# The Interview Attendance Problem in Kaggle

## Predict which candidates will attend the intervew

https://www.kaggle.com/vishnusraghavan/the-interview-attendance-problem/ (https://www.kaggle.com/vishnusraghavan/the-interview-attendance-problem/)

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib as mpl
from   datetime import datetime
import os
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import make_scorer
from sklearn.metrics import accuracy_score
from sklearn.metrics import f1_score
from sklearn.pipeline import Pipeline
from sklearn.externals import joblib

%matplotlib inline
```

### Main Goals

a. Create a model predicting if a candidate will attend an interview. This will be indicated by the "Observed Attendance" column in the data set. Create the model only using the records where this column is not null

b. Provide a probability and a prediction for the candidates where the "Observed Attendance" column is null.

In [2]:
```
class OneHotEncodeData(BaseEstimator, TransformerMixin):
    def __init__(self):
        '''
        This class is to one-hot encode the categorical features.
        '''
        self.one_hot_feature_names = ['Client name',
                        'Industry',
                        'Location',
                        'Position to be closed',
                        'Nature of Skillset',
                        'Interview Type',
                        #'Name(Cand ID)',
                        'Gender',
                        'Candidate Current Location',
                        'Candidate Job Location',
                        'Interview Venue',
                        'Candidate Native location',
                        'Have you obtained the necessary permission to star
                        'Hope there will be no unscheduled meetings',
                        'Can I Call you three hours before the interview an
                        'Can I have an alternative number/ desk number. I a
                        'Have you taken a printout of your updated resume.
                        'Are you clear with the venue details and the landm
                        'Has the call letter been shared',
                        'Marital Status']
        self.label_encoders   = None
        self.one_hot_encoders = None

    def fit(self, X, y=None):
        '''
        This method trains label encoders and one-hot encoders.
        '''

        X1 = X.copy()

        # one_hot_features = np.zeros((feature_values.shape[0], 0))

        label_encoders   = {}
        one_hot_encoders = {}
        for fname in self.one_hot_feature_names:
            label_encoder   = LabelEncoder()
            one_hot_encoder = OneHotEncoder(categories='auto')
            feature         = X1[fname]
            feature_label_encoded = label_encoder.fit_transform(feature)
            label_encoders[fname]   = label_encoder;
            one_hot_encoder.fit(feature_label_encoded.reshape(-1,1))
            one_hot_encoders[fname] = one_hot_encoder;

        # save label encoders and one-hot encoders for encoding test datase
        self.label_encoders   = label_encoders
        self.one_hot_encoders = one_hot_encoders

        return self

    def transform(self, X, y=None):
        '''
```

```
        This method uses trained label encoders and one-hot encoders
        to one-hot encodes the given catogrical fields.
        '''
        X1 = X.copy()

        # one-hot encode
        one_hot_features = np.zeros((X1.shape[0], 0))
        for fname in self.one_hot_feature_names:
            label_encoder   = self.label_encoders[fname]
            one_hot_encoder = self.one_hot_encoders[fname]
            feature  = X1[fname]
            fencoded = label_encoder.transform(feature)
            f1hot    = one_hot_encoder.transform(fencoded.reshape(-1,1)).to
            one_hot_features = np.c_[one_hot_features, f1hot]

        # drop the original features that have just been one-hot encoded
        X1 = pd.DataFrame(X1).drop(self.one_hot_feature_names, axis=1).valu

        # combine one-hot codes into the features array
        X1 = np.c_[X1, one_hot_features]

        return X1
```

In [3]:
```
class FeaturesUppercase(BaseEstimator, TransformerMixin):
    def __init__(self, feature_names, drop_feature_names):
        '''
        This class is to change feature values to uppercase.
        '''
        self.feature_names      = feature_names
        self.drop_feature_names = drop_feature_names

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        '''
        This method is to change feature values to uppercase.
        '''
        X_uppercase = X.copy()

        for fname in self.feature_names:
            values = X_uppercase[fname]
            values = values.fillna('NaN')
            values = map(lambda x: x.strip().upper(), values)
            X_uppercase[fname] = values

