In [28]:
```
%matplotlib inline

import math
from enum import IntEnum

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import graphviz

from sklearn import tree, metrics, svm
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelBinarizer, Imputer
```

In [29]:
```
def convert_raw_csv(input_file, output_file):
    df = pd.read_csv(input_file, header = 0, sep=',', thousands=',')
    toScale = ['attr1_1','sinc1_1','intel1_1','fun1_1','amb1_1','shar1_1',
              'attr2_1','sinc2_1','intel2_1','fun2_1','amb2_1','shar2_1',
              'attr3_1','sinc3_1','intel3_1','fun3_1','amb3_1',
              'attr4_1','sinc4_1','intel4_1','fun4_1','amb4_1','shar4_1',
              'attr5_1','sinc5_1','intel5_1','fun5_1','amb5_1']

    def scaleAttrs(r):
        for group in [toScale[0:6],toScale[6:12],toScale[12:17],toScale[17:23]
,toScale[23:28]]:
            s = np.sum(r[group])
            assert not s == 0 and not s == np.isnan(s)
            r[group] = r[group]/s
        return r

    df[toScale] = df[toScale].apply(scaleAttrs, axis=1)
    df.to_csv(output_file, index=False)
```

In [30]:
```
def with_pAge(df):
    df = df.copy()
    ages = df[['iid','age']].groupby(['iid']).mean()
    df['pAge'] = df['pid'].apply(lambda x: math.nan if math.isnan(x) else ages
.age[x])
    return df
```

In [31]:
```
def impute(X, verbose=False):
    # Copy to avoid looping over the array we're modifying
    cols = X.columns.values
    for col in cols:
        if X[col].dtypes=='object':
            #print('Classifying {0}'.format(col))
            X = X.drop(col, axis=1)
            if verbose:
                print('Dropping column {0}'.format(col))
            # This is really heavy
            #classes = X[col].str.get_dummies().rename(columns=lambda x: 'fiel
d-{0}'.format(x).replace(' ',''))
            #X = pd.concat([X,classes])
        elif X[col].dtypes=='float64' and X[col].isnull().values.any():
            assert not col == 'iid' and not col == 'id' and not col == 'idg'
            #print('Imputing {0}'.format(col))
            # Fill in missing values
            if col == 'field_cd' or \
                col == 'gender' or \
                col == 'undergrd' or \
                col == 'race' or \
                col == 'from' or \
                col == 'career_c':
                X[[col]]=Imputer(missing_values='NaN', strategy='most_frequent
', axis=0).fit_transform(X[[col]])
            else:
                X[[col]]=Imputer(missing_values='NaN', strategy='mean', axis=0
).fit_transform(X[[col]])
    return X
```

In [32]:
```
# Preprocess data
def preprocess(df, verbose=False):
    return impute(df.drop(columns=['iid', 'id', 'idg', 'condtn', 'wave', 'roun
d', 'position',
                             'positin1', 'order', 'partner', 'pid',
                             'zipcode', # zipcode -> income
                             #'undergra', -> {mn_sat, tuition}
                             'attr', 'sinc', 'intel', 'fun', 'amb', 'shar', 'li
ke', 'prob',
                             'match',
                             #'gender',
                             'you_call', 'them_cal', 'date_3', 'numdat_3', 'num
_in_3',
                            ], errors='ignore'), verbose=verbose)
```

In [33]:
```
def splitBy(df, attr):
    return df.drop(columns=[attr]), df[attr]
```

