COMPAS Analysis-Jack DeGesero

November 11, 2023

1 Analysis of COMPAS Score, Detecting Inaccuracies

1.1 Jack DeGesero

1.1.1 The data regarded predicts whether or not criminal defendants are likely to be reoffenders based on multiple attributes, in this report we will be examining sex, age (binned in three categories: <25, 25-45, >45), decile score, and if they did re-offend or not within a two year time frame (is a recidivist), our class attribute. All the data examined is sourced from Broward County, FL.

This data is sourced from Pro Publica, who initially led the report with all variables. https://www.propublica.org/article/how-we-analyzed-the-compas-recidivism-algorithm

[1]:
```
#import packages

import numpy as py #series'
import pandas as pd #dataframes
import statistics #stats

import matplotlib.pyplot as plt #graphing

from sklearn import tree #for tree
from sklearn import model_selection #for partition into test and training data
from sklearn import preprocessing #to change attributes
from sklearn.metrics import accuracy_score #for checking model accuracy
from sklearn.metrics import classification_report, confusion_matrix #to show␣
↪confusion matrix
```

[3]:
```
#load the data
df1 = pd.read_csv('compas_small.csv')
df2 = pd.read_csv('compas_small_Ca.csv')
df3 = pd.read_csv('compas_small_AfAm.csv')

#get all relevant columns, and class attribute (is_recid)
df1 = df1[['sex', 'age_cat', 'decile_score', 'is_recid']]
df2 = df2[['sex', 'age_cat', 'decile_score', 'is_recid']]
df3 = df3[['sex', 'age_cat', 'decile_score', 'is_recid']]
```

[4]:
```
df1 #All races
```

[4]:

```
        sex         age_cat  decile_score is_recid
0      Male  Greater than 45             1       no
1      Male          25 - 45             3      yes
2      Male     Less than 25             4      yes
3      Male     Less than 25             8       no
4      Male          25 - 45             1       no
…       …              …           …      …
7209   Male     Less than 25             7       no
7210   Male     Less than 25             3       no
7211   Male  Greater than 45             1       no
7212 Female          25 - 45             2       no
7213 Female     Less than 25             4      yes

[7214 rows x 4 columns]
```

[5]:

```
df2 #Caucasians
```

[5]:

```
        sex      age_cat  decile_score is_recid
0      Male      25 - 45             6      yes
1    Female      25 - 45             1       no
2      Male Less than 25             3      yes
3      Male      25 - 45             4       no
4    Female      25 - 45             1       no
…       …           …           …      …
2449   Male      25 - 45             2       no
2450 Female      25 - 45             1      yes
2451   Male Less than 25             8       no
2452   Male Less than 25            10      yes
2453   Male Less than 25             6      yes

[2454 rows x 4 columns]
```

[6]:

```
df3 #African Americans
```

[6]:

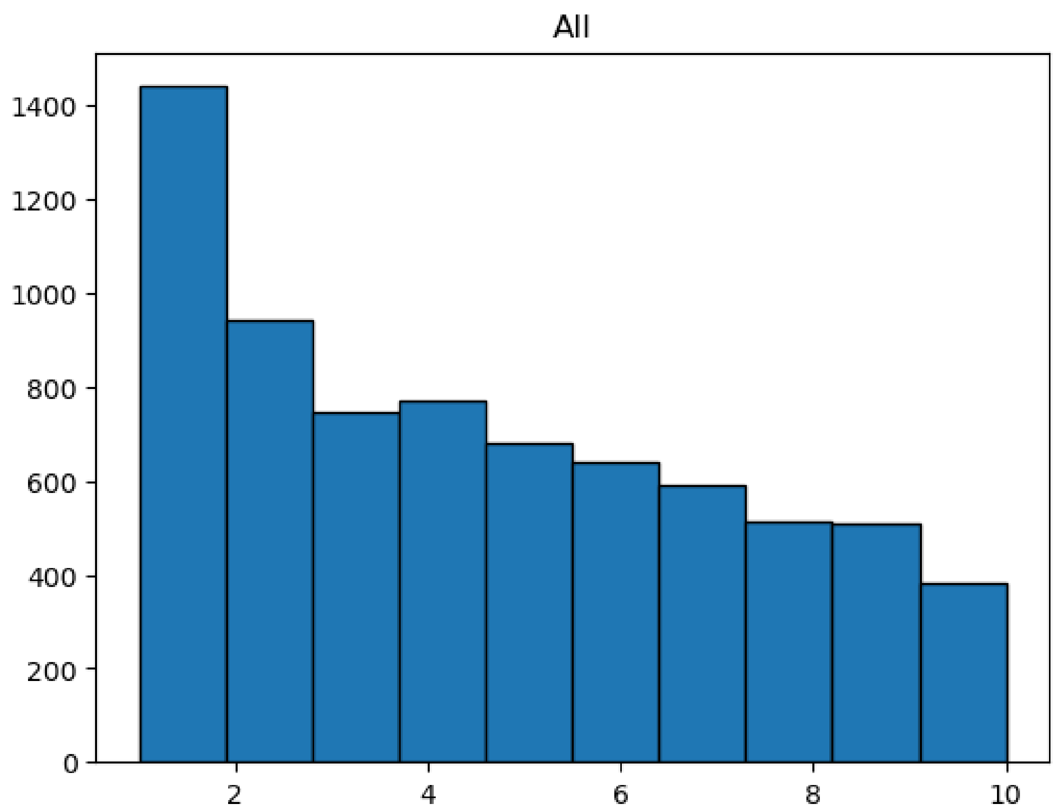
```
        sex      age_cat  decile_score is_recid
0      Male      25 - 45             3      yes
1      Male Less than 25             4      yes
2      Male Less than 25             8       no
3      Male Less than 25             6      yes
4      Male      25 - 45             4       no
…       …           …           …      …
3691   Male      25 - 45             2      yes
3692   Male Less than 25             9       no
3693   Male Less than 25             7       no
3694   Male Less than 25             3       no
3695 Female      25 - 45             2       no
```