        # drop less important features
        X_uppercase = X_uppercase.drop(self.drop_feature_names, axis=1)

        return X_uppercase
```

In [4]:
```
class ParseInterviewDate(BaseEstimator, TransformerMixin):
    def __init__(self):
        '''
        This class is to splits the date of interview into day (2 digits),
        '''
    def __parseDate(self, string, delimit):
        try:
            if ('&' in string):
                subs = tuple(string.split('&'))
                string = subs[0]
        except:
            print ('TypeError: {}'.format(string))
            return None

        string = string.strip()

        try:
            d = datetime.strptime(string, '%d{0}%m{0}%Y'.format(delimit))
        except:
            try:
                d = datetime.strptime(string, '%d{0}%m{0}%y'.format(delimit
            except:
                try:
                     d = datetime.strptime(string, '%d{0}%b{0}%Y'.format(de
                except:
                    try:
                         d = datetime.strptime(string, '%d{0}%b{0}%y'.forma
                    except:
                        try:
                            d = datetime.strptime(string, '%b{0}%d{0}%Y'.fo
                        except:
                            try:
                                d = datetime.strptime(string, '%b{0}%d{0}%y
                            except:
                                d = None
        return d

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        '''
        This method splits the date of interview into day (2 digits), month
        '''

        X1 = X.copy()

        days = []
        months = []
        years = []
        ditems = X1['Date of Interview'].values
        for ditem in ditems:
            if (isinstance(ditem, str) and len(ditem) > 0):
                if ('.' in ditem):
                    d = self.__parseDate(ditem, '.')
                elif ('/' in ditem):
```

```
                d = self.__parseDate(ditem, '/')
            elif ('-' in ditem):
                d = self.__parseDate(ditem, '-')
            elif (' ' in ditem):
                d = self.__parseDate(ditem, ' ')
            else:
                d = None

            if (d is None):
                # print("{}, invalid format of interview date!".format(
                days.append(0) # 0 - NaN
                months.append(0)
                years.append(0)
            else:
                days.append(d.day)
                months.append(d.month)
                years.append(d.year)
        else:
            days.append(0)
            months.append(0)
            years.append(0)

    X1['Year'] = years
    X1['Month'] = months
    X1['Day'] = days

    return X1
```

In [5]:
```
class BucketSkillset(BaseEstimator, TransformerMixin):
    def __init__(self):
        '''
        This class is to re-bucket the skill sets and candidates location f
        to combine small catogaries into one catogary 'Others'.
        '''
        self.skillset = ['JAVA/J2EE/Struts/Hibernate', 'Fresher', 'Accounti
          'JAVA/SPRING/HIBERNATE/JSF', 'Java J2EE', 'SAS', 'Oracle Plsql',
          'Lending and Liabilities', 'Banking Operations', 'Java', 'Core Ja
          'Senior software engineer-Mednet', 'ALS Testing', 'SCCM', 'COTS D
          'Sr Automation Testing', 'Regulatory', 'Hadoop', 'testing', 'Java

        self.candidate_locations = ['Chennai', 'Hyderabad', 'Bangalore', 'G
                        'Pune', 'Coimbatore', 'Allahabad', 'Noida', 'Visa
                        'Trivandrum', 'Kolkata', 'Trichy', 'Vellore']


    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        '''
        This method is to re-bucket the skill sets and candidates native lo
        '''

        X1 = X.copy()

        fnames = ('Nature of Skillset', 'Candidate Native location')
        fset   = (self.skillset, self.candidate_locations)
        for i, fname in enumerate(fnames):
            fvalues = X1[fname]
            X2 = map(lambda x: x if x in fset[i] else 'Others', fvalues)
            X1[fname] = pd.Series(X2)

        return X1
```

In [6]:
```
class GridSearch(object):
    def __init__(self, cv=10):
        '''
        This class finds the best model via Grid Search.
        '''
        self.grid_param = [
            {'n_estimators': range(68,69), # range(60, 70) # best 68
             'max_depth'   : range(8,9)}   # range(5, 10)}  # best 8
        ]
        self.cv = cv
        self.scoring_function = make_scorer(f1_score, greater_is_better=Tru
        self.gridSearch = None

