In [ ]:

```
import seaborn as sns
import pandas as pd
import pandas as pd
import numpy as np
import matplotlib.pylab as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge, LassoCV, BayesianRidge
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
!pip install dmba
from dmba import classificationSummary, gainsChart, liftChart
from sklearn.ensemble import RandomForestClassifier
```

```
Collecting dmba
  Downloading https://files.pythonhosted.org/packages/44/7e/22fc51d7f54ac4662c5edcf0133083499bbea91bd6a6beb0c5b13f565a20/dmba-0.0.13-py3-none-any.whl (https://files.pythonhosted.org/packages/44/7e/22fc51d7f54ac4662c5edcf0133083499bbea91bd6a6beb0c5b13f565a20/dmba-0.0.13-py3-none-any.whl)
Installing collected packages: dmba
Successfully installed dmba-0.0.13
no display found. Using non-interactive Agg backend
```

In [ ]:

```
from google.colab import files

uploaded = files.upload()

for fn in uploaded.keys():
  print('User uploaded file "{name}" with length {length} bytes'.format(
      name=fn, length=len(uploaded[fn])))
```

Choose Files No file chosen

Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

```
Saving speed-dating2_V3.csv to speed-dating2_V3.csv
User uploaded file "speed-dating2_V3.csv" with length 1887004 bytes
```