In [34]:
```
def model(X,y,test_size=0.2,random_state=0,min_samples_split=0.02, max_depth=1
0, accuracy_file=None, print_stats=True):
    X_train, X_test, y_train, y_test = train_test_split(
     X, y, test_size=test_size, random_state=random_state)

    clf = tree.DecisionTreeClassifier(min_samples_split=min_samples_split, max
_depth=max_depth)
    clf = clf.fit(X_train, y_train)

    y_predict = clf.predict(X_test)

    accuracy = metrics.accuracy_score(y_test, y_predict)
    tn, fp, fn, tp = metrics.confusion_matrix(y_test, y_predict).ravel()/len(y
_test)
    if print_stats:
        accuracy_str = """Accuracy: {0:.2f}%
True negatives: {1:.2f}%\tFalse negatives: {2:.2f}%
False positives: {3:.2f}%\tTrue positives: {4:.2f}%\n""".format(
            accuracy*100, tn*100, fp*100, fn*100, tp*100)
        print(accuracy_str)

    if not accuracy_file == None:
        with open(accuracy_file, 'w') as f:
            f.write(accuracy_str)
    return clf
```

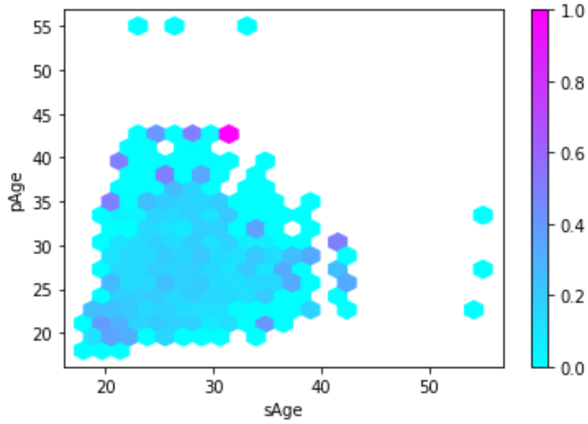
In [35]:
```
def vizualize(model, columns, out_file=None):
    graph = graphviz.Source(
        tree.export_graphviz(model, out_file=None,
                             feature_names=columns,
                             filled=True, rounded=True,
                             special_characters=True))
    if not out_file == None:
        graph.render(out_file)

    return graph
```

In [36]:
```
class Gender(IntEnum):
    FEMALE = 0;
    MALE = 1;
```

In [37]:
```
# Scaling the attrs turned out to be really slow, so store preprocessed data.
#convert_raw_csv("data.csv", "data_converted.csv")
```

In [38]:
```
df = pd.read_csv("data_converted.csv", header=0, sep=',')
```

In [39]:
```
with_pAge(df).rename({'age':'sAge'}, axis='columns').plot.hexbin(
    x='sAge', y='pAge', C='match',
    cmap=plt.cm.cool,
    reduce_C_function=np.mean,
    gridsize=22,
    sharex=False, sharey=False)
plt.show()
```

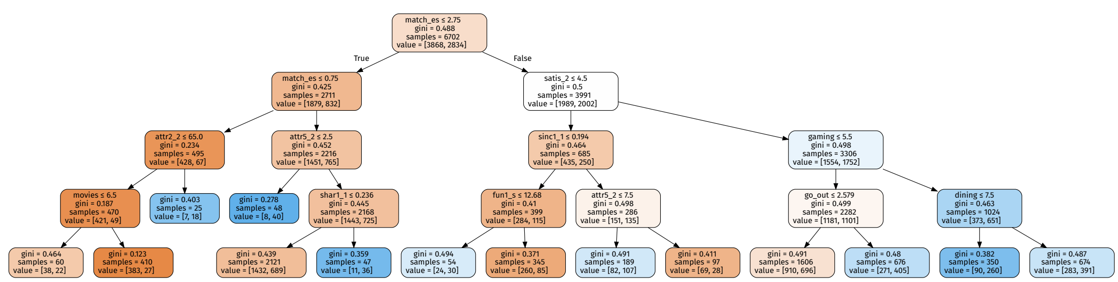


In [40]:
```
X, Y = splitBy(preprocess(df), 'dec')
X = X.drop(columns=['gender'])
```