    def fit(self, X, y):
        rfc = RandomForestClassifier()
        self.gridSearch = GridSearchCV(rfc, self.grid_param, cv=self.cv, sc
        self.gridSearch.fit(X, y)
        return self.gridSearch.best_estimator_
```

In [7]:
```
class PredictInterview(object):
    def __init__(self):
        '''
        This class is to predict the probability of a candidate attending s
        '''
        self.dataset_file_name = 'Interview_Attendance_Data.csv'
        self.feature_names = ['Date of Interview',
                              'Client name',
                              'Industry',
                              'Location',
                              'Position to be closed',
                              'Nature of Skillset',
                              'Interview Type',
                              #'Name(Cand ID)',
                              'Gender',
                              'Candidate Current Location',
                              'Candidate Job Location',
                              'Interview Venue',
                              'Candidate Native location',
                              'Have you obtained the necessary permission to start
                              'Hope there will be no unscheduled meetings',
                              'Can I Call you three hours before the interview and
                              'Can I have an alternative number/ desk number. I as
                              'Have you taken a printout of your updated resume. H
                              'Are you clear with the venue details and the landma
                              'Has the call letter been shared', 'Marital Status']

        self.drop_feature_names = [
                               'Name(Cand ID)',
                               'Date of Interview',
                               'Unnamed: 22',
                               'Unnamed: 23',
                               'Unnamed: 24',
                               'Unnamed: 25',
                               'Unnamed: 26']

        self.dataset = None
        self.rfc     = None
        self.gridSearch = None
        self.X_train = None
        self.y_train = None
        self.X_test  = None
        self.y_test  = None
        self.y_pred  = None
        self.X_clean = None
        self.y_clean = None
        self.X_train_encoded = None
        self.X_test_encoded  = None
        self.y_train_encoded = None
        self.accuracy_score  = None
        self.f1_score        = None
        self.oneHotEncoder   = None
        self.X_test_name_ids = None
        self.pipeline = None
```

```
    def loadData(self, path=None):
        '''
        This method loads a dataset file as a Pandas DataFrame, assuming th
        It also shuffles the loaded dataset as part of data preprocessing.
        '''
        if (path != None):
            path = os.path.join(path, self.dataset_file_name)
        else:
            path = self.dataset_file_name

        dataset = pd.read_csv(path)

        # shuffle data
        self.dataset = dataset.sample(frac=1).reset_index(drop=True)

        return self.dataset

    def PreprocessData(self):
        '''
        This method preprocesses the loaded dataset before applying one-hot
        '''

        y = self.dataset['Observed Attendance']                # extract la
        X = self.dataset.drop(['Observed Attendance'], axis=1) # extract fe

        self.oneHotEncoder = OneHotEncodeData()

        self.pipeline = Pipeline([
            ('bucket_skillset', BucketSkillset()),
            ('parse_interview_date', ParseInterviewDate()),
            ('features_to_uppercase', FeaturesUppercase(self.feature_names,
            ('one_hot_encoder', self.oneHotEncoder)
        ])

        X_1hot = self.pipeline.fit_transform(X)

        # fill up missing labels and then change labels to uppercase
        y = y.fillna('NaN')
        y_uppercase = map(lambda x: x.strip().upper(), y.values)
        y_uppercase = pd.Series(y_uppercase)

        # separate labeled records from unlabeled records
        self.X_train_encoded = X_1hot[y_uppercase != 'NAN']
        self.X_test_encoded  = X_1hot[y_uppercase == 'NAN']

        # save Names/ID for reporting later one
        self.X_test_name_ids = self.dataset['Name(Cand ID)'][y_uppercase ==

        y_train = y_uppercase[y_uppercase != 'NAN']
        # encode labels as follows: 0 - NO, 1 - YES, NAN - NAN
        y = map(lambda x: 1 if x == 'YES' else 0, y_train)
        y = pd.Series(y)

        self.y_train_encoded = y.values

        self.X_clean = X_1hot
        self.y_clean = y_uppercase
```

```
        return None

    def __splitData(self):
        '''
        This method triggers data preprocsssing and split dataset into trai
        '''
        self.X_train, self.X_test, self.y_train, self.y_test = train_test_s


        return (self.X_train, self.X_test, self.y_train, self.y_test)

    def trainModel(self):
        '''
        This method triggers splitting dataset and then find a best RandomF
        using the training features and labels.
        '''
        X_train, X_test, y_train, y_test = self.__splitData()
        self.gridSearch = GridSearch()
        self.rfc = self.gridSearch.fit(X_train, y_train)
        return self.rfc