[3696 rows x 4 columns]

[7]:
```
#Check if any na values are present in each data set
any([df1['decile_score'].isna().any(),df2['decile_score'].isna().
↪any(),df3['decile_score'].isna().any()])
```

[7]: False

[8]:
```
plt.hist(df1['decile_score'], ec='black')
plt.title("All")
plt.show()
```

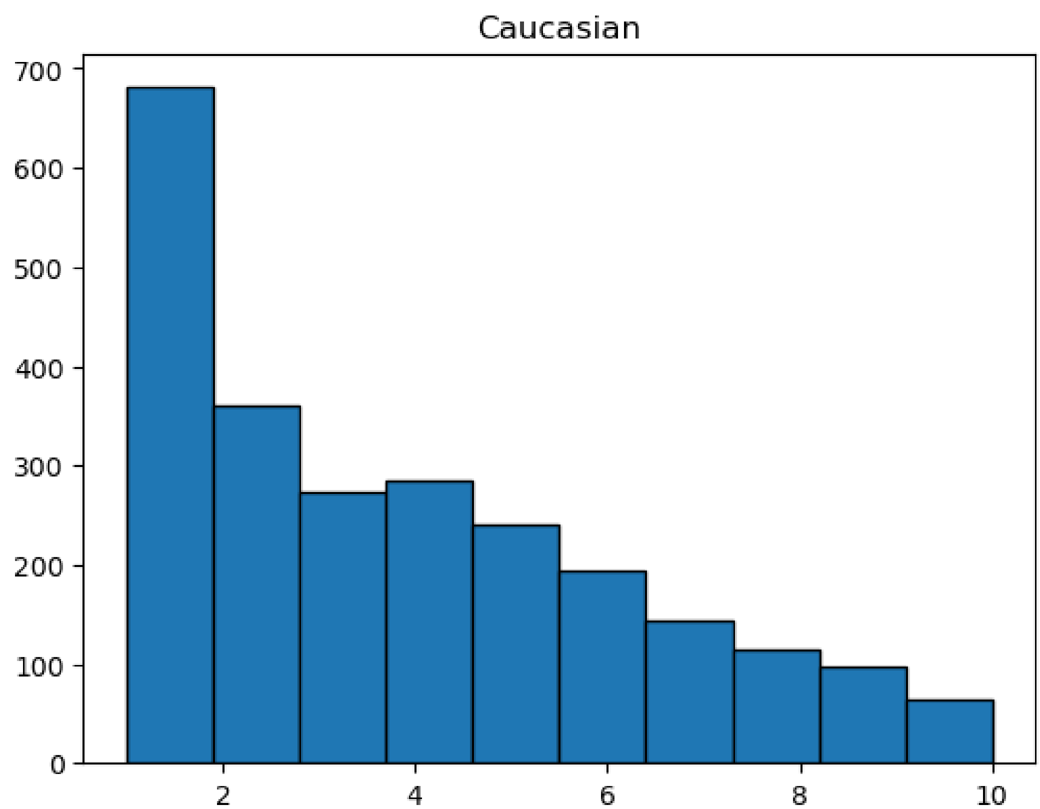


[9]:
```
print("Median of All: " + str(df1['decile_score'].median()) + ", Mode of All: "␣
↪+str(df1['decile_score'].mode().values[0]))
```

Median of All: 4.0, Mode of All: 1

1.1.2 From the figure above, we can see the median for decile score is greater than the mean indicating it is positively skewed.