In [30]:

```
dating = pd.read_csv("speed-dating2_V3.csv")
```

In [31]:
```
dating.shape # the shape of the data
```

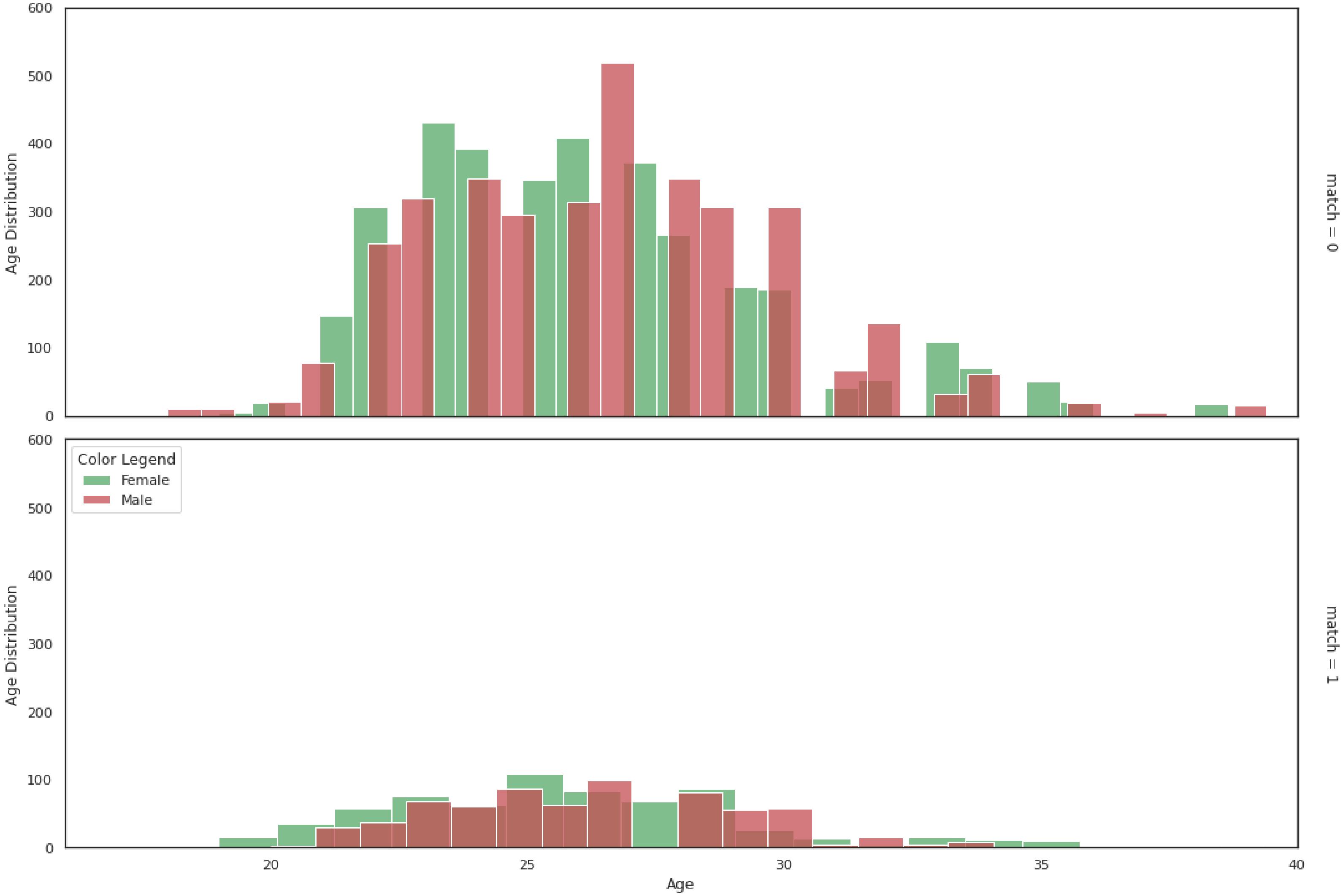
Out[31]:
```
(8378, 66)
```

In [32]:
```
dating.columns # the columns of the dataset
```

Out[32]:
```
Index(['gender', 'age', 'age_o', 'd_age', 'race', 'race_o', 'samerace',
       'importance_same_race', 'importance_same_religion', 'field',
       'pref_o_attractive', 'pref_o_sincere', 'pref_o_intelligence',
       'pref_o_funny', 'pref_o_ambitious', 'pref_o_shared_interests',
       'attractive_o', 'sincere_o', 'intelligence_o', 'funny_o', 'ambitious_o',
       'shared_interests_o', 'attractive_important', 'sincere_important',
       'intelligence_important', 'funny_important', 'ambition_important',
       'shared_interests_important', 'attractive', 'sincere', 'intelligence',
       'funny', 'ambition', 'attractive_partner', 'sincere_partner',
       'intelligence_partner', 'funny_partner', 'ambition_partner',
       'shared_interests_partner', 'sports', 'tvsports', 'exercise', 'dining',
       'museums', 'art', 'hiking', 'gaming', 'clubbing', 'reading', 'tv',
       'theater', 'movies', 'concerts', 'music', 'shopping', 'yoga',
       'interests_correlate', 'expected_happy_with_sd_people',
       'expected_num_interested_in_me', 'expected_num_matches', 'like',
       'guess_prob_liked', 'met', 'decision', 'decision_o', 'match'],
      dtype='object')
```

In [34]:

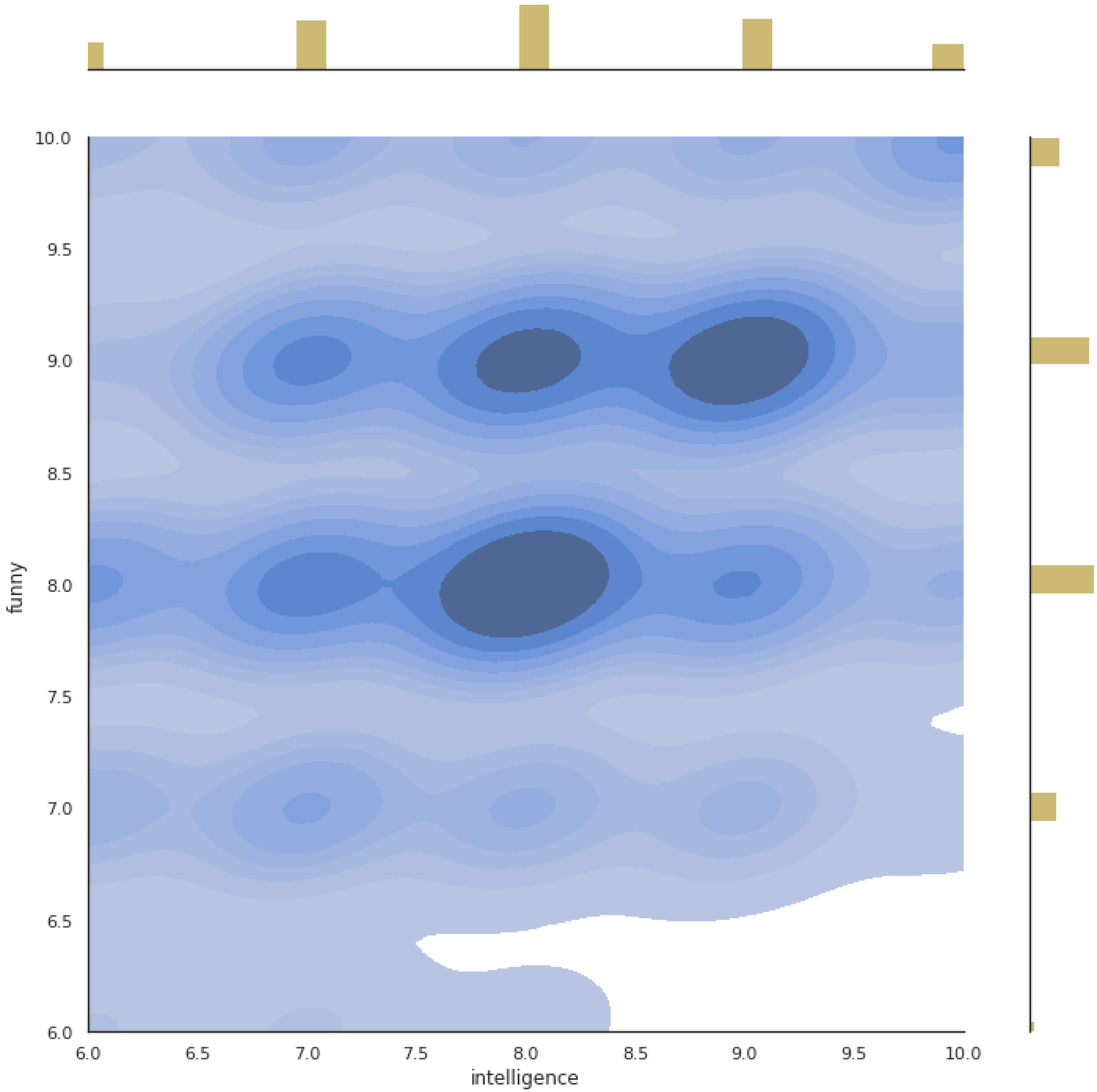
```
# histograms of the age distribution based on the gender and on the matching status

import matplotlib.pyplot as plt
g = sns.FacetGrid(dating, hue="gender",  row="match", height = 5, aspect = 3, palette = {'female':"g",'
g.map_dataframe(sns.histplot, x="age")

g = (g.set_axis_labels("Age","Age Distribution").
set(xlim=(16,40),ylim=(0,600)))
g.set_titles("Age")
plt.legend(title='Color Legend', loc='upper left', labels=['Female', 'Male'])

plt.show()
```



Age Distribution
match = 0
match = 1
Color Legend
Female
Male
Age
0
100
200
300
400
500
600
20
25
30
35
40

In [35]:

```
# visual that shows the relationship between funny and intelligence with the distribution for these vari
import matplotlib.pyplot as plt
%matplotlib inline
sns.set_theme(style="white")
g = sns.JointGrid(data=dating, x="intelligence", y="funny", height = 10, ratio = 7,xlim=(6, 10), ylim=(6
g.plot_joint(sns.kdeplot,
             fill=True)
g.plot_marginals(sns.histplot, color="y", alpha=1, bins=55 , weights = 5)
plt.show()
```


10.0
9.5
9.0
8.5
8.0
7.5
7.0
6.5
6.0
funny
6.0
6.5
7.0
7.5
8.0
8.5
9.0
9.5
10.0
intelligence

In [36]:

```
dating.field = dating.field.astype('category') # convert field to a categorical variable
```

In [37]:

```
dating.field.value_counts() # get the frequency of each profession
```

Out[37]:

```
business                          631
law                               604
mba                               468
social work                       414
international affairs             307
                                 ...
mfa  poetry                         6
business [finance & marketing]      6
soa -- writing                      6
theory                              5
marine geophysics                   5
Name: field, Length: 215, dtype: int64
```

In [39]:

```
import numpy as np
dating.frequent_professions = dating.field.replace(dating.field.value_counts().index[7:], np.nan) # get
```

