```
###########  Feature Engineering  ###########
#                                           #
#                                           #
############################################
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
from pandas import Series,DataFrame
import matplotlib.pyplot as plt
import re
from sklearn import preprocessing
from sklearn.ensemble import BaggingRegressor

data_train=pd.read_csv("/Users/jessicazhao/Documents/homework/Data
data_test=pd.read_csv("/Users/jessicazhao/Documents/homework/Data/
data_train
data_train.info()
fig=plt.figure()
fig.set(alpha=0.2)
data_train.survived.value_counts().plot(kind='bar')
plt.title(u"survived (1 is survived) ")
plt.ylabel(u"population")
plt.xlabel(u"Pclass")
plt.show()
plt.scatter(data_train.Survived,data_train.Age)
plt.show()
############ deal with the null value of cabin############
combine["Cabin_2"]=combine["Cabin_1"][combine["Cabin_1"]!=0]
combine["Cabin_2"][combine["PassengerId"].isin(combine["Passenger
combine["Cabin_2"][combine["PassengerId"].isin(combine["Passenger
combine["Cabin_2"][combine["PassengerId"].isin(combine["Passenger

##############
fig=plt.figure()
fig.set(alpha=0.2)
Survived_0=data_train.Survived[data_train.Cabin_1==0].Value_count
Survived_1=data_train.Survived[data_train.Cabin_1==1].value_count
Survived_2=data_train.Survived[data_train.Cabin_1==2].value_count
Survived_3=data_train.Survived[data_train.Cabin_1==3].value_count
Survived_4=data_train.Survived[data_train.Cabin_1==4].value_count
Survived_5=data_train.Survived[data_train.Cabin_1==5].value_count
Survived_6=data_train.Survived[data_train.Cabin_1==6].value_count
Survived_7=data_train.Survived[data_train.Cabin_1==7].value_count
df=pd.DataFrame({u'NA':Survived_0,u'A':Survived_1,u'b':Survived_2
df.plot(kind='bar')
plt.show()
```

```
###################
# merge train and test
combine=pd.concat([data_train,data_test])
# rebuild index
combine.reset_index(inplace=True)
# delete index column
combine.drop('index', axis=1, inplace=True)
print combine.shape[1], "columns:", combine.columns.values
print "Row count:", combine.shape[0]
###################“Fare”, “Embarked” using median value to fill 
combine['Fare'][np.isnan(combine['Fare'])]=combine['Fare'].median
#common value:
combine.Embarked[ df.Embarked.isnull() ] = df.Embarked.dropna().m
##############decrease the varience of value in particular varia
combine['Fare_bin']=pd.qcut(combine['Fare'],4)
combine = pd.concat([combine, pd.get_dummies(combine['Fare_bin'])
#####translate the non-numeric value into numeric value
combine['Embarked'][combine['Embarked']=='S']=1
combine['Embarked'][combine['Embarked']=='C']=2
combine['Embarked'][combine['Embarked']=='Q']=3

##########store pro1.csv##########
#                                        #
combine.to_csv('/Users/jessicazhao/Documents/homework/DataAnalyti
########################################

#########deal with Age->Child###############
child=combine['Age']
Child=pd.DataFrame(data=child)
Child['Age'][Child['Age']<18]=1
Child['Age'][Child['Age']>=18]=0
combine['Child']=Child

########## missing data :Age ##################
#####average +/- stantard deviation #####--good
average_age   = combine["Age"].mean()
std_age       = combine["Age"].std()
count_nan_age = combine["Age"].isnull().sum()
rand = np.random.randint(average_age - std_age, average_age + std_
combine['Age'][df.Age.isnull()] = rand
Child['Age'][Child['Age']<18]=1
Child['Age'][Child['Age']>=18]=0
#################################

##########store pro2.csv##########
#                                        #
```

```
combine.to_csv('/Users/jessicazhao/Documents/homework/DataAnalyti
########################################

combine["Age"][combine["PassengerId"].isin(combine["PassengerId"]
combine["Age"][combine["PassengerId"].isin(combine["PassengerId"]
combine["Age"][combine["PassengerId"].isin(combine["PassengerId"]
combine["Age"][combine["PassengerId"].isin(combine["PassengerId"]
combine["Age"][combine["PassengerId"].isin(combine["PassengerId"]
combine["Age"][combine["PassengerId"].isin(combine["PassengerId"]
combine["Age"][combine["PassengerId"].isin(combine["PassengerId"]
combine["Age"][combine["PassengerId"].isin(combine["PassengerId"]

combine.to_csv("/Users/jessicazhao/Documents/homework/DataAnalyti
######  2.random forests (pro4.csv) #####
from sklearn.ensemble import RandomForestRegressor

### using RandomForestClassifier predict the missing data
def set_missing_ages(df):


    age_df = df[['Age','Age_sc','Fare_sc', 'Parch', 'SibSp', 'Pcl

    # saperate passengers into two groups:passengers with age/wit
    known_age = age_df[age_df.Age.notnull()].as_matrix()
    unknown_age = age_df[age_df.Age.isnull()].as_matrix()

    # predicton
    y = known_age[:, 0]

    # X:a set of feature
    X = known_age[:, 1:]

    # fiting RandomForestRegressor
    rfr = RandomForestRegressor(random_state=0, n_estimators=2000
    rfr.fit(X, y)

    # prediction process
    predictedAges = rfr.predict(unknown_age[:, 1::])

    # fill the blank with the prediction
    df.loc[ (df.Age.isnull()), 'Age' ] = predictedAges

    return df, rfr

def set_Cabin_type(df):
    df.loc[ (df.Cabin.notnull()), 'Cabin' ] = "Yes"
    df.loc[ (df.Cabin.isnull()), 'Cabin' ] = "No"
    return df

combine, rfr = set_missing_ages(combine)
combine = set_Cabin_type(combine)

##########restore pro5.csv###########
#                                       #
combine.to_csv('/Users/jessicazhao/Documents/homework/DataAnalyti
#######################################

########### Name->Title############
combine['Title'] = combine['Name'].map(lambda x: re.compile(", (.