[12]:
```
plt.hist(df2['decile_score'], ec='black')
plt.title("Caucasian")
plt.show()
```

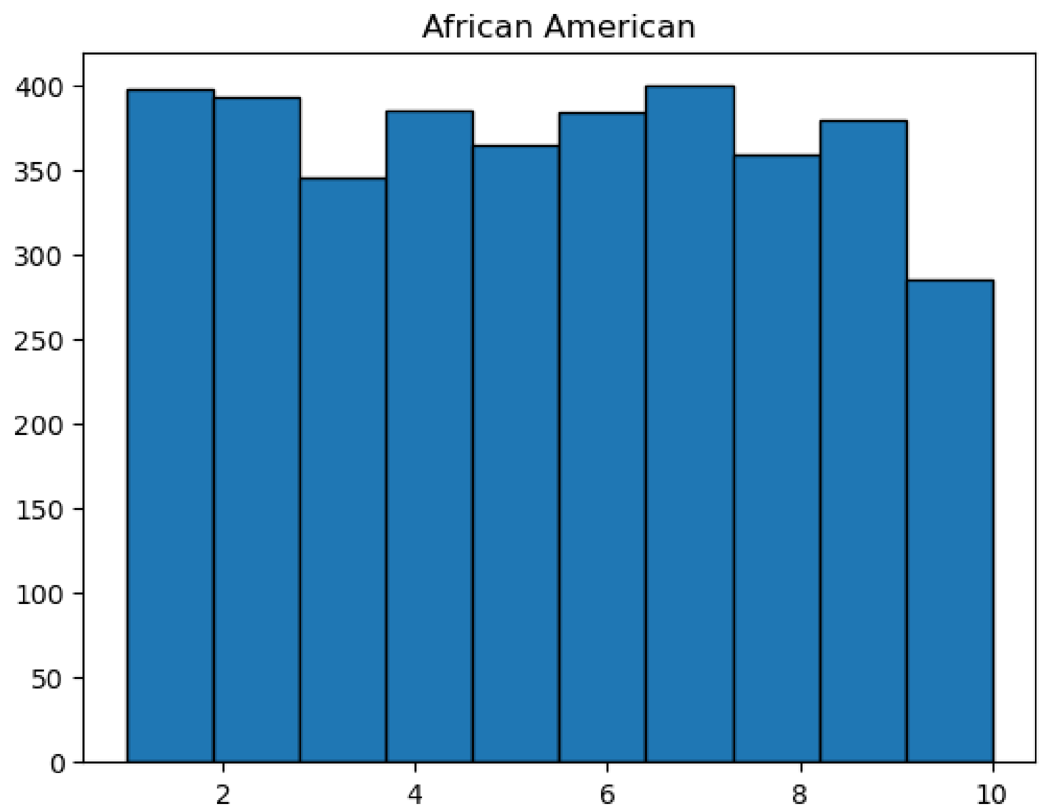


[13]:
```
print("Median of Caucasians: " + str(df2['decile_score'].median()) + ", Mode of␣
↪Caucasians: "+ str(df2['decile_score'].mode().values[0]))
```

```
Median of Caucasians: 3.0, Mode of Caucasians: 1
```

1.1.3 From this figure, we can see the histogram for Caucasians is also positively skewed. It's distributions are similar to the set of all.

[11]:
```
plt.hist(df3['decile_score'], ec='black')
plt.title("African American")
plt.show()
```



[12]:
```
print("Median of African Americans: " + str(df3['decile_score'].median()) + ", Mode of African Americans: "+ str(df1['decile_score'].mode().values[0]))
```

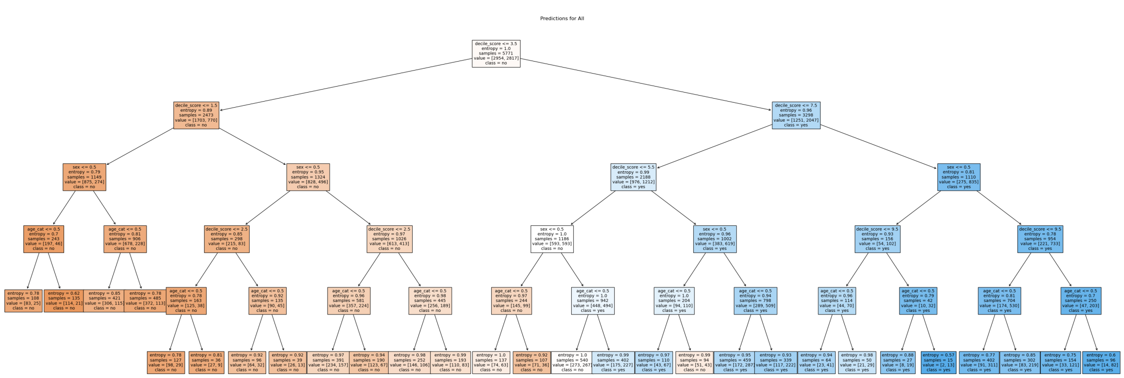
```
Median of African Americans: 5.0, Mode of African Americans: 1
```