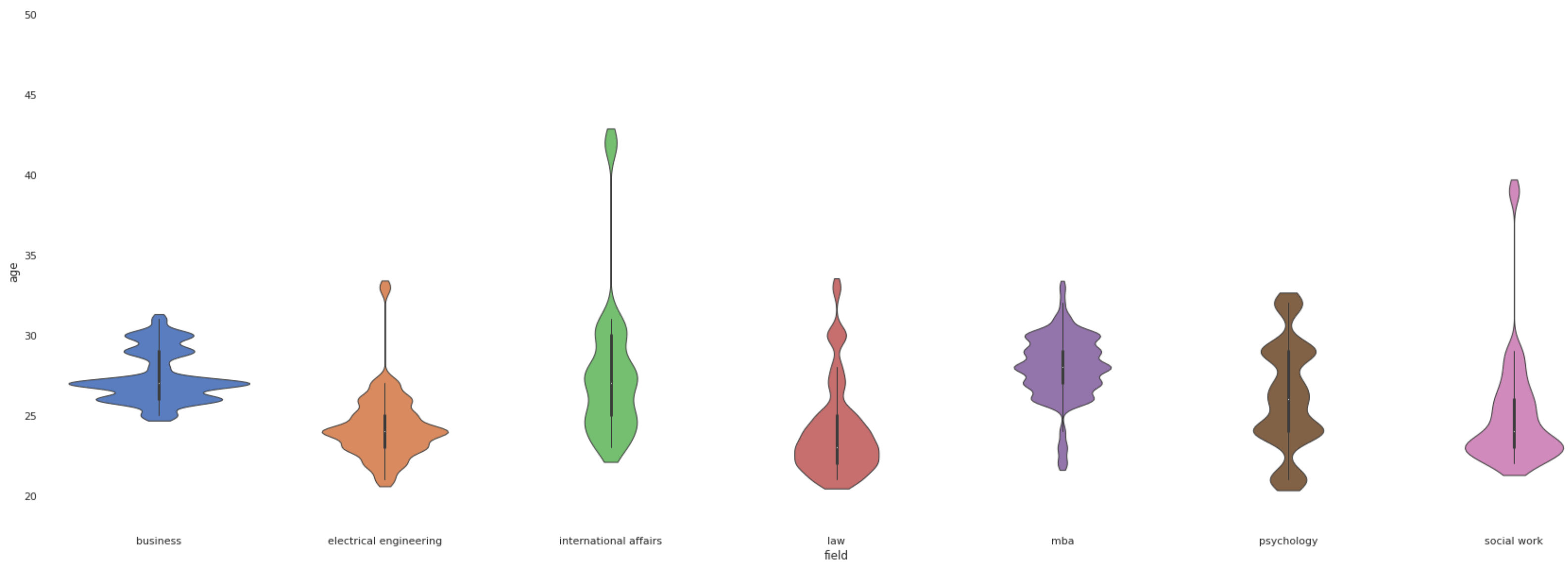
```
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2: UserWarning: Pandas doesn't allow columns t
o be created via a new attribute name - see https://pandas.pydata.org/pandas-docs/stable/indexing.html#attr
ibute-access (https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access)
```

In [40]:
```
dating.frequent_professions
```

Out[40]:
```
0       law
1       law
2       law
3       law
4       law
       ...
8373    NaN
8374    NaN
8375    NaN
8376    NaN
8377    NaN
Name: field, Length: 8378, dtype: category
Categories (7, object): ['business', 'electrical engineering', 'international affairs', 'law',
                         'mba', 'psychology', 'social work']
```

In [42]:

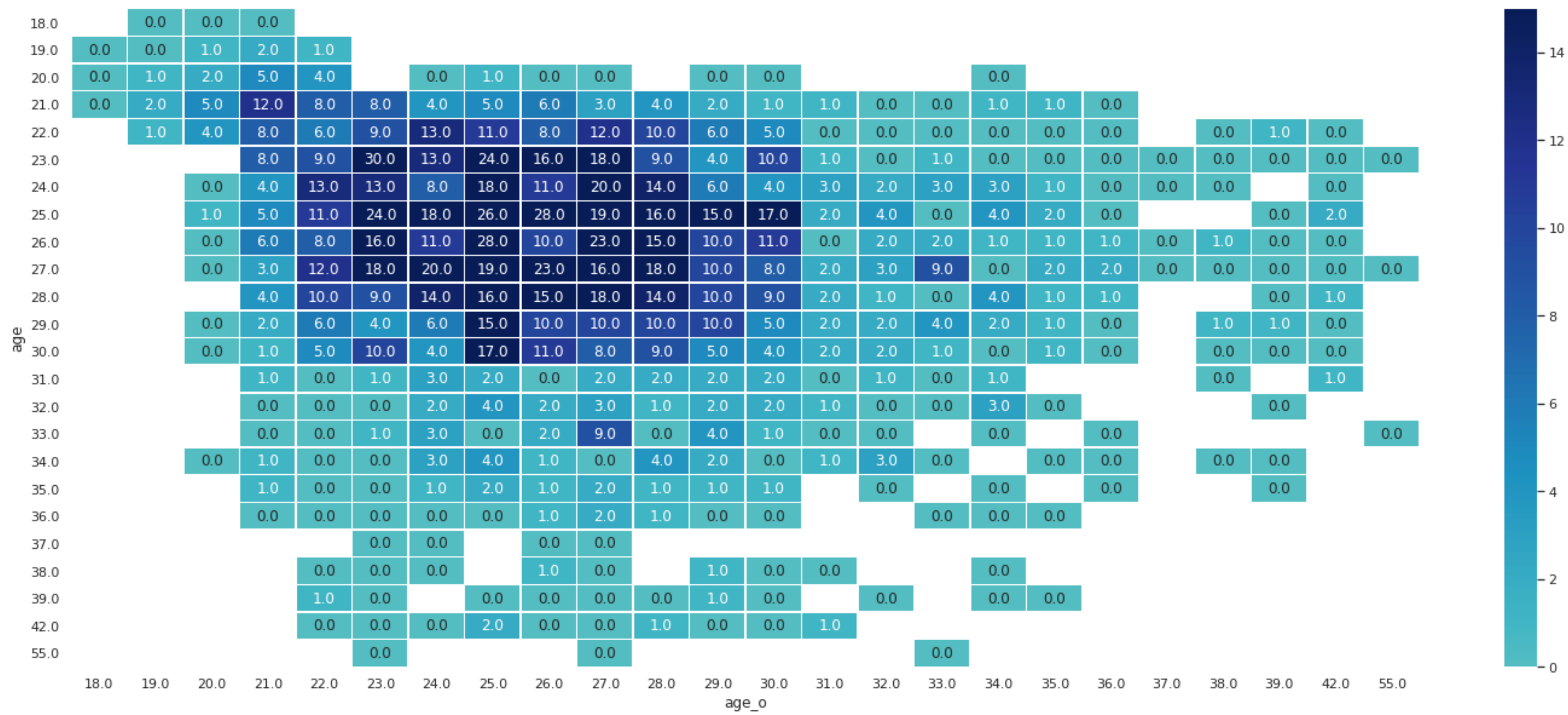
```
# violinplot that shows the distribution of age for the most frequent professions
# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(30, 10))

# Draw a violinplot with a narrower bandwidth than the default
sns.violinplot(y = dating.age,x=dating.frequent_professions, palette="muted", bw=.2, cut=1, linewidth=1)

# Finalize the figure
ax.set(ylim=(18, 50))
ax.set(xlim=(None, None))
sns.despine(left=True, bottom=True)
```



In [43]:

```
# heatmap of the ages of the persons who took the survey and their dates
import numpy as np
table = pd.pivot_table(dating, values='match', index="age",
                    columns="age_o", aggfunc=np.sum)

# Draw a heatmap with the numeric values in each cell
f, ax = plt.subplots(figsize=(25, 10))
sns.heatmap(table, annot=True, vmin = 0, vmax = 15,  linewidths=.5, ax=ax, center =1, fmt = ".1f",cmap='
```

Out[43]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5fc0f74a58>



In [50]:
```
dating.dropna(inplace=True) # drop the rows with at least one empty value
```