combine['Title'][combine.Title == 'Jonkheer'] = 'Master'
combine['Title'][combine.Title.isin(['Ms','Mlle'])] = 'Miss'
combine['Title'][combine.Title == 'Mme'] = 'Mrs'
combine['Title'][combine.Title.isin(['Capt', 'Don', 'Major', 'Col
combine['Title'][combine.Title.isin(['Dona', 'Lady', 'the Countes
###########  Family   ############
#withfamily->1/0
combine['Family']=combine['Parch']+combine['SibSp']
combine['Family'][combine['Family']>1]=1
combine['Family'][combine['Family']==0]=0
#picture1
Survived_1 = combine.Survived[combine.Family == 1].value_counts()
Survived_0 = combine.Survived[combine.Family == 0].value_counts()
df=pd.DataFrame({u'Family=0':Survived_0,u'Family=1':Survived_1})
df.plot(kind='bar')
plt.xlabel(u'Survived')
plt.ylabel(u'counts')
plt.title(u'Survived&Family')
plt.show()
#picture2
Family_0 = combine.Family[combine.Survived == 0].value_counts()
Family_1=combine.Family[combine.Survived==1].value_counts()
df_0=pd.DataFrame({u'Survived=1':Family_1,u'Survived=0':Family_0}
df_0.plot(kind='bar')
plt.xlabel(u'Family(1->withFamily)')
plt.ylabel(u'counts')
plt.title(u'Survived&Family')
plt.show()
#familysize:<3->small;>=3->big
#familyname
combine['Surname']=combine['Name'].map(lambda x: re.compile("(Mr|
combine['Surname']=pd.factorize(df['Surname'])[0]
#title
```

```
combine['Title_id']=pd.factorize(combine['Title'])[0]+1
#familysize
combine['FamilySize']=combine['Parch']+combine['SibSp']+1
combine['FamilySize'].loc[combine['FamilySize']<3]='small'
combine['FamilySize'].loc[combine['FamilySize']!='small']='big'
combine['FamilySize'][combine['FamilySize']=='small']=0
combine['FamilySize'][combine['FamilySize']=='big']=1
combine['FamilySize']=combine['FamilySize'].astype(int)
##########store pro3.csv##########
#                                        #
combine.to_csv('/Users/jessicazhao/Documents/homework/DataAnalyti
##########################################

##scalling Age&Fare
scaler=preprocessing.StandardScaler()
combine['Age_sc']=scaler.fit_transform(combine['Age'])
combine['Fare_sc']=scaler.fit_transform(combine['Fare'])
##########store pro4.csv(Age is still missing)pro4_1.csv(conplet
#                                        #
combine.to_csv('/Users/jessicazhao/Documents/homework/DataAnalyti
##########################################

###########  Logistic Regression  ###########
#                                            #
#                                            #
############################################
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
from pandas import Series,DataFrame
import matplotlib.pyplot as plt
import re
from sklearn.ensemble import BaggingRegressor
from sklearn import linear_model
from sklearn.ensemble import RandomForestClassifier
from sklearn import cross_validation

############start logistic regression

df=pd.read_csv("/Users/jessicazhao/Documents/homework/DataAnalyti
data_test=pd.read_csv("/Users/jessicazhao/Documents/homework/Data
#df=pd.read_csv("/Users/jessicazhao/Documents/homework/DataAnalyt
#df=df.filter(regex='Survived|Age_sc|SibSp|Parch|Fare_[0, 7.896]|
df_test=pd.read_csv("/Users/jessicazhao/Documents/homework/DataAn
train_df = df.filter(regex='Survived|Age_sc|Fare_sc|Sex|Surname|P
train_np = train_df.as_matrix()
```

```
train_np

# the result of Survival
y = train_np[:, 0]

# X is the set of feature
X = train_np[:, 1:]

# fit those feature with BaggingRegressor
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e
clf
bagging_clf = BaggingRegressor(clf, n_estimators=200, max_samples
bagging_clf.fit(X, y)

test = df_test.filter(regex='Age_sc|Fare_sc|Sex|Surname|Pclass|Fa
test.info()
predictions = bagging_clf.predict(test)

predictions.astype(np.int32)