1.1.4 Finally, African Americans have a slight positive skew; however, the median is higher than both the latter graphs indicating it's more evenly skewed (i.e. more entries with higher decile scores). This will effect this groups participation in the class attribute.

[29]:
```
#Preprocess some attributes to make scikit more digestible
df1['sex'] = preprocessing.LabelEncoder().fit(df1['sex']).transform(df1['sex'])
df1['age_cat'] = preprocessing.OneHotEncoder(sparse=False).fit_transform(df1['age_cat'].values.reshape(-1, 1))
```

[30]:
```
#Make Trained Decision Tree for 'All' (df1)

#Grab attributes and class attribute
allAtr = df1[['sex', 'age_cat', 'decile_score']]
classAtr = df1['is_recid']
```

```python
#Partition 20% of data to be tested, map to xtr-Training Attributes, xt-Test Attributes, ytr-Class Training Attributes, yt-Class Test Attributes (used for accuracy)
xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr, test_size=0.2, random_state=42)

#instantiates tree object
AllTree = tree.DecisionTreeClassifier(criterion="entropy",max_depth=5) #depth limited to 5 for visualization purposes

#make the tree
AllTree.fit(xtr, ytr)

#predict based on test data
prediction = AllTree.predict(xt)

#plot data
plt.figure(figsize=(48, 16))
tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=['no', 'yes'], filled=True, impurity=True, precision=2, fontsize=10)
plt.title('Predictions for All')
plt.savefig('decision_tree.png', dpi=300) #very low res in out[]:, see picture
plt.show()
```



[31]:
```python
importances = AllTree.feature_importances_ #get gain of each attribute examined, put in list

print("Gain:")

# Print the attributes in the order of importance
for i in xtr.columns:
    print(f'Attribute: {i}, Importance: {importances[xtr.columns.get_loc(i)]:.2f}')
```

```
Gain:
Attribute: sex, Importance: 0.05
Attribute: age_cat, Importance: 0.03
Attribute: decile_score, Importance: 0.92
```

[41]:
```
#Run the test

print("Confusion matrix: \n" + str(confusion_matrix(yt,prediction).T) +␣
↪"\nTP,FN\nFP,TN")
```

```
Confusion matrix:
[[570 284]
 [219 370]]
TP,FN
FP,TN
```

[68]:
```
confusion_matrix(yt,prediction).T[0,0]/(confusion_matrix(yt,prediction).
↪T[0,0]+confusion_matrix(yt,prediction).T[0,1])
```

[68]: 0.667447306791569

[78]:
```
precision = confusion_matrix(yt,prediction).T[0,0]/
↪(confusion_matrix(yt,prediction).T[0,0]+confusion_matrix(yt,prediction).
↪T[1,0])

recall = confusion_matrix(yt,prediction).T[0,0]/
↪(confusion_matrix(yt,prediction).T[0,0]+confusion_matrix(yt,prediction).
↪T[0,1])

print("Accuracy for All: " + str(accuracy_score(yt,prediction)) +
      "\nEquation for accuracy: (TP+TN)/All" +
     "\n\nPrecision, what % of tuples predicted positive were correct: " +
     str(precision) +
     "\nEquation for precision TP/(TP+FP)" +
     "\n\nRecall, what % of tuples were classified as positive: "+
     str(recall) +
     "\nEquation for recall: TP/(TP+FN)" +
     "\n\nF measure, measures balance between both precision and␣
↪recall(harmonic mean): " +
     str((2*precision*recall)/(precision + recall)) +
     "\nEquation for F measure: (2*precision*recall)/(precision+recall)")
```

```
Accuracy for All: 0.6514206514206514
Equation for accuracy: (TP+TN)/All

Precision, what % of tuples predicted positive were correct: 0.7224334600760456
Equation for precision TP/(TP+FP)

Recall, what % of tuples were classified as positive: 0.667447306791569
```

```
Equation for recall: TP/(TP+FN)

F measure, measures balance between both precision and recall(harmonic mean):
0.6938527084601339
Equation for F measure: (2*precision*recall)/(precision+recall)
```

1.1.5 The main model is accurate about 65% of the time. Precision shows us that positives are correct approx 72% of the time. Note the F measure, this will tell us which model is better overall (unweighted).

