0	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	2	1	0	PC 17599	3.0	
2	3	1.0	3	Heikkinen, Miss. Laina	female	1	0	0	STON/O2. 3101282	1.0	
3	4	1.0	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	2	1	0	113803	3.0	
4	5	0.0	3	Allen, Mr. William Henry	male	2	0	0	373450	1.0	

1.3 One hot Encoding - Categorical data

In [37]:
```
categorical = ['Pclass', 'Sex', 'Embarked', 'Age', 'Fare','FamilySize','Famil
for c in categorical:
    ds = pd.concat([ds, pd.get_dummies(ds[c], prefix=c)], axis=1)
    ds = ds.drop([c], axis=1)
```

In [38]:
```
ds = ds.drop(['PassengerId', 'Ticket', 'LastName', 'SibSp', 'Parch','Name','F
```

In [39]:
```
train = ds[ds['Survived'].notnull()]
test = ds[ds['Survived'].isnull()]
test = test.drop(['Survived'], axis=1)
```

In [40]:
```
train.head()
```

Out[40]:

	Survived	Cabin	Pclass1and2Female	Pcalss3Male	IsAlone	Pclass_1	Pclass_2	Pclass_3
0	0.0	0	0	1	0	0	0	1
1	1.0	1	1	0	0	1	0	0
2	1.0	0	0	0	1	0	0	1
3	1.0	1	1	0	0	1	0	0
4	0.0	0	0	1	1	0	0	1

5 rows × 35 columns

In [41]:
```
test.head()
```

Out[41]:

	Cabin	Pclass1and2Female	Pcalss3Male	IsAlone	Pclass_1	Pclass_2	Pclass_3	Sex_femal
0	0	0	1	1	0	0	1	
1	1	0	0	0	0	0	1	
2	0	0	0	1	0	1	0	
3	1	0	1	1	0	0	1	
4	0	0	0	0	0	0	1	

5 rows × 34 columns

In [42]:
```
train.to_csv('train.csv')
test.to_csv('test.csv')
#load to csv file to double check the data.
```

1.4 Create a sub-test group to test accuracy rate

In [43]:
```
X_train = np.array(train.iloc[:800,1:])
y_train = np.array(train.iloc[:800,0])
```

In [44]:
```
X_test = np.array(train.iloc[800:,1:])
y_test = np.array(train.iloc[800:,0])
```

In [45]:
```
submit_test = np.array(test)
```

2. KNN model

2.1 Create KNN class

In [46]:
```
class KNearestNeighbor(object):

    def __init__(self):
        pass
    def train(self, X, y):
        self.X_train = X
        self.y_train = y

    #predict compute distances and predict labels
    def predict(self, X, k=1):
        dists = self.compute_distances(X)
        return self.predict_labels(dists, k=k)

    def compute_distances(self, X):
        num_test = X.shape[0]
        num_train = self.X_train.shape[0]
        dists = np.zeros((num_test, num_train))
        dists = np.sqrt(np.sum(X**2, axis=1).reshape(num_test, 1) + np.sum(se
        return dists

    def predict_labels(self, dists, k=1):
        num_test = dists.shape[0]
        y_pred = np.zeros(num_test)
        for i in range(num_test):
            closest_y = []
            top_k_indx = np.argsort(dists[i])[:k]
            closest_y = self.y_train[top_k_indx]
            vote = Counter(closest_y)
            count = vote.most_common()
            y_pred[i] = count[0][0]
        return y_pred
```

2.2 Test KNN model

In [71]:
```
classifier = KNearestNeighbor()
```

In [72]:
```
classifier.train(X_train, y_train)
```

In [73]:
```
dists = classifier.compute_distances(X_test)
```

In [74]:
```
y_test_pred = classifier.predict_labels(dists, k=5)
# try accuacy rate here, and came out k = 5 has best result
```

In [75]:
```
num_test = y_test.shape[0]
num_correct = np.sum(y_test_pred == y_test)
accuracy = float(num_correct) / num_test
print('Got %d / %d correct => accuracy: %f' % (num_correct, num_test, accurac
```

```
Got 74 / 91 correct => accuracy: 0.813187
```

