In [524]:

```
import pandas as pd
import numpy as np
import re
```

Reading data from csv

In [525]:

```
data_Singapore = pd.read_csv("data/SingaporeTrain.csv")
data_NY = pd.read_csv("data/NYTrain.csv")
data_London = pd.read_csv("data/LondonTrain.csv")
data_Singapore.head()
# data_NY.head()
# data_London.head()
```

Out[525]:

	row ID	educationInfoForAgeGroupEstimation	workInfoForAgeGrou
0	d0cc2d6caf6eb55ccf606770f24df2b5	NaN	
1	eb4e2aa0bf687227821b17cfcd5365e3	Bukit Batok Secondary School ITE College West ...	
2	acfb8e93139fe820996b27ee3c6bc9db	St Patrick's School, Singapore Class of 2...	
3	ea251951bd3c7667daf214e8efa92f4e	NaN	
4	f8f7b497ad8167535345bfc3d59d6aaa	NaN	PT. Duta Marg September

Combining all the data to a single dataframe

In [526]:

```
data_Singapore['source'] = 'Singapore'
data_NY['source'] = 'NY'
data_London['source'] = 'London'
data = pd.concat([data_Singapore, data_NY, data_London],ignore_index=True)
data.shape
```

Out[526]:

```
(10241, 11)
```

In [527]:

```
data['source'].unique()
```

Out[527]:

```
array(['Singapore', 'NY', 'London'], dtype=object)
```

In [528]:

```
data['gender'].unique()
```

Out[528]:

```
array(['female', 'male', nan], dtype=object)
```

In [529]:

```
data['realAge'].unique()
```

Out[529]:

```
array([nan, 27., 19., 20., 22., 21., 34., 31., 23., 18., 36., 24., 2
6.,
       29., 30., 25., 56., 46., 33., 28., 82., 35., 37., 39.])
```

In [530]:

```
data['ageGroup'].unique()
```

Out[530]:

```
array([nan, 'AGE10_20', 'AGE20_30', 'AGE30_40', 'AGE40_50', 'AGE50_IN
F'],
      dtype=object)
```

In [531]:

```
data['relationship'].unique()
```

Out[531]:

```
array([nan, 'single', 'in a relationship', 'married'], dtype=object)
```

In [532]:

```
data['educationLevel'].unique()
```

Out[532]:

```
array([nan, 'college', 'school', 'undergraduate', 'graduate'],
      dtype=object)
```

In [533]:

```
data['occupation'].unique()
```

Out[533]:

```
array([nan, 'archetecture and engineering',
       'food preparation and service related',
       'arts, design, entertainment, sports, and media', 'management',
       'office and administrative support', 'personal care and servic
e',
       'protective service', 'healthcare support', 'sales and relate
d',
       'legal', 'transportation and material moving',
       'computer and mathematical', 'production',
       'life, physical, and social science',
       'education, training, and library',
       'healthcare practitioners and technical',
       'building and grounds cleaning and maintenance',
       'farming, fishing and forestry', 'construction and extraction',
       'business and financial operations',
       'community and social service'], dtype=object)
```

In [534]:

```
data['income'].unique()
```

Out[534]:

```
array([nan, '$$', '$', '$$$', '$$$$'], dtype=object)
```

In [535]:

```
data['income'] = data['income'].fillna("no").apply(lambda x: -1 if x == "no" else l
```

In [536]:

```
data['income'].unique()
```

Out[536]:

```
array([-1,  2,  1,  3,  4])
```

Exctracting features from "workInfoForAgeGroupEstimation"

In [537]:

```
row_id = 4
row = data['workInfoForAgeGroupEstimation'][row_id]
# row = 'PT. Duta Marga Lestarindo September 19 present Land Transport Authority Pr
row
```

Out[537]:

```
'PT. Duta Marga Lestarindo September 2013 to present Land Transport Au
thority Project Engineer  August 2011 to July 2013 T.Y. Lin Internati
onal Civil Engineer  January 2010 to June 2010'
```

In [538]:

```
def process_work_info(row):
    years_raw = re.compile(r'\b(?:19|20)\d{2}|\bpresent').findall(row)
    years =  [int(year) if year != 'present' else 2018 for year in years_raw]

    start_year, number_of_places, total_length, mean_length, min_length, max_length
    if row != None and len(years)<2:
        number_of_places = 1
    else:
        pairs = [years[i:i+2] for i in range(0, len(years)-1, 2)]
        lengths = [pair[1]-pair[0] for pair in pairs]
        number_of_places = len(pairs)
        max_length = max(lengths)
        min_length = min(lengths)
        total_length = sum(lengths)
        mean_length = total_length / float(len(lengths))
        start_year = min(years)
        working_now = None
        if len(years_raw)>0:
            working_now = float('present' in years_raw)

    return start_year, number_of_places, total_length, mean_length, min_length, max
```