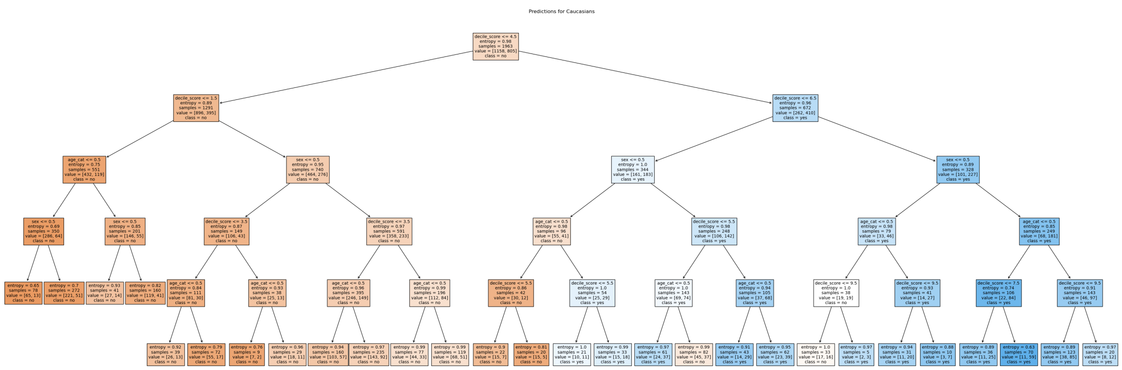
1.1.6 I ran the same data through the software Weka and got an accuracy of about 68%, very interesting to see how some models may classify more accurately than other models! Really goes to show why most contemporary models are proprietary.

1.1.7 Now, we repeat the same analysis for Caucasians and African Americans. Find Caucasian analysis below:

[79]:
```
#Preprocess some attributes to make scikit more digestible
df2['sex'] = preprocessing.LabelEncoder().fit(df2['sex']).transform(df2['sex'])
df2['age_cat'] = preprocessing.OneHotEncoder(sparse=False).
 ↪fit_transform(df2['age_cat'].values.reshape(-1, 1))
```

[80]:
```
#Make Trained Decision Tree for 'Caucasians' (df2)

#Grab attributes and class attribute
allAtr = df2[['sex', 'age_cat', 'decile_score']]
classAtr = df2['is_recid']

#Partition 20% of data to be tested, map to xtr-Training Attributes, xt-Test␣
 ↪Attributes, ytr-Class Training Attributes, yt-Class Test Attributes (used␣
 ↪for accuracy)
xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr,␣
 ↪test_size=0.2, random_state=42)

#instantiates tree object
AllTree = tree.DecisionTreeClassifier(criterion="entropy",max_depth=5)

#make the tree
AllTree.fit(xtr, ytr)

#predict based on test data
prediction = AllTree.predict(xt)

#plot data
plt.figure(figsize=(48, 16))
tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=['no',␣
 ↪'yes'], filled=True, impurity=True, precision=2, fontsize=10)
```

```python
plt.title('Predictions for Caucasians')
plt.savefig('decision_tree_caucasian.png', dpi=300) #very low res in out[]:,␣
 ↪see picture
plt.show()
```



[81]:
```python
importances = AllTree.feature_importances_ #get gain of each attribute␣
 ↪examined, put in list

print("Gain:")

# Print the attributes in the order of importance
for i in xtr.columns:
    print(f'Attribute: {i}, Importance: {importances[xtr.columns.get_loc(i)]:.
 ↪2f}')
```

```
Gain:
Attribute: sex, Importance: 0.07
Attribute: age_cat, Importance: 0.09
Attribute: decile_score, Importance: 0.84
```

[82]:
```python
#Run the test

print("Confusion matrix: \n" + str(confusion_matrix(yt,prediction).T) +␣
 ↪"\nTP,FN\nFP,TN")
```

```
Confusion matrix:
[[225 125]
 [ 46  95]]
TP,FN
FP,TN
```