2.3 Predict test dataset

In [76]:
```
classifier_submit = KNearestNeighbor()
```

In [77]:
```
classifier_submit.train(np.array(train.iloc[:,1:]), np.array(train.iloc[:,0])
```

In [78]:
```
dists = classifier_submit.compute_distances(submit_test)
```

In [79]:
```
y_submit_pred = classifier_submit.predict_labels(dists, k=5)
```

In [80]:
```
passengerid = pd.read_csv('gender_submission.csv')
```

In [81]:
```
y_submit = pd.DataFrame(data=y_submit_pred, columns=['Survived'])
```

In [82]:
```
df = [passengerid['PassengerId'], y_submit]
result = pd.concat(df,axis=1)
result.to_csv('submisson.csv', index=False)
```

3. Logistic Regression

3.1 Create LR class

In [59]:
```
class LogisticRegression(object):

    def __init__(self):
        pass
    def train(self,X,y):
        self.X_train = X
        self.y_train = y

    def sigmoid(self,z):
        s = 1/(1+np.exp(-z))
        return s

    def initilialize_with_zeros(self,dim):
        w = np.zeros((dim,1))
        b = 0
        return w,b

    def propagate(self,w,b):
        # Forward Propagation
        m = self.X_train.shape[1]
        A = self.sigmoid(np.dot(w.T,self.X_train.T) + b)
        cost = (-1/m)*(np.sum(self.y_train*np.log(A) + (1-self.y_train)*np.lo

        # Backward Propagation
        dw = 1/m*(np.dot(self.X_train.T, (A-self.y_train).T))
        db = 1/m*np.sum(A-self.y_train)

        grads = {'dw':dw, 'db':db}
        return grads, cost

    def optimize(self,w,b,num_iter, learning_rate, print_cost=False):
        costs = []
        for i in range(num_iter):
            grads, cost = self.propagate(w,b)
            dw = grads['dw']
            db = grads['db']

            w = w - dw * learning_rate
            b = b - db * learning_rate

            if i % 100 == 0:
                costs.append(cost)
            if print_cost and i % 100 == 0:
                print('Cost after iteration %i: %f' %(i,cost))

        params = {'w':w,'b':b}
        grads = {'dw':dw,'db':db}

        return params, grads, costs

    def predict(self,w,b,X):
        m = X.shape[0]
        Y_prediction = np.zeros((1,m))
        w = w.reshape(X.shape[1],1)
        A = self.sigmoid(np.dot(w.T,X.T)+b)
```

```
        for i in range(A.shape[1]):
            if A[0,i] > 0.5:
                Y_prediction[0,i] = 1
            else:
                Y_prediction[0,i] = 0

        return Y_prediction

    def model(self, X_test = None, y_test = None, num_iterations = 100000, le
        w, b = self.initilialize_with_zeros(self.X_train.shape[1])
        parameters, grads, costs = self.optimize(w, b, num_iterations, learni
        w = parameters["w"]
        b = parameters["b"]
        Y_prediction_test = self.predict(w, b, X_test)
        Y_prediction_train = self.predict(w, b, self.X_train)
        print("train accuracy: {} %".format(100 - np.mean(np.abs(Y_predictior
        if y_test is not None:
            print("test accuracy: {} %".format(100 - np.mean(np.abs(Y_predict

        d = {"costs": costs,
        "Y_prediction_test": Y_prediction_test,
        "Y_prediction_train" : Y_prediction_train,
        "w" : w,
        "b" : b,
        "learning_rate" : learning_rate,
        "num_iterations": num_iterations}

        return d
```

3.2 Test LR model

In [60]:

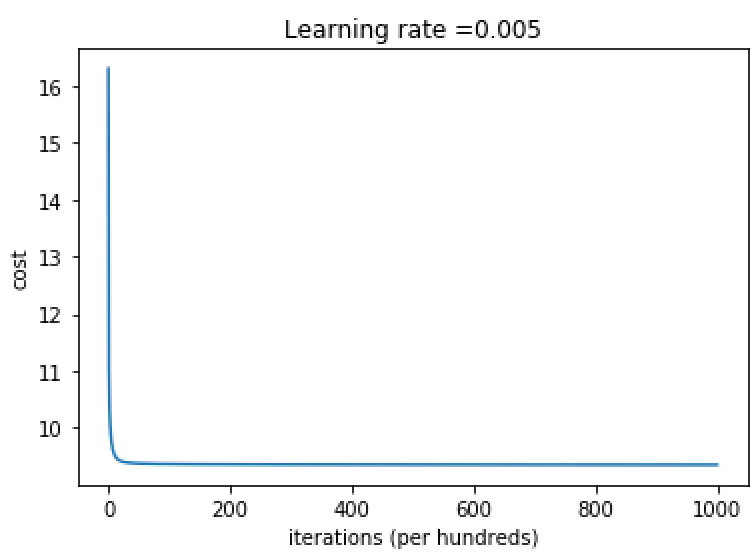
```
classifier = LogisticRegression()
```