result = pd.DataFrame({'PassengerId':data_test['PassengerId'].as_
result.to_csv("/Users/jessicazhao/Documents/homework/DataAnalytic
#######cross-validation(do it over and over)

#score model by using cross-validation
print cross_validation.cross_val_score(clf,X,y,cv=5)

#slice 70% training data from the whole as cross-validation train
split_train, split_cv = cross_validation.train_test_split(df, tes
train_cv = split_train.filter(regex='Survived|Age_sc|Fare_sc|Sex|

#model logistic regression with splited training data
clf = linear_model.LogisticRegression(C=1.0, penalty='l1', tol=1e
clf.fit(train_cv.as_matrix()[:,1:], train_cv.as_matrix()[:,0])

#test model with splited training data(split_cv)
test_cv = split_cv.filter(regex='Survived|Age_sc|Fare_sc|Sex|Surn
predictions = clf.predict(test_cv.as_matrix()[:,1:])

#find out bad case for further analysis and optimization
origin_data_train = pd.read_csv("/Users/jessicazhao/Documents/hom
bad_cases = origin_data_train.loc[origin_data_train['PassengerId'
bad_cases

##############  Random Forest  ##############
#                                           #
#                                           #
```

```python
############################################
# -*- coding: utf-8 -*-
import pandas as pd
import numpy as np
from pandas import Series,DataFrame
import matplotlib.pyplot as plt
import re

from sklearn.ensemble import BaggingRegressor
from sklearn import linear_model
from sklearn.ensemble import RandomForestClassifier

data_test=pd.read_csv("/Users/jessicazhao/Documents/homework/Data/
df=pd.read_csv("/Users/jessicazhao/Documents/homework/DataAnalytic
df_test=pd.read_csv("/Users/jessicazhao/Documents/homework/DataAna
df=df.filter(regex='Survived|Age_sc|Fare_sc|Sex|Surname|Pclass|Far

####

from sklearn.ensemble import RandomForestClassifier
X = df[:df.shape[0]].values[:, 1::]
y = df[:df.shape[0]].values[:, 0]

X_test = df_test[:df_test.shape[0]].values[:, 1::]

random_forest = RandomForestClassifier(max_depth=5,oob_score=True
random_forest.fit(X, y)

Y_pred = random_forest.predict(X_test)

print random_forest.score(X, y)
submission = pd.DataFrame({
            "PassengerId": data_test["PassengerId"],
            "Survived": Y_pred.astype(int)
        })
submission.to_csv("/Users/jessicazhao/Documents/homework/DataAnaly

#########Feature importances with RandomForest

#from sklearn import important_features
data_test=pd.read_csv("/Users/jessicazhao/Documents/homework/Data/
df=pd.read_csv("/Users/jessicazhao/Documents/homework/DataAnalytic
df=df.filter(regex='Survived|Age_sc|SibSp|Parch|Fare_sc|Sex|Pclass
df_test=pd.read_csv("/Users/jessicazhao/Documents/homework/DataAna
X=df[:df.shape[0]].values[:,1::]
y=df[:df.shape[0]].values[:,1]
```

```
features_list = df.columns.values[1::]

# Fit a random forest with (mostly) default parameters to determi
forest = RandomForestClassifier(oob_score=True, n_estimators=1000
forest.fit(X, y)
feature_importance = forest.feature_importances_

# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_import

# Get the indexes of all features over the importance threshold
important_idx = np.where(feature_importance)[0]

# Get the sorted indexes of important features
sorted_idx = np.argsort(feature_importance[important_idx])[::-1]
print "\nFeatures sorted by importance (DESC):\n", important_idx[

# Adapted from http://scikit-learn.org/stable/auto_examples/ensem
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[important_idx][sorted_idx[::-1]]
plt.yticks(pos, important_idx[sorted_idx[:-1]])
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()

sorted_idx
feature_importance
forest.get_params()
df.filter(regex='Survived|Age_sc|SibSp|Parch|Fare_[0, 7.896]|Fare
```