In [41]:
```
uni_model = model(X, Y, test_size=0.2)
vizualize(model(X, Y, test_size=0.2, max_depth=4, print_stats=False), X.column
s)
```

```
Accuracy: 68.26%
True negatives: 47.37%  False negatives: 11.81%
False positives: 19.93% True positives: 20.88%
```

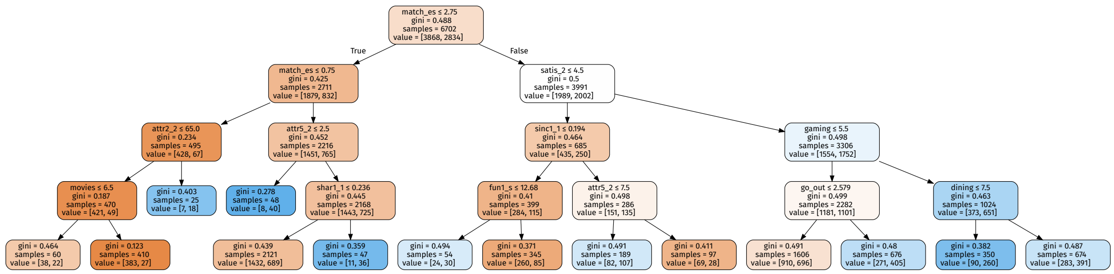
Out[41]:



In [42]:
```
X_nobias = impute(X.drop(["race","imprace","imprelig","income"], axis=1))
nobias_model = model(X_nobias,Y, test_size=0.2)
vizualize(model(X_nobias,Y, max_depth=4, test_size=0.2, print_stats=False), X_
nobias.columns)
```

```
Accuracy: 68.32%
True negatives: 47.32%  False negatives: 11.87%
False positives: 19.81% True positives: 21.00%
```

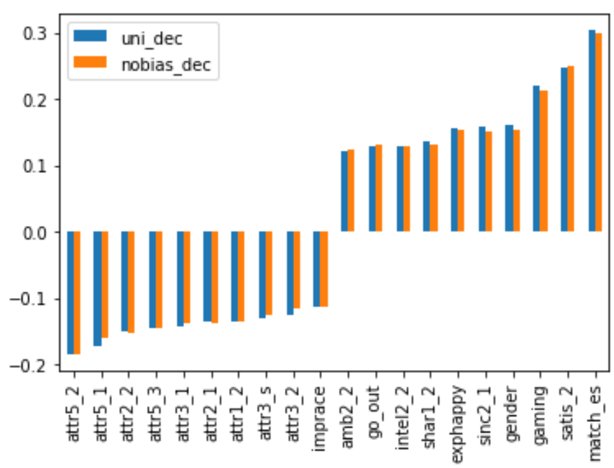
Out[42]:

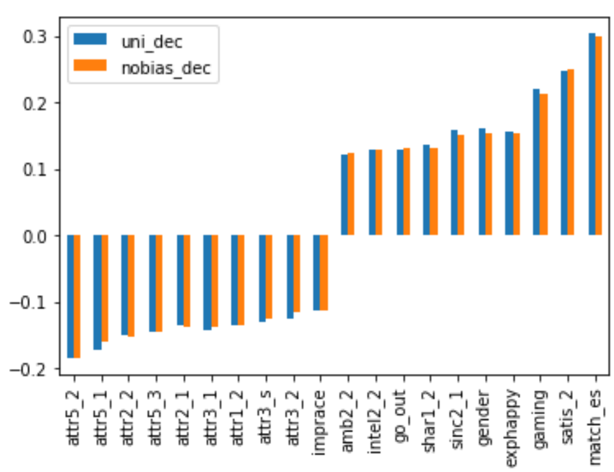


In [43]:
```
def printDiscriminationScore(attr, d_uni, d_nobias):
    print("""Discrimination score(slift) towards {0}:
 Unisex model: {1}
 No bias model: {2}""".format(attr, d_uni, d_nobias))
```