In [61]:

```
classifier.train(np.array(train.iloc[:800,1:]), np.array(train.iloc[:800,0]))
```

In [62]:
```
d = classifier.model(np.array(train.iloc[800:,1:]), np.array(train.iloc[800:,
```
```
Cost after iteration 84100: 9.350703
Cost after iteration 84200: 9.350699
Cost after iteration 84300: 9.350695
Cost after iteration 84400: 9.350691
Cost after iteration 84500: 9.350687
Cost after iteration 84600: 9.350683
Cost after iteration 84700: 9.350679
Cost after iteration 84800: 9.350675
Cost after iteration 84900: 9.350671
Cost after iteration 85000: 9.350667
Cost after iteration 85100: 9.350664
Cost after iteration 85200: 9.350660
Cost after iteration 85300: 9.350656
Cost after iteration 85400: 9.350652
Cost after iteration 85500: 9.350648
Cost after iteration 85600: 9.350644
Cost after iteration 85700: 9.350640
Cost after iteration 85800: 9.350636
Cost after iteration 85900: 9.350632
Cost after iteration 86000: 9.350629
```

In [63]:
```
# Plot learning curve (with costs)
costs = np.squeeze(d['costs'])
plt.plot(costs)
plt.ylabel('cost')
plt.xlabel('iterations (per hundreds)')
plt.title("Learning rate =" + str(d["learning_rate"]))
plt.show()
```



3.3 Predict test dataset

In [83]:
```
classifier_submission = LogisticRegression()
```

In [84]:
```
classifier_submission.train(np.array(train.iloc[:,1:]), np.array(train.iloc[:
```

In [85]:
```
d = classifier_submission.model(submit_test, print_cost = True)
```

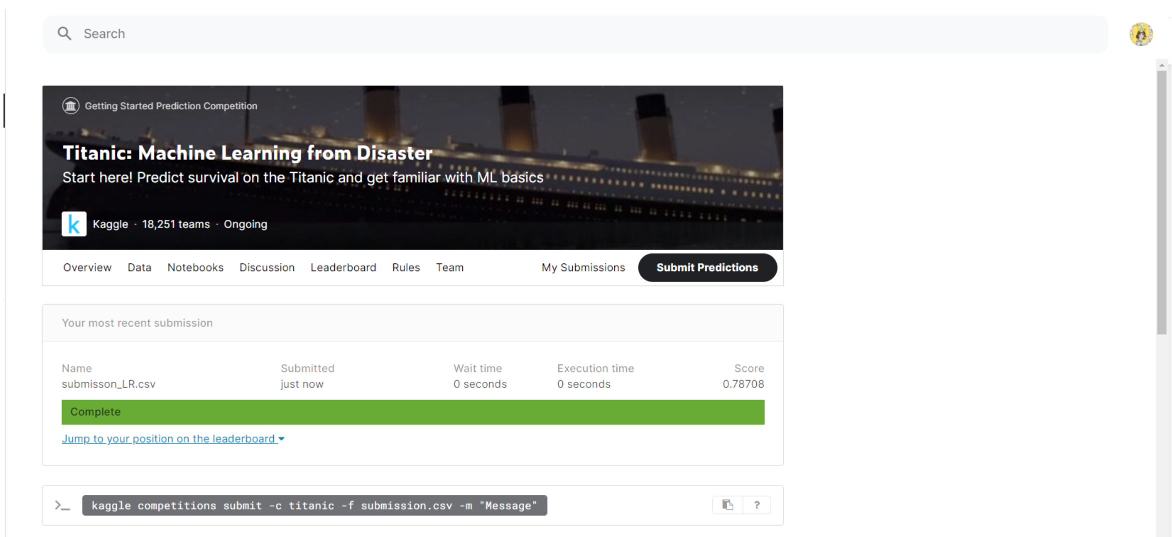
```
Cost after iteration 82500: 10.257049
Cost after iteration 82600: 10.257041
Cost after iteration 82700: 10.257033
Cost after iteration 82800: 10.257025
Cost after iteration 82900: 10.257016
Cost after iteration 83000: 10.257008
Cost after iteration 83100: 10.257000
Cost after iteration 83200: 10.256992
Cost after iteration 83300: 10.256984
Cost after iteration 83400: 10.256976
Cost after iteration 83500: 10.256968
Cost after iteration 83600: 10.256960
Cost after iteration 83700: 10.256952
Cost after iteration 83800: 10.256944
Cost after iteration 83900: 10.256936
Cost after iteration 84000: 10.256928
Cost after iteration 84100: 10.256920
Cost after iteration 84200: 10.256912
Cost after iteration 84300: 10.256904
Cost after iteration 84400: 10.256896
```