In [51]:
```
# define the predictors and the outcome of the machine learning algorithms
predictors = ['age', 'age_o', 'samerace',
       'pref_o_attractive', 'pref_o_sincere', 'pref_o_intelligence',
       'pref_o_funny', 'pref_o_ambitious',
       'pref_o_shared_interests', 'attractive_o',
       'intelligence_o', 'funny_o',
       'shared_interests_o', 'sports', 'exercise',
       'dining', 'museums',
        'art', 'hiking',
       'gaming', 'clubbing', 'reading', 'tv']
outcome = 'match'
```

In [52]:
```
X = dating[predictors]
y = dating[outcome]
X.shape
```

Out[52]:
```
(1048, 23)
```

In [53]:
```
train_X, valid_X, train_y, valid_y = train_test_split(X, y, test_size=0.3, random_state=1)
# divide the dataset into training and test
```

In [54]:
```
from sklearn.tree import DecisionTreeClassifier
from dmba import plotDecisionTree
```

Logistic Regression

In [55]:
```
# apply the logistic regression
logit_reg = LogisticRegression(penalty="l2", C=1e42, solver='liblinear',class_weight = 'balanced')
logit_reg.fit(train_X, train_y)
```

Out[55]:
```
LogisticRegression(C=1e+42, class_weight='balanced', dual=False,
                   fit_intercept=True, intercept_scaling=1, l1_ratio=None,
                   max_iter=100, multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=None, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [56]:
```
print(pd.DataFrame({'coeff': logit_reg.coef_[0]}, index=X.columns)) # the weight of each attribute
```

```
                          coeff
age                   -0.040752
age_o                 -0.099627
samerace               0.006123
pref_o_attractive     -0.020162
pref_o_sincere         0.007752
pref_o_intelligence   -0.011214
pref_o_funny           0.022069
pref_o_ambitious      -0.035991
pref_o_shared_interests -0.004772
attractive_o           0.228624
intelligence_o        -0.075500
funny_o                0.190518
shared_interests_o     0.362429
sports                 0.103954
exercise              -0.151861
dining                -0.062621
museums                0.156020
art                    0.073188
hiking                 0.008262
gaming                -0.092739
clubbing               0.053732
reading               -0.010917
tv                     0.054569
```

In [57]:

```
# print the metrics of the model
from sklearn.metrics import accuracy_score, precision_score,recall_score,f1_score,roc_auc_score,classifi
lr_prediction_train = logit_reg.predict_proba(train_X)[:,1] > 0.5
lr_prediction_valid = logit_reg.predict_proba(valid_X)[:,1] > 0.5
print("Accuracy on train is:",accuracy_score(train_y,lr_prediction_train))
print("Accuracy on test is:",accuracy_score(valid_y,lr_prediction_valid))
print("Precision_score train is:",precision_score(train_y,lr_prediction_train))
print("Precision_score on test is:",precision_score(valid_y,lr_prediction_valid))
print("Recall_score on train is:",recall_score(train_y,lr_prediction_train))
print("Recall_score on test is:",recall_score(valid_y,lr_prediction_valid))
print("f1_score on train is:",f1_score(train_y,lr_prediction_train))
print("f1_score on test is:",f1_score(valid_y,lr_prediction_valid))
```

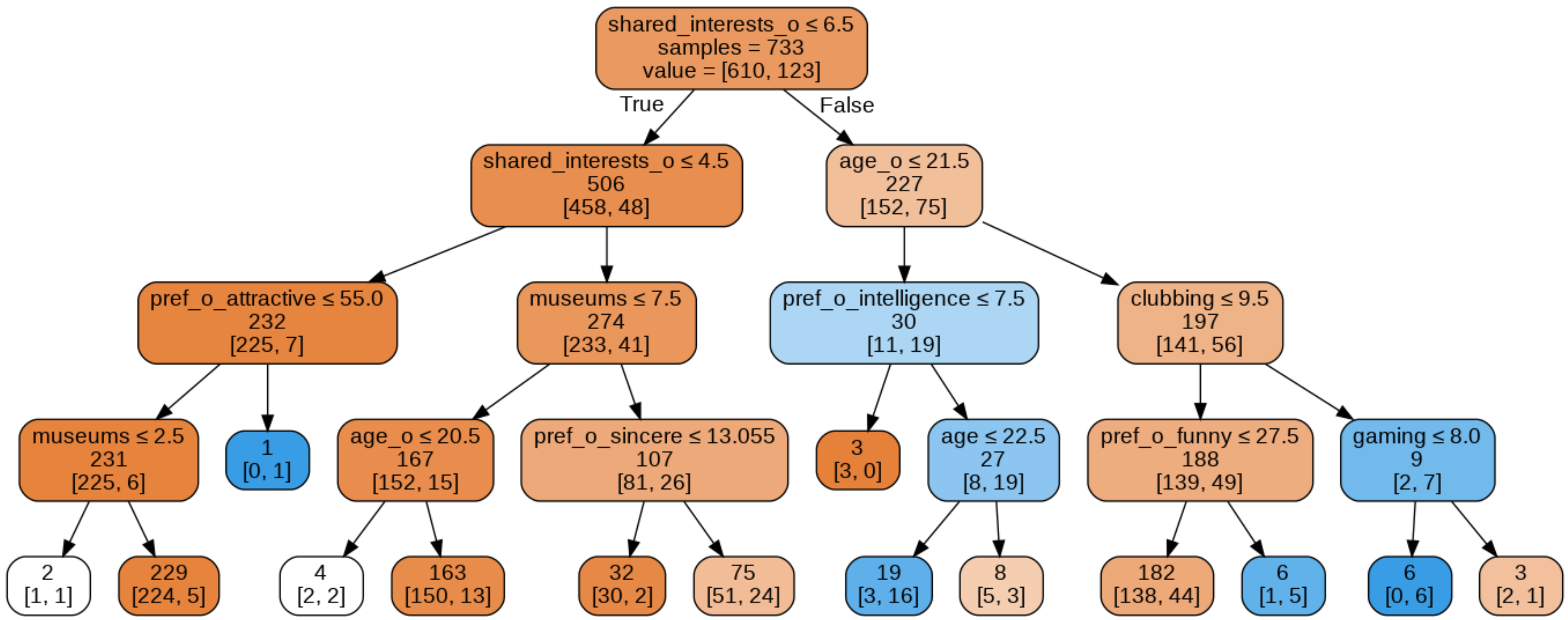
```
Accuracy on train is: 0.73806275579809
Accuracy on test is: 0.6761904761904762
Precision_score train is: 0.3657587548638132
Precision_score on test is: 0.3274336283185841
Recall_score on train is: 0.7642276422764228
Recall_score on test is: 0.5873015873015873
f1_score on train is: 0.49473684210526314
f1_score on test is: 0.4204545454545455
```

Decision Tree

In [58]:

```python
# apply the decision tree model
DecisionTree = DecisionTreeClassifier(max_depth = 4)
DecisionTree.fit(train_X, train_y)

plotDecisionTree(DecisionTree, feature_names=train_X.columns)
```

Out[58]:



In [59]:

```
importances = DecisionTree.feature_importances_

im = pd.DataFrame({'feature': train_X.columns, 'importance': importances})
im = im.sort_values('importance',ascending=False)
print(im)
```

```
                   feature  importance
12       shared_interests_o    0.395000
1                     age_o    0.145472
20                 clubbing    0.086523
6              pref_o_funny    0.076571
16                  museums    0.074710
4            pref_o_sincere    0.056023
5       pref_o_intelligence    0.050365
0                       age    0.046269
3         pref_o_attractive    0.035583
19                   gaming    0.033483
15                   dining    0.000000
21                  reading    0.000000
18                   hiking    0.000000
17                      art    0.000000
11                  funny_o    0.000000
14                 exercise    0.000000
13                   sports    0.000000
10           intelligence_o    0.000000
9              attractive_o    0.000000
8   pref_o_shared_interests    0.000000
7         pref_o_ambitious    0.000000
2                  samerace    0.000000
22                       tv    0.000000
```

In [60]:

```
dt_prediction_train = DecisionTree.predict(train_X)
dt_prediction_valid = DecisionTree.predict(valid_X)

print("Accuracy score on train is:",accuracy_score(train_y,dt_prediction_train))
print("Accuracy score on test is:",accuracy_score(valid_y,dt_prediction_valid))
print("Precision score on train is:",precision_score(train_y,dt_prediction_train))
print("Precision score on test is:",precision_score(valid_y,dt_prediction_valid))
print("Recall score on train is:",recall_score(train_y,dt_prediction_train))
print("Recall score on test is:",recall_score(valid_y,dt_prediction_valid))
print("F1 score on train is:",f1_score(train_y,dt_prediction_train))
print("F1 score on test is:",f1_score(valid_y,dt_prediction_valid))
```