[84]:
```python
precision = confusion_matrix(yt,prediction).T[0,0]/
 ↪(confusion_matrix(yt,prediction).T[0,0]+confusion_matrix(yt,prediction).
 ↪T[1,0])
```

```
recall = confusion_matrix(yt,prediction).T[0,0]/
↪(confusion_matrix(yt,prediction).T[0,0]+confusion_matrix(yt,prediction).
↪T[0,1])

print("Accuracy for Caucasians: " + str(accuracy_score(yt,prediction)) +
      "\nEquation for accuracy: (TP+TN)/All" +
     "\n\nPrecision, what % of tuples predicted positive were correct: " +
     str(precision) +
     "\nEquation for precision TP/(TP+FP)" +
     "\n\nRecall, what % of tuples were classified as positive: "+
     str(recall) +
     "\nEquation for recall: TP/(TP+FN)" +
     "\n\nF measure, measures balance between both precision and␣
↪recall(harmonic mean): " +
     str((2*precision*recall)/(precision + recall)) +
     "\nEquation for F measure: (2*precision*recall)/(precision+recall)")
```

```
Accuracy for Caucasians: 0.6517311608961304
Equation for accuracy: (TP+TN)/All

Precision, what % of tuples predicted positive were correct: 0.8302583025830258
Equation for precision TP/(TP+FP)

Recall, what % of tuples were classified as positive: 0.6428571428571429
Equation for recall: TP/(TP+FN)

F measure, measures balance between both precision and recall(harmonic mean):
0.7246376811594204
Equation for F measure: (2*precision*recall)/(precision+recall)
```

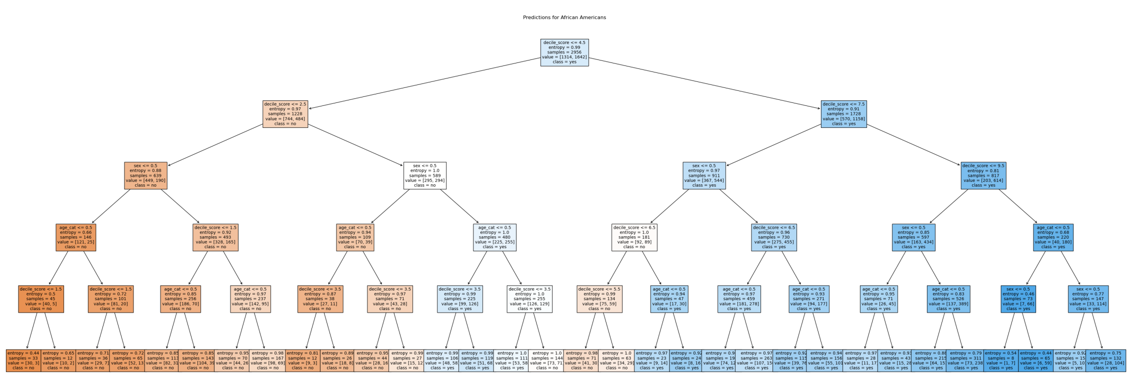
1.1.8 The model made for caucasians is slightly more accurate, correctly predicting positives 83% of the time. Our F measure is higher than the model for all races.

1.1.9 Find African Amercian analysis below:

[85]:
```
#Preprocess some attributes to make scikit more digestible
df3['sex'] = preprocessing.LabelEncoder().fit(df3['sex']).transform(df3['sex'])
df3['age_cat'] = preprocessing.OneHotEncoder(sparse=False).
↪fit_transform(df3['age_cat'].values.reshape(-1, 1))
```

[96]:
```
#Make Trained Decision Tree for 'African Americans' (df3)

#Grab attributes and class attribute
allAtr = df3[['sex', 'age_cat', 'decile_score']]
classAtr = df3['is_recid']
```

```
#Partition 20% of data to be tested, map to xtr-Training Attributes, xt-Test Attributes, ytr-Class Training Attributes, yt-Class Test Attributes (used for accuracy)
xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr, test_size=0.2, random_state=42)

#instantiates tree object
AllTree = tree.DecisionTreeClassifier(criterion="entropy",max_depth=5)
#make the tree
AllTree.fit(xtr, ytr)

#predict based on test data
prediction = AllTree.predict(xt)

#plot data
plt.figure(figsize=(48, 16))
tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=['no', 'yes'], filled=True, impurity=True, precision=2, fontsize=10)
plt.title('Predictions for African Americans')
plt.savefig('decision_tree_africanamerican.png', dpi=300) #very low res in out[]:, see picture
plt.show()
```



[87]:
```
importances = AllTree.feature_importances_ #get gain of each attribute examined, put in list

print("Gain:")

# Print the attributes in the order of importance
for i in xtr.columns:
    print(f'Attribute: {i}, Importance: {importances[xtr.columns.get_loc(i)]:.2f}')
```

```
Gain:
Attribute: sex, Importance: 0.10
Attribute: age_cat, Importance: 0.03
Attribute: decile_score, Importance: 0.86
```

[26]:
```
#Run the test

print("Confusion matrix: \n" + str(confusion_matrix(yt,prediction).T) +␣
↪"\nTP,FN\nFP,TN")
```

```
Confusion matrix:
[[155 191]
 [ 85 309]]
TP,FP
FN,TN
```