    def predictClasses(self):
        '''
        This method predicts classes (YES or NO) using a trained model.
        '''
        if (self.rfc is None):
            print("No trained model available, please train a model first!"
            return None

        self.y_pred = self.rfc.predict(self.X_test)
        return self.y_pred

    def getModelMetrics(self):
        '''
        This method obtains the class prediction scores: (Accuracy Score, R
        '''
        if (self.y_test is None or self.y_pred is None):
            print('Failed to get model performance metrics because y_test i
            return None

        self.accuracy_score = accuracy_score(self.y_test, self.y_pred)
        self.f1_score = f1_score(self.y_test, self.y_pred)

        return (self.accuracy_score, self.f1_score)

    def predictNullAttendanceProbability(self):
        '''
        This method uses a trained model to predict the attendance probabil
        the candidates where the "Observed Attendance" column is null.
        '''
        y_pred = self.rfc.predict_proba(self.X_test_encoded)
        return y_pred

    def predictNullAttendanceClasses(self):
        '''
        This method predicts classes (YES or NO) using a trained model for
```

```
        '''
        y_pred = self.rfc.predict(self.X_test_encoded)
        return y_pred

    def predictAttendanceProbability(self, X):
        '''
        Given one preprocessed (including one-hot encoding) data smaple X,
        this method returns the probability of attendance probability.
        '''
        y_pred = self.rfc.predict_proba(X)
        return y_pred

    def predictAttendanceClass(self, X):
        '''
        Given one preprocessed (including one-hot encoding) data smaple X,
        this method returns the attendance Yes/No.
        '''
        y_pred = self.rfc.predict(X)
        return y_pred
```

## Task 1 (a)

a. Create a model predicting if a candidate will attend an interview. This will be indicated by the "Observed Attendance" column in the data set. Create the model only using the records where this column is not null

In [8]:
```
predictInterview = PredictInterview()
predictInterview.loadData()
predictInterview.PreprocessData()
predictInterview.trainModel()
predictInterview.predictClasses()
accuracy_score, f1_score = predictInterview.getModelMetrics()

print('accuracy score = {0}, F1 score = {1}'.format(accuracy_score, f1_scor
```

```
accuracy score = 0.691228070175, F1 score = 0.785365853659
```

## Task 1 (b)

b. Provide a probability and a prediction for the candidates where the "Observed Attendance" column is null.

In [9]:
```
pred_probs   = predictInterview.predictNullAttendanceProbability()
pred_classes = predictInterview.predictNullAttendanceClasses()

x = predictInterview.X_test_name_ids
z = zip(x, pred_probs, pred_classes)
answers = ('no', 'yes')