```
Accuracy score on train is: 0.8649386084583902
Accuracy score on test is: 0.8095238095238095
Precision score on train is: 0.875
Precision score on test is: 0.5882352941176471
Recall score on train is: 0.22764227642276422
Recall score on test is: 0.15873015873015872
F1 score on train is: 0.3612903225806452
F1 score on test is: 0.25
```

Gradient Boosted Tree

In [61]:

```
from sklearn.ensemble import GradientBoostingClassifier

gbm = GradientBoostingClassifier(random_state=0)
gbm.fit(train_X, train_y)
gbm.predict(valid_X[:2])
```

Out[61]:

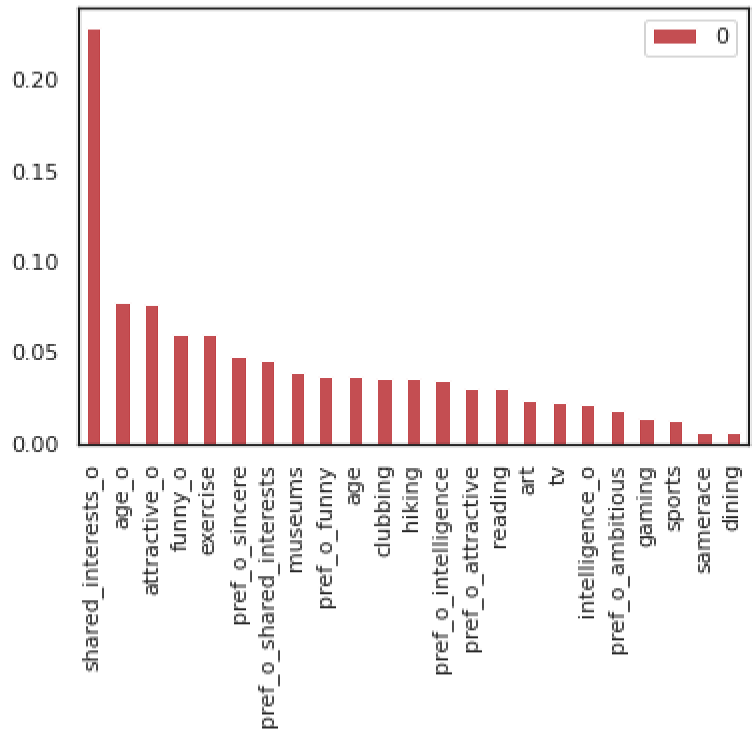
```
array([0, 0])
```

In [62]:

```
importances = list(zip(gbm.feature_importances_, valid_X.columns))
pd.DataFrame(importances, index=[x for (_,x) in importances]).sort_values(by = 0, ascending = False).plo
```

Out[62]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f5fc423f2e8>
```



In [63]:

```python
gbt_prediction_train = gbm.predict(train_X)
gbt_prediction_valid = gbm.predict(valid_X)

print("Accuracy on train is:",accuracy_score(train_y,gbt_prediction_train))
print("Accuracy on test is:",accuracy_score(valid_y,gbt_prediction_valid))
print("Precision_score train is:",precision_score(train_y,gbt_prediction_train))
print("Precision_score on test is:",precision_score(valid_y,gbt_prediction_valid))
print("Recall_score on train is:",recall_score(train_y,gbt_prediction_train))
print("Recall_score on test is:",recall_score(valid_y,gbt_prediction_valid))
print("f1_score on train is:",f1_score(train_y,gbt_prediction_train))
print("f1_score on test is:",f1_score(valid_y,gbt_prediction_valid))
```

```
Accuracy on train is: 0.9399727148703957
Accuracy on test is: 0.8158730158730159
Precision_score train is: 0.9876543209876543
Precision_score on test is: 0.6086956521739131
Recall_score on train is: 0.6504065040650406
Recall_score on test is: 0.2222222222222222
f1_score on train is: 0.7843137254901961
f1_score on test is: 0.3255813953488372
```

In [ ]:

```python

```