In [86]:
```
df = [passengerid['PassengerId'], pd.DataFrame(data=d["Y_prediction_test"].T,
result = pd.concat(df,axis=1)
result.to_csv('submisson_LR.csv', index=False)
```

4. Submit Kaggle score - 0.787

In [68]:
```
from IPython.display import Image
Image("score.png")
```

Out[68]:



In [87]:

```
Image("score_2.png")
```

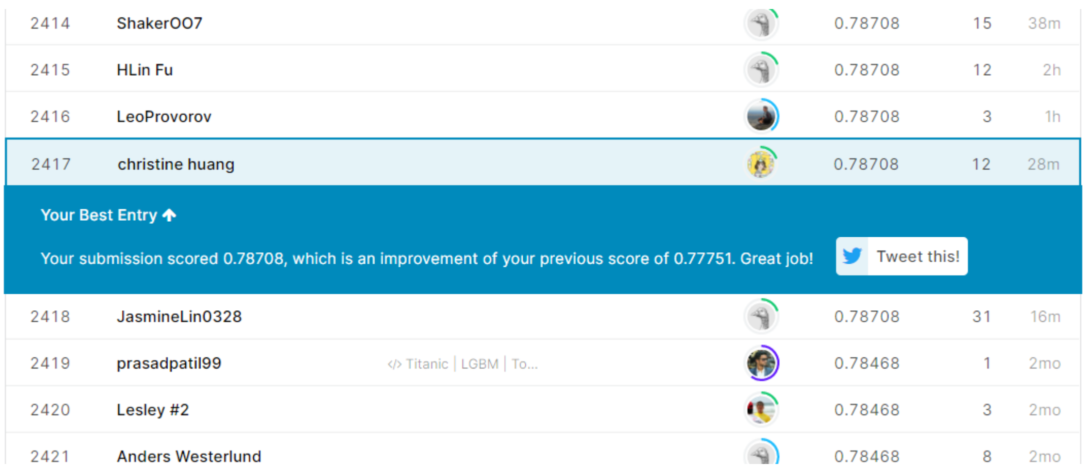
Out[87]:

1852	Sayantan Hazra			0.78947	3	35m
1853	harshit yadav			0.78708	8	2mo
1854	Darren Fong			0.78708	12	2mo
1855	Maria C Vesterli	</> RandomForestClas...		0.78708	7	2mo
1856	Vladislav Amiaga			0.78708	15	2mo
1857	Steve Readman			0.78708	3	2mo

In [88]:

```
Image("score_3.png")
```

Out[88]:



2414	ShakerOO7			0.78708	15	38m
2415	HLin Fu			0.78708	12	2h
2416	LeoProvorov			0.78708	3	1h
2417	christine huang			0.78708	12	28m

Your Best Entry ↑

Your submission scored 0.78708, which is an improvement of your previous score of 0.77751. Great job!

Tweet this!

2418	JasmineLin0328			0.78708	31	16m
2419	prasadpatil99	</> Titanic | LGBM | To...		0.78468	1	2mo
2420	Lesley #2			0.78468	3	2mo
2421	Anders Westerlund			0.78468	8	2mo