result = [[x1, p1[1], answers[c]] for x1, p1, c in z]
result_df = pd.DataFrame(np.array(result), columns=['Names/ID', 'Probabilit
result_df.to_csv('interview_prediction.csv')
result_df.head(100)
```

Out[9]:

| | Names/ID | Probability | Yes/No |
|---|---|---|---|
| 0 | Candidate 1000 | 0.904925527257 | yes |
| 1 | Candidate 1138 | 0.702611545358 | yes |
| 2 | Candidate 676 | 0.400012146243 | no |
| 3 | Candidate 631 | 0.782720408285 | yes |
| 4 | Candidate 859 | 0.690722966142 | yes |
| 5 | Candidate 60 | 0.708465668128 | yes |
| 6 | Candidate 523 | 0.473331142852 | no |
| 7 | Candidate 967 | 0.0846085635885 | no |
| 8 | Candidate 383 | 0.767593150781 | yes |
| 9 | Candidate 250 | 0.739531204147 | yes |
| 10 | Candidate 436 | 0.751207887564 | yes |
| 11 | Candidate 904 | 0.605965670103 | yes |
| 12 | Candidate 120 | 0.738894109853 | yes |
| 13 | Candidate 1108 | 0.693779182002 | yes |
| 14 | Candidate 871 | 0.68258967498 | yes |
| 15 | Candidate 10 | 0.797101029926 | yes |
| 16 | Candidate 706 | 0.701777570016 | yes |
| 17 | Candidate 1063 | 0.392217251406 | no |
| 18 | Candidate 220 | 0.684279291308 | yes |
| 19 | Candidate 1171 | 0.702611545358 | yes |
| 20 | Candidate 1078 | 0.383739819663 | no |
| 21 | Candidate 140 | 0.855557005482 | yes |
| 22 | Candidate 451 | 0.862428933337 | yes |
| 23 | Candidate 90 | 0.267887796679 | no |
| 24 | Candidate 200 | 0.667334950188 | yes |
| 25 | Candidate 493 | 0.623485191882 | yes |
| 26 | Candidate 598 | 0.737452699606 | yes |

| | Names/ID | Probability | Yes/No |
|---|---|---|---|
| **27** | Candidate 347 | 0.467567342065 | no |
| **28** | Candidate 937 | 0.0347228167081 | no |
| **29** | Candidate 1233 | 0.318414145679 | no |
| **...** | ... | ... | ... |
| **64** | Candidate 100 | 0.664405756458 | yes |
| **65** | Candidate 721 | 0.702611545358 | yes |
| **66** | Candidate 427 | 0.764428721514 | yes |
| **67** | Candidate 398 | 0.770192450403 | yes |
| **68** | Candidate 210 | 0.684279291308 | yes |
| **69** | Candidate 553 | 0.668081977743 | yes |
| **70** | Candidate 412 | 0.294365304337 | no |
| **71** | Candidate 1207 | 0.726280652032 | yes |
| **72** | Candidate 190 | 0.684279291308 | yes |
| **73** | Candidate 240 | 0.64574216413 | yes |
| **74** | Candidate 736 | 0.700133124357 | yes |
| **75** | Candidate 829 | 0.501182363297 | yes |
| **76** | Candidate 571 | 0.152795592167 | no |
| **77** | Candidate 982 | 0.264324601177 | no |
| **78** | Candidate 260 | 0.735200551189 | yes |
| **79** | Candidate 230 | 0.684279291308 | yes |
| **80** | Candidate 766 | 0.703603437716 | yes |
| **81** | Candidate 889 | 0.812153335812 | yes |
| **82** | Candidate 321 | 0.735858888348 | yes |
| **83** | Candidate 952 | 0.836308366022 | yes |
| **84** | Candidate 844 | 0.742955620406 | yes |
| **85** | Candidate 814 | 0.715002781939 | yes |
| **86** | Candidate 20 | 0.838185030918 | yes |
| **87** | Candidate 781 | 0.71317141719 | yes |
| **88** | Candidate 466 | 0.778581799089 | yes |
| **89** | Candidate 356 | 0.497673700506 | no |
| **90** | Candidate 290 | 0.419758286006 | no |
| **91** | Candidate 478 | 0.500935301346 | yes |
| **92** | Candidate 110 | 0.664405756458 | yes |
| **93** | Candidate 40 | 0.68720821916 | yes |

94 rows × 3 columns

## Testing one data sample

This is to test how to use the trained data preprocessing pipeline (including One-Hot encoders) and the trained Random Forest model to predict the interview attendance of one individual data sample.

In [10]:
```
'''
get the shuffled whole dataset without pre-processing
'''
data = predictInterview.dataset
print("dataset type: ", type(data), "shape: ", data.shape)

'''
Take one data sample from the whole dataset to create a dataframe
'''
columns = data.columns
values  = data.values[1]
pairs   = zip(columns, values)

data1 = {}
new_columns = []
for c, v in pairs:
    if c in columns:
        if c in ['Observed Attendance']:
            print('Observed attendance = {}'.format(v))
            continue
        else:
            new_columns.append(c)
            data1[c] = [v]

sample_df = pd.DataFrame(data1, columns=new_columns)

'''
pre-process the one data sample
'''
X_1hot = predictInterview.pipeline.transform(sample_df)

'''
predict
'''
pred_prob =  predictInterview.predictAttendanceProbability(X_1hot)
pred_class = predictInterview.predictAttendanceClass(X_1hot)

classes = ('No', 'Yes')

print('attendance prob = {}, attendance class: {}'.format(pred_prob[0,1], c
```

```
('dataset type: ', <class 'pandas.core.frame.DataFrame'>, 'shape: ', (123
4, 27))
Observed attendance = Yes
attendance prob = 0.708465668128, attendance class: Yes
```