In [44]:
```
def discriminationScore(df):
    means = df[['dec','nobias_dec','uni_dec']].mean()
    d_uni = abs(means['dec']-means['uni_dec'])
    d_nobias = abs(means['dec']-means['nobias_dec'])
    return d_uni, d_nobias
```

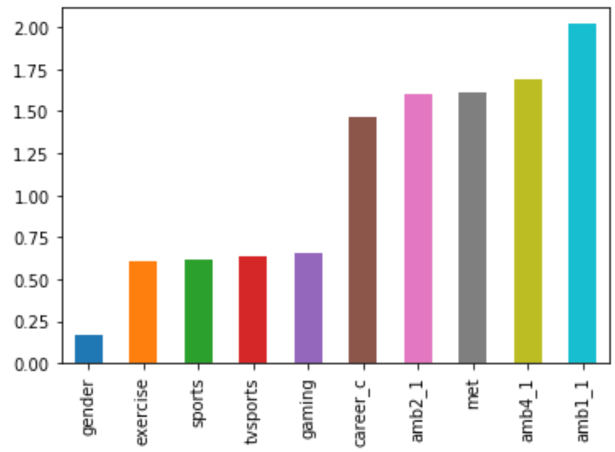
In [45]:
```
df_dec = impute(preprocess(df.copy()))

df_dec['uni_dec'] = uni_model.predict(X.as_matrix())
df_dec['nobias_dec'] = nobias_model.predict(X_nobias.as_matrix())

print()

### Pearson coefficent of correlation between attributes and decisions
corr = df_dec.corr().drop(['uni_dec', 'nobias_dec', 'dec'])

### Sorted by unisex model correlation
uni_corr = corr[['uni_dec', 'nobias_dec']].sort_values(by='uni_dec')
uni_corr.head(10).append(uni_corr.tail(10)).plot.bar()
plt.show()

### Sorted by no-bias model correlation
nobias_corr = corr[['uni_dec', 'nobias_dec']].sort_values(by='nobias_dec')
nobias_corr.head(10).append(nobias_corr.tail(10)).plot.bar()
plt.show()

printDiscriminationScore('gender', *discriminationScore(df_dec[df_dec.gender =
= 1]))
```






```
Discrimination score(slift) towards gender:
 Unisex model: 0.06938483547925606
 No bias model: 0.06819265617548875
```

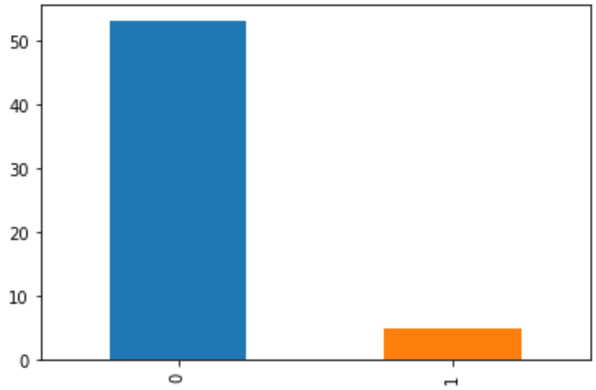
In [46]:
```
not_equal = df_dec[df_dec.nobias_dec != df_dec.uni_dec]
not_equal_mean = not_equal.mean().drop(['uni_dec', 'nobias_dec', 'dec'])
df_dec_mean = df_dec.mean().drop(['uni_dec', 'nobias_dec', 'dec'])

### Relative difference in means of attributes between general data set and ca
ses, where model decisions differ
not_equal_prop = (not_equal_mean/df_dec_mean).sort_values()
not_equal_prop.head(5).append(not_equal_prop.tail(5)).plot.bar()
plt.show()
```