[88]:
```
precision = confusion_matrix(yt,prediction).T[0,0]/
↪(confusion_matrix(yt,prediction).T[0,0]+confusion_matrix(yt,prediction).
↪T[1,0])

recall = confusion_matrix(yt,prediction).T[0,0]/
↪(confusion_matrix(yt,prediction).T[0,0]+confusion_matrix(yt,prediction).
↪T[0,1])

print("Accuracy for African Americans: " + str(accuracy_score(yt,prediction)) +
      "\nEquation for accuracy: (TP+TN)/All" +
     "\n\nPrecision, what % of tuples predicted positive were correct: " +
     str(precision) +
     "\nEquation for precision TP/(TP+FP)" +
     "\n\nRecall, what % of tuples were classified as positive: "+
     str(recall) +
     "\nEquation for recall: TP/(TP+FN)" +
     "\n\nF measure, measures balance between both precision and␣
↪recall(harmonic mean): " +
     str((2*precision*recall)/(precision + recall)) +
     "\nEquation for F measure: (2*precision*recall)/(precision+recall)")
```

```
Accuracy for African Americans: 0.6270270270270271
Equation for accuracy: (TP+TN)/All

Precision, what % of tuples predicted positive were correct: 0.4393063583815029
Equation for precision TP/(TP+FP)

Recall, what % of tuples were classified as positive: 0.6495726495726496
Equation for recall: TP/(TP+FN)

F measure, measures balance between both precision and recall(harmonic mean):
0.5241379310344828
```

```
Equation for F measure: (2*precision*recall)/(precision+recall)
```

1.1.10 The model for African Americans is notably less accurate, only correcting classifying positives 43% of the time. The disparity in the False Positives goes to show why this prediction should not be the sole deciding factor in determining recidivism.

1.1.11 We can also check if we examine more attributes (for the decision tree) how it will affect our accuracy. Let's attempt this with our 'all' data, but this time we will also consider number of juvenile misdemeanors.

[89]:
```python
#read the data
df4 = pd.read_csv('compas_small.csv')

df4 = df4[['sex', 'age_cat', 'decile_score', 'juv_misd_count', 'is_recid']]
```

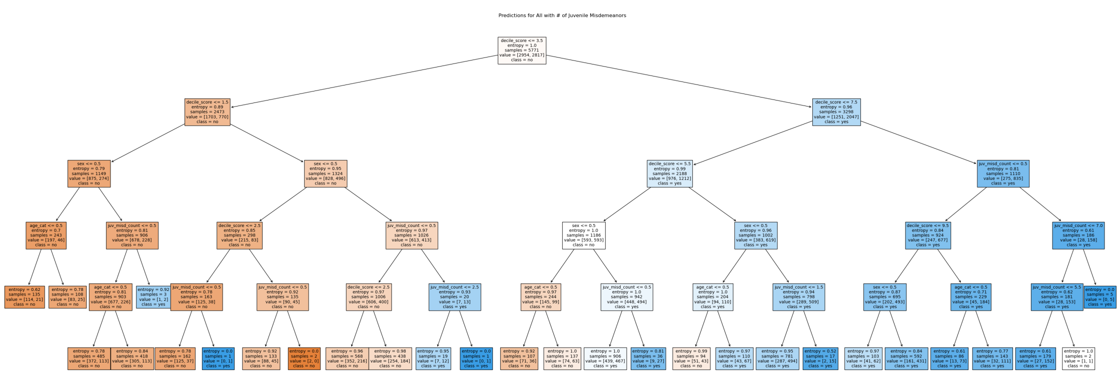
[90]:
```python
df4
```

[90]:
```
         sex          age_cat  decile_score  juv_misd_count is_recid
0       Male  Greater than 45             1               0       no
1       Male          25 - 45             3               0      yes
2       Male     Less than 25             4               0      yes
3       Male     Less than 25             8               1       no
4       Male          25 - 45             1               0       no
…        …                …             …               …      …
7209    Male     Less than 25             7               0       no
7210    Male     Less than 25             3               0       no
7211    Male  Greater than 45             1               0       no
7212  Female          25 - 45             2               0       no
7213  Female     Less than 25             4               0      yes

[7214 rows x 5 columns]
```

[91]:
```python
#Preprocess some attributes to make scikit more digestible
df4['sex'] = preprocessing.LabelEncoder().fit(df4['sex']).transform(df4['sex'])
df4['age_cat'] = preprocessing.OneHotEncoder(sparse=False).
 ↪fit_transform(df4['age_cat'].values.reshape(-1, 1))
```