In [539]:

```
process_work_info(row)
```

Out[539]:

```
(2010, 3, 7, 2.3333333333333335, 0, 5, 1.0)
```

In [540]:

```
data['workInfoForAgeGroupEstimation'] = data['workInfoForAgeGroupEstimation'].map(s
data = data.join(pd.DataFrame(data['workInfoForAgeGroupEstimation'].tolist(), index
data.drop(columns=['workInfoForAgeGroupEstimation'], inplace=True)
```

Exctracting features from "educationInfoForAgeGroupEstimation"

In [541]:

```
# possible_levels = ['college', 'school', 'undergraduate', 'graduate']

# possible_values_for_levels = {}
# "Secondary School", "Ngee", "Institute", "University", "High School"
# # CHIJ Secondary (Toa Payoh) - автономная католическая школа для девочек в Сингап
# Class of 2010
```

In [542]:

```
row_id = 45
row = data['educationInfoForAgeGroupEstimation'][row_id]
# row = "Republic Polytechnic 2012 to 2015 • Singapore CHIJ Kellock Singapore CHIJ
row
```

Out[542]:

```
'SMK Seri Tanjong Class of 2009 • Melaka sek. men. seri tanjung,melaka
(PMR) Class of 2005 • Malacca City, Malaysia sek. men. seri tanjung,me
laka (SPM) Class of 2007 Maktab Koperasi Malaysia Class of 2012 • Kuch
ing, Malaysia Uni of Oxford Oxford, Oxfordshire skm seri tanjung Class
of 2009 • Malacca City, Malaysia'
```

In [543]:

```
def process_education_info(row):
    class_of_years = [int(i) for i in re.compile(r'(?<=of )\d{4}').findall(row)]
    range_years_starts = [int(i) for i in re.compile(r'\b(?:19|20)\d{2}(?= to )').f
    range_years_ends = [int(i) if i != 'present' else 2018 for i in re.compile(r'(?

    first_start_year = None
    last_start_year = None
    if len(class_of_years)>0 or len(range_years_starts)>0:
        first_start_year = min(class_of_years+range_years_starts)
        last_start_year = max(class_of_years+range_years_starts)


    finish_year = None
    if len(range_years_ends)>0:
        finish_year = max(range_years_ends)
    # else
    #     if len(class_of_years)>0 | len(range_years_starts)>0:
    #     finish_year = max(class_of_years) + 4 #usually one program = 4year

    # study_now = None

    number_of_programs = None
    num = len(class_of_years)+len(range_years_starts)
    if num<1 and len(row)>0:
        number_of_programs = 1
    else:
        number_of_programs = num

    return first_start_year, last_start_year, finish_year, number_of_programs
```

In [544]:

```
process_education_info(row)
```

Out[544]:

```
(2005, 2012, None, 5)
```

In [545]:

```
data['educationInfoForAgeGroupEstimation'] = data['educationInfoForAgeGroupEstimati
data = data.join(pd.DataFrame(data['educationInfoForAgeGroupEstimation'].tolist(),
data.drop(columns=['educationInfoForAgeGroupEstimation'], inplace=True)
```

Some preprocessing

In [546]:

```
columns_to_hot = ['gender', 'relationship', 'educationLevel', 'occupation', 'source
data = pd.get_dummies(data, columns=columns_to_hot)
data_with_labels = data[data['ageGroup'].notnull()].drop(columns=['row ID']).fillna

from sklearn.preprocessing import LabelEncoder
LE = LabelEncoder()
data_with_labels['ageGroup'] = LE.fit_transform(data_with_labels['ageGroup'])
```

In [547]:

```
data_with_labels.head()
```

Out[547]:

	realAge	ageGroup	income	w_start_year	w_number_of_places	w_total_length	w_mean_le
2	-1.0	0	-1	-1.0	1	-1.0	
5	27.0	1	-1	-1.0	1	-1.0	
11	-1.0	0	-1	-1.0	1	-1.0	
16	19.0	1	-1	-1.0	1	-1.0	
17	-1.0	1	-1	-1.0	1	-1.0	

5 rows × 47 columns

MODELS

Train-test split

In [548]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(data_with_labels.drop(columns=[
```

Classifiers

In [549]:

```
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import f1_score, accuracy_score

rf_cls = RandomForestClassifier(n_estimators=100)
xgb_cls = XGBClassifier()
```

Learning and scores

RandomForestClassifier

In [550]:

```
%%time
rf_cls.fit(X_train, y_train)
y_pred_on_train = rf_cls.predict(X_train)
y_pred_on_test = rf_cls.predict(X_test)

print("F1 score on train:", f1_score(y_train, y_pred_on_train, average='weighted'))
print("F1 score on test:", f1_score(y_test, y_pred_on_test, average='weighted'))
print("Accuracy on train:", accuracy_score(y_train, y_pred_on_train))
print("Accuracy on test:", accuracy_score(y_test, y_pred_on_test))
```