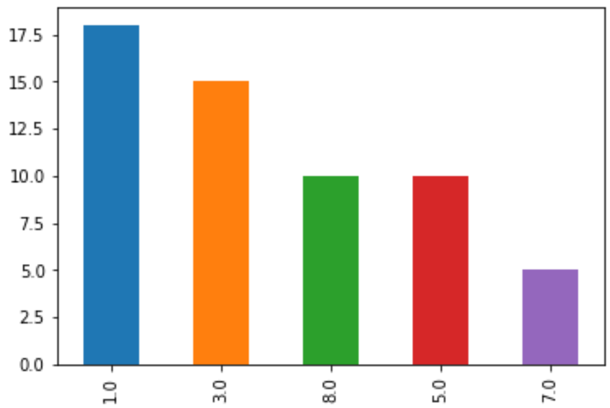
In [47]:

```
print("In the cases, where decision between model differs:")
print("Gender:")
not_equal.gender.value_counts().plot.bar()
plt.show()

print("Sports:")
not_equal.sports.value_counts().plot.bar()
plt.show()

print("Gaming:")
not_equal.gaming.value_counts().plot.bar()
plt.show()

print("Excercise:")
not_equal.exercise.value_counts().plot.bar()
plt.show()

print("Race:")
not_equal.race.value_counts().plot.bar()
plt.show()

print("amb1_1:")
not_equal.amb1_1.value_counts().plot.bar()
plt.show()

print("amb2_1:")
not_equal.amb2_1.value_counts().plot.bar()
plt.show()

print("amb4_1:")
not_equal.amb4_1.value_counts().plot.bar()
plt.show()

print("shopping:")
not_equal.shopping.value_counts().plot.bar()
plt.show()

print("career_c")
not_equal.career_c.value_counts().plot.bar()
plt.show()
```

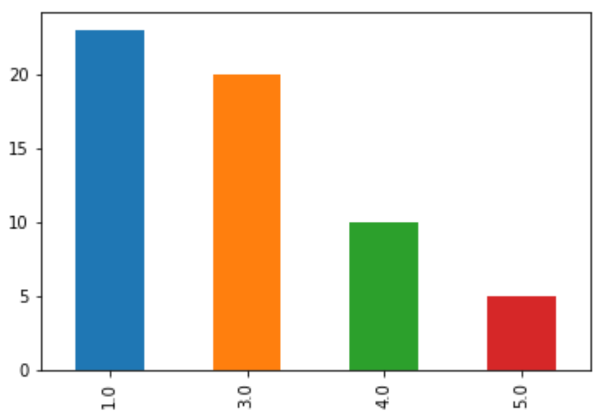
```
In the cases, where decision between model differs:
Gender:
```