[97]:
```python
#Make Trained Decision Tree for 'All with # of Juvenile Misdemeanors' (df4)

#Grab attributes and class attribute
allAtr = df4[['sex', 'age_cat', 'decile_score', 'juv_misd_count']]
classAtr = df4['is_recid']

#Partition 20% of data to be tested, map to xtr-Training Attributes, xt-Test␣
 ↪Attributes, ytr-Class Training Attributes, yt-Class Test Attributes (used␣
 ↪for accuracy)
```

```python
xtr, xt, ytr, yt = model_selection.train_test_split(allAtr, classAtr, test_size=0.2, random_state=42)

#instantiates tree object
AllTree = tree.DecisionTreeClassifier(criterion="entropy",max_depth=5)

#make the tree
AllTree.fit(xtr, ytr)

#predict based on test data
prediction = AllTree.predict(xt)

#plot data
plt.figure(figsize=(48, 16))
tree.plot_tree(AllTree, feature_names=allAtr.columns, class_names=['no', 'yes'], filled=True, impurity=True, precision=2, fontsize=10)
plt.title('Predictions for All with # of Juvenile Misdemeanors')
plt.savefig('decision_tree_juvc.png', dpi=300) #very low res in out[]:, see picture
plt.show()
```



[98]:
```python
importances = AllTree.feature_importances_ #get gain of each attribute examined, put in list

# Print the attributes in the order of importance
for i in xtr.columns:
    print(f'Attribute: {i}, Importance: {importances[xtr.columns.get_loc(i)]:.2f}')
```

```
Attribute: sex, Importance: 0.05
Attribute: age_cat, Importance: 0.02
Attribute: decile_score, Importance: 0.89
Attribute: juv_misd_count, Importance: 0.05
```

[99]:
```python
precision = confusion_matrix(yt,prediction).T[0,0]/
 ↪(confusion_matrix(yt,prediction).T[0,0]+confusion_matrix(yt,prediction).
 ↪T[1,0])

recall = confusion_matrix(yt,prediction).T[0,0]/
 ↪(confusion_matrix(yt,prediction).T[0,0]+confusion_matrix(yt,prediction).
 ↪T[0,1])

print("Accuracy for all (w/ juv midemeanors): " +␣
 ↪str(accuracy_score(yt,prediction)) +
       "\nEquation for accuracy: (TP+TN)/All" +
      "\n\nPrecision, what % of tuples predicted positive were correct: " +
      str(precision) +
      "\nEquation for precision TP/(TP+FP)" +
      "\n\nRecall, what % of tuples were classified as positive: "+
      str(recall) +
      "\nEquation for recall: TP/(TP+FN)" +
      "\n\nF measure, measures balance between both precision and␣
 ↪recall(harmonic mean): " +
      str((2*precision*recall)/(precision + recall)) +
      "\nEquation for F measure: (2*precision*recall)/(precision+recall)")
```

```
Accuracy for all (w/ juv midemeanors): 0.6340956340956341
Equation for accuracy: (TP+TN)/All

Precision, what % of tuples predicted positive were correct: 0.6185044359949303
Equation for precision TP/(TP+FP)

Recall, what % of tuples were classified as positive: 0.6825174825174826
Equation for recall: TP/(TP+FN)

F measure, measures balance between both precision and recall(harmonic mean):
0.6489361702127661
Equation for F measure: (2*precision*recall)/(precision+recall)
```

1.1.12 After we added in the juvenile misdemeanor count, the model decreased in accuracy. This is why attribute selection is vital to ensure there are no redundancies examined in our data. A good way to avoid redundancy in our attributes is to check correlation. We can see below that, although weak, juvenile misdemeanor count is positively correlated with decile score. It's the most notable correlation as compared to the other attributes.

[103]:
```python
corref1 = py.corrcoef(df4['decile_score'].values,df4['juv_misd_count'].
 ↪values)[0, 1]
corref2 = py.corrcoef(df4['age_cat'].values,df4['juv_misd_count'].values)[0, 1]
corref3 = py.corrcoef(df4['sex'].values,df4['juv_misd_count'].values)[0, 1]
```

```
print("Pearson Correlation of Decile Score and Misdeameanor count: " +
      str(corref1) +
     "\n\nPearson Correlation of Age Category and Misdeameanor count: " +
      str(corref2) +
     "\n\nPearson Correlation of Sex and Misdeameanor count: " +
      str(corref3))
```

```
Pearson Correlation of Decile Score and Misdeameanor count: 0.21592745594052032

Pearson Correlation of Age Category and Misdeameanor count: 0.023859526429586123

Pearson Correlation of Sex and Misdeameanor count: 0.04763666735215307
```