```
F1 score on train: 0.9838628968137404
F1 score on test: 0.7729322199851372
Accuracy on train: 0.9838492597577388
Accuracy on test: 0.7836990595611285
CPU times: user 255 ms, sys: 0 ns, total: 255 ms
Wall time: 256 ms

/home/saitkulov/anaconda3/lib/python3.6/site-packages/sklearn/metrics/
classification.py:1135: UndefinedMetricWarning: F-score is ill-defined
and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

XGBClassifier

In [551]:

```
%%time
xgb_cls.fit(X_train, y_train)
y_pred_on_train = xgb_cls.predict(X_train)
y_pred_on_test = xgb_cls.predict(X_test)

print("F1 score on train:", f1_score(y_train, y_pred_on_train, average='weighted'))
print("F1 score on test:", f1_score(y_test, y_pred_on_test, average='weighted'))
print("Accuracy on train:", accuracy_score(y_train, y_pred_on_train))
print("Accuracy on test:", accuracy_score(y_test, y_pred_on_test))
```

```
F1 score on train: 0.905917985924844
F1 score on test: 0.786013506631259
Accuracy on train: 0.9057873485868102
Accuracy on test: 0.7931034482758621
CPU times: user 612 ms, sys: 2.75 ms, total: 615 ms
Wall time: 639 ms

/home/saitkulov/anaconda3/lib/python3.6/site-packages/sklearn/preproce
ssing/label.py:151: DeprecationWarning: The truth value of an empty ar
ray is ambiguous. Returning False, but in future this will result in a
n error. Use `array.size > 0` to check that an array is not empty.
  if diff:
/home/saitkulov/anaconda3/lib/python3.6/site-packages/sklearn/preproce
ssing/label.py:151: DeprecationWarning: The truth value of an empty ar
ray is ambiguous. Returning False, but in future this will result in a
n error. Use `array.size > 0` to check that an array is not empty.
  if diff:
/home/saitkulov/anaconda3/lib/python3.6/site-packages/sklearn/metrics/
classification.py:1135: UndefinedMetricWarning: F-score is ill-defined
and being set to 0.0 in labels with no predicted samples.
  'precision', 'predicted', average, warn_for)
```

Confusion Matrix

In [552]:

```
y_pred = rf_cls.predict(X_test)
reversefactor = dict(zip(range(5),LE.classes_))
y_test = np.vectorize(reversefactor.get)(y_test)
y_pred = np.vectorize(reversefactor.get)(y_pred)
# Making the Confusion Matrix
print(pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted']))
```

Predicted	AGE10_20	AGE20_30	AGE30_40
Actual			
AGE10_20	86	31	0
AGE20_30	24	148	1
AGE30_40	0	5	16
AGE40_50	0	1	5
AGE50_INF	0	1	1

In [553]:

```
y_pred = xgb_cls.predict(X_test)
reversefactor = dict(zip(range(5), LE.classes_))
# y_test = np.vectorize(reversefactor.get)(y_test)
y_pred = np.vectorize(reversefactor.get)(y_pred)
# Making the Confusion Matrix
print(pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted']))
```

```
Predicted  AGE10_20  AGE20_30  AGE30_40  AGE50_INF
Actual
AGE10_20         94        23         0          0
AGE20_30         29       141         3          0
AGE30_40          0         4        17          0
AGE40_50          0         2         4          0
AGE50_INF         0         0         1          1
```

```
/home/saitkulov/anaconda3/lib/python3.6/site-packages/sklearn/preproce
ssing/label.py:151: DeprecationWarning: The truth value of an empty ar
ray is ambiguous. Returning False, but in future this will result in a
n error. Use `array.size > 0` to check that an array is not empty.
  if diff:
```

Features importances

In [554]:

```
sorted_importance = sorted(list(zip(X_train.columns, rf_cls.feature_importances_)),
sorted_importance[:20]
```

Out[554]:

```
[('e_last_start_year', 0.18960774756123866),
 ('e_first_start_year', 0.14330488085354062),
 ('realAge', 0.10129244730638326),
 ('e_number_of_programs', 0.05252184144417283),
 ('educationLevel_school', 0.05181927046449902),
 ('educationLevel_undergraduate', 0.04671285470437419),
 ('educationLevel_college', 0.03904064444118559),
 ('w_total_length', 0.035550013873838514),
 ('w_start_year', 0.035123524776755934),
 ('w_mean_length', 0.03015800334808914),
 ('e_finish_year', 0.029320987967452767),
 ('w_max_length', 0.027529850878679714),
 ('w_min_length', 0.027252999580911763),
 ('relationship_single', 0.02237254691843906),
 ('gender_male', 0.01880580257581921),
 ('w_number_of_places', 0.017554512641087726),
 ('gender_female', 0.01692018996426364),
 ('income', 0.012765081846118713),
 ('relationship_married', 0.011564627156773673),
 ('occupation_management', 0.010985810266447267)]
```