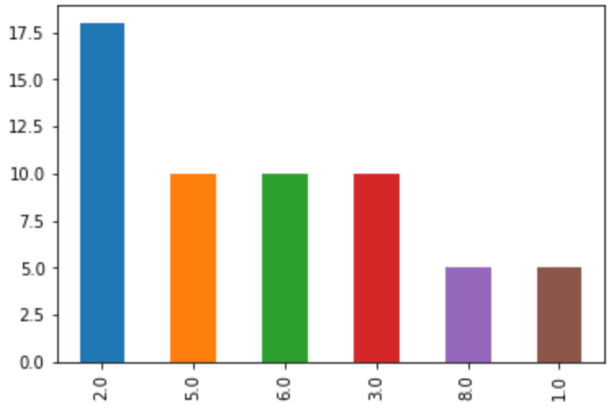

```
Sports:
```



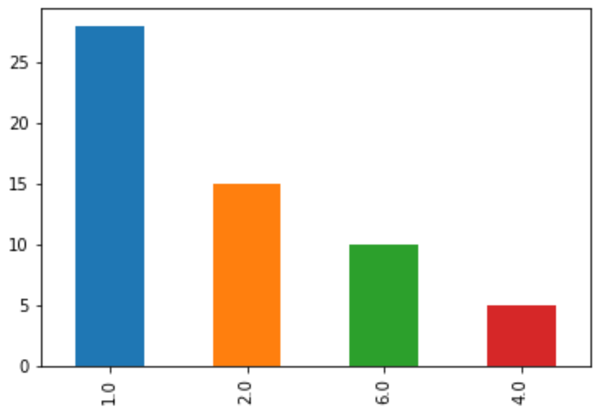

```
Gaming:
```



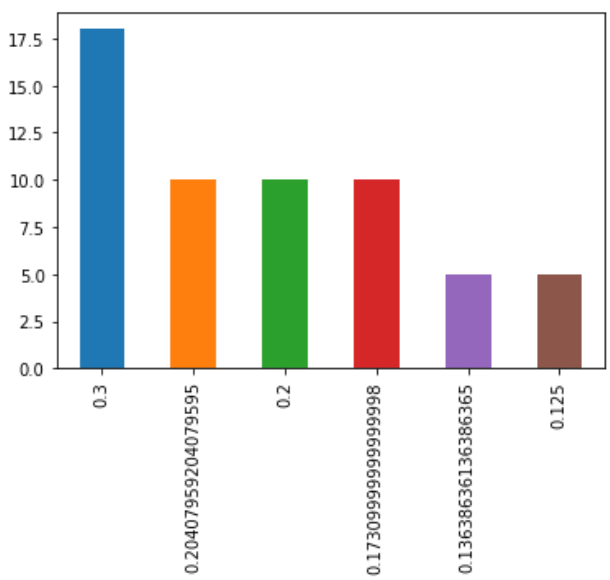

```
Excercise:
```



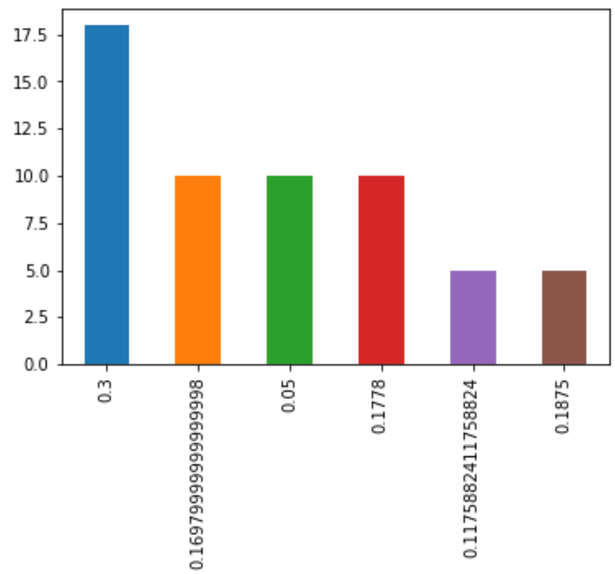
Race:



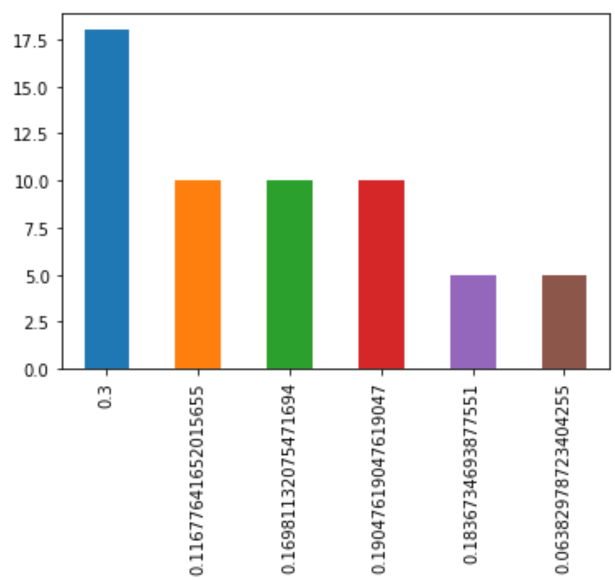
amb1_1:



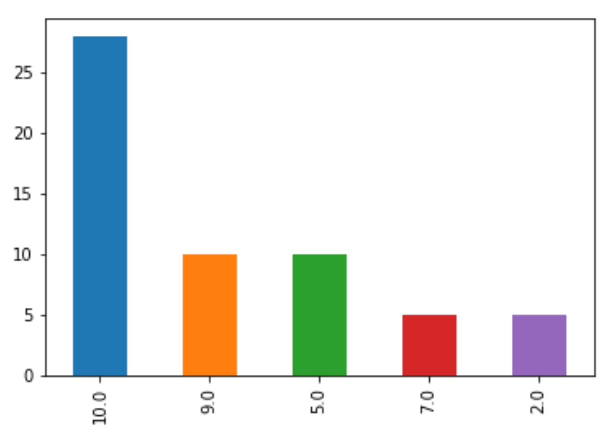
amb2_1:



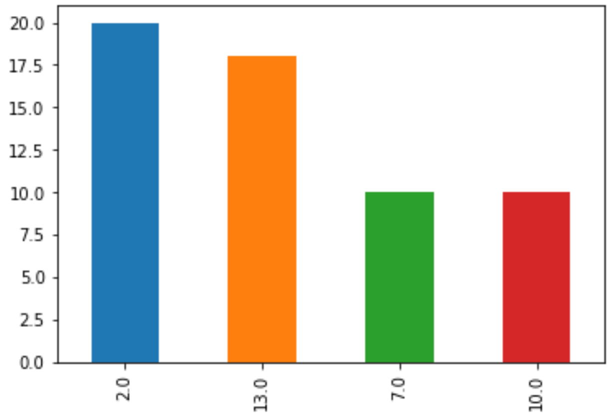

```
amb4_1:
```



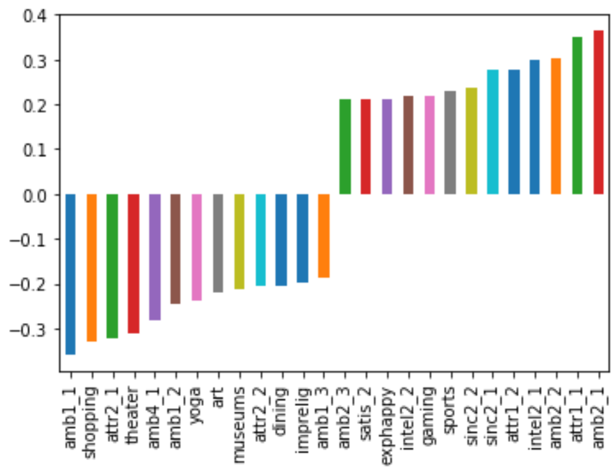

```
shopping:
```



career_c



In [48]:
```
df = preprocess(df)

### Pearson coefficent of correlation between gender and attributes
corr = df.corr()['gender'].drop('gender').sort_values()
corr.head(13).append(corr.tail(13)).plot.bar()
plt.show()
```



In [49]:
```
def computeElift(df, A, B, C):
    AB_df = df.query('{0} and {1}'.format(A,B))
    B_df = df.query(B)
    return ((AB_df.query(C).count()/AB_df.count()) / (B_df.query(C).count()/ B
_df.count()))[0]

elift = computeElift(df, 'amb1_1 > {0}'.format(df.amb1_1.mean()), 'age > 20 an
d age < 40', 'gender == {0}'.format(Gender.FEMALE))
print('''Elift for
A: amb1_1 > {0}
B: 20 < age < 40
C: gender == 0 (female)
is: {1}'''.format(df.amb1_1.mean(), elift))
```

```
Elift for
A: amb1_1 > 0.1065982659354377
B: 20 < age < 40
C: gender == 0 (female)
is: 1.3356487789464861
```