In [555]:

```
sorted_importance = sorted(list(zip(X_train.columns, xgb_cls.feature_importances_))
sorted_importance[:20]
```

Out[555]:

```
[('e_last_start_year', 0.25258893),
 ('e_first_start_year', 0.15848717),
 ('realAge', 0.14047727),
 ('educationLevel_undergraduate', 0.05357947),
 ('w_mean_length', 0.052678972),
 ('w_total_length', 0.041422784),
 ('educationLevel_school', 0.040972535),
 ('w_start_year', 0.036920305),
 ('e_number_of_programs', 0.035119317),
 ('e_finish_year', 0.027014859),
 ('educationLevel_college', 0.022062134),
 ('w_min_length', 0.01710941),
 ('w_number_of_places', 0.016208915),
 ('w_max_length', 0.013057182),
 ('gender_female', 0.012606934),
 ('income', 0.012156686),
 ('occupation_education, training, and library', 0.012156686),
 ('occupation_legal', 0.010355696),
 ('relationship_single', 0.0081044575),
 ('occupation_management', 0.0081044575)]
```

Preprocess test csv data

In [556]:

```
def process_test_csv(path):
    test_data_Singapore = pd.read_csv(path+"SingaporeTest.csv")
    test_data_NY = pd.read_csv(path+"NYTest.csv")
    test_data_London = pd.read_csv(path+"LondonTest.csv")

    test_data_Singapore['source'] = 'Singapore'
    test_data_NY['source'] = 'NY'
    test_data_London['source'] = 'London'
    test_data = pd.concat([test_data_Singapore, test_data_NY, test_data_London],ign

    test_data['income'] = test_data['income'].fillna("no").apply(lambda x: -1 if x

    test_data['workInfoForAgeGroupEstimation'] = test_data['workInfoForAgeGroupEsti
    test_data = test_data.join(pd.DataFrame(test_data['workInfoForAgeGroupEstimatio
    test_data.drop(columns=['workInfoForAgeGroupEstimation'], inplace=True)

    test_data['educationInfoForAgeGroupEstimation'] = test_data['educationInfoForAg
    test_data = test_data.join(pd.DataFrame(test_data['educationInfoForAgeGroupEsti
    test_data.drop(columns=['educationInfoForAgeGroupEstimation'], inplace=True)

    columns_to_hot = ['gender', 'relationship', 'educationLevel', 'occupation', 'so
    test_data = pd.get_dummies(test_data, columns=columns_to_hot)
    test_data_with_labels = test_data[test_data['ageGroup'].notnull()].drop(columns
    for i in set(data_with_labels.columns)-set(test_data_with_labels.columns):
        test_data_with_labels.insert(len(test_data_with_labels.columns), i, 0)

    test_data_with_labels['ageGroup'] = LE.transform(test_data_with_labels['ageGrou
    return test_data_with_labels
```

Reading and processing new csv

In [557]:

```
path = "path to the folder with test data "
test_data_with_labels = process_test_csv(path)
test_data_with_labels = test_data_with_labels[data_with_labels.columns]
X, y = test_data_with_labels.drop(columns=['ageGroup']), test_data_with_labels.ageG
```

Predicting and scores

RandomForestClassifier

In [562]:

```
%%time
y_pred_on_test_file = rf_cls.predict(X)

print("F1 score on test.csv:", f1_score(y, y_pred_on_test_file, average='weighted')
print("Accuracy on test.csv:", accuracy_score(y, y_pred_on_test_file))
```

In [563]:

```
reversefactor = dict(zip(range(5),LE.classes_))
y_test = np.vectorize(reversefactor.get)(y)
y_pred_on_test_file = np.vectorize(reversefactor.get)(y_pred_on_test_file)
# Making the Confusion Matrix
print(pd.crosstab(y_test, y_pred_on_test_file, rownames=['Actual'], colnames=['Pred
```

XGBClassifier

In [566]:

```
%%time
y_pred_on_test_file = xgb_cls.predict(X)

print("F1 score on test.csv:", f1_score(y, y_pred_on_test_file, average='weighted')
print("Accuracy on test.csv:", accuracy_score(y, y_pred_on_test_file))
```

In [567]:

```
reversefactor = dict(zip(range(5),LE.classes_))
y_test = np.vectorize(reversefactor.get)(y)
y_pred_on_test_file = np.vectorize(reversefactor.get)(y_pred_on_test_file)
# Making the Confusion Matrix
print(pd.crosstab(y_test, y_pred_on_test_file, rownames=['Actual'], colnames=['Pred
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```