In [1]:
```
import pandas as pd
import statsmodels.formula.api as smf
import numpy as np
import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
from sklearn.model_selection import GridSearchCV
from sklearn.tree import *
```

In [2]:
```
train = pd.read_csv(r"yelp_train.csv")
test = pd.read_csv(r"yelp_test.csv")

Xtest = test.drop("stars", axis = 1)
Ytest = test["stars"]
Xtrain = train.drop("stars", axis = 1)
Ytrain = train["stars"]
```

In [3]:
```
### Regression model using all available independent variables.
model = smf.ols('stars ~ review_count + C(GoodForKids, Treatment(reference="(Missing)")) + C(Alco
                + C(BusinessAcceptsCreditCards, Treatment(reference="(Missing)")) + C(WiFi, Treat
                + C(BikeParking, Treatment(reference="(Missing)")) + C(ByAppointmentOnly, Treatme
                + C(WheelechairAccessible, Treatment(reference="(Missing)")) + C(OutdoorSeating,
                + C(RestaurantsReservations, Treatment(reference="(Missing)")) + C(DogsAllowed, T
                + C(Caters, Treatment(reference="(Missing)"))', data = train).fit()
```

In [4]:
```
#ols predictions, storing for later.
r2d2 = model.predict(Xtest)
```

In [5]:
```
#dummy encode the X training matrix.
Dummy_train = pd.get_dummies(Xtrain, columns = [col for col in Xtrain.columns if col != "review_
```

In [6]:
```
#grid of candidates, 10 folds for each of 675 different models.

grid_values = {'ccp_alpha': np.linspace(0,0.01, 20),
               'min_samples_leaf': [4,5,6],
               'min_samples_split': [20,25,30],
               'max_depth': [ 5,10,15,20],
               }

# Find and fit the good model
dtr = DecisionTreeRegressor()
dtc_cv_acc = GridSearchCV(dtr, param_grid = grid_values, scoring = 'neg_mean_squared_error', cv=
dtc_cv_acc.fit(Dummy_train, Ytrain)
```

Out[6]:
```
▸        GridSearchCV
▸ estimator: DecisionTreeRegressor
      ▸ DecisionTreeRegressor
```

In [7]:
```
# checked the best parameters for the tree manually. did not do this for the next tree.
print(dtc_cv_acc.best_params_)
```

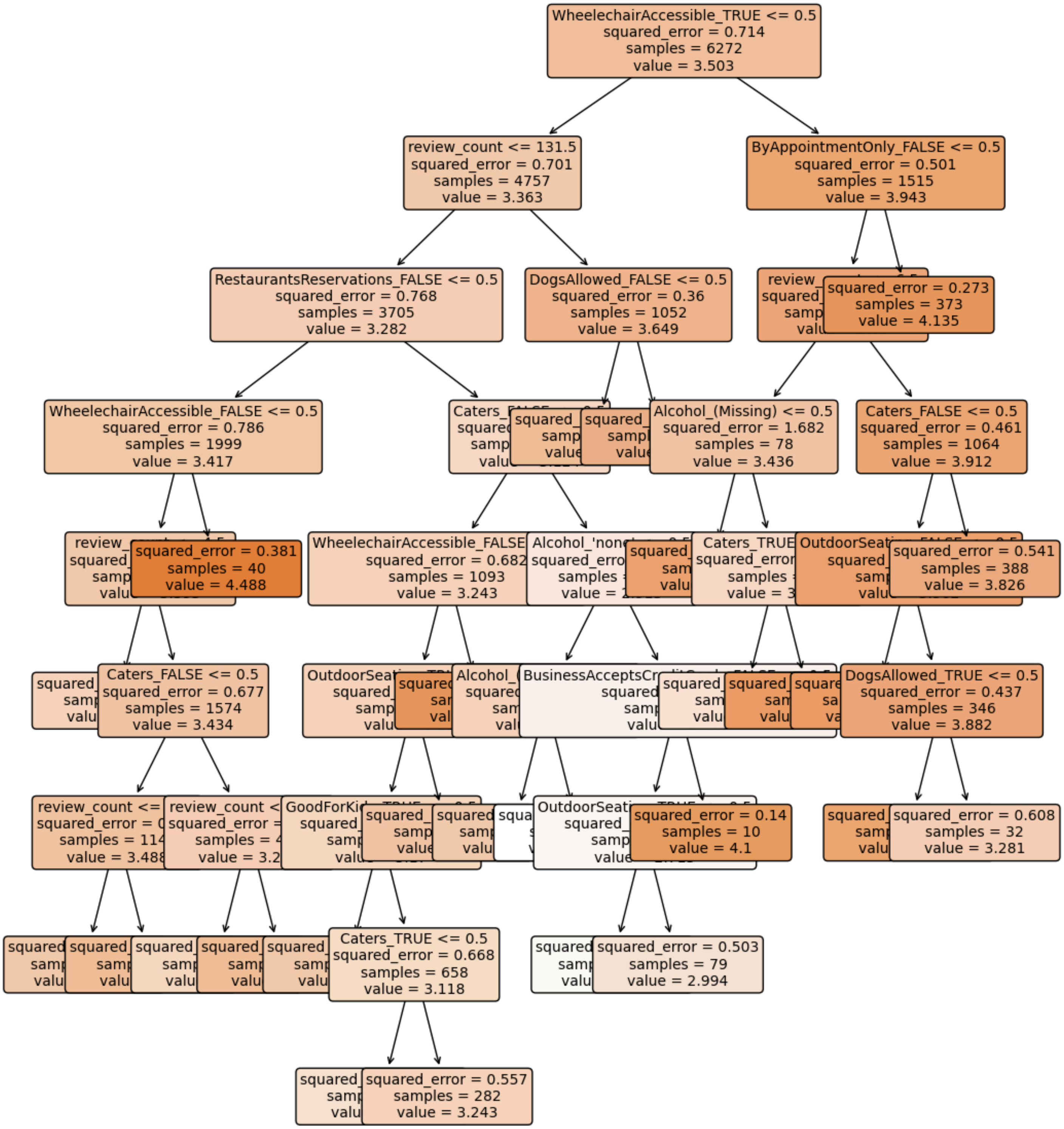
```
{'ccp_alpha': 0.0010526315789473684, 'max_depth': 15, 'min_samples_leaf': 6, 'min_samples_split':
30}
```

In [8]:
```
#setting my decision tree parameters to the candidates found by grid search.
dtrBest =  DecisionTreeRegressor(min_samples_leaf=6,
                                  ccp_alpha=0.0010526315789473684,
                                 max_depth = 15,
                                 min_samples_split = 25)
tree = dtrBest.fit(Dummy_train,Ytrain)

#Tree diagram plot

print('Node count =', dtrBest.tree_.node_count)
plt.figure(figsize=(12,15))
plot_tree(dtrBest,
          feature_names = Dummy_train.columns,
          filled=True,
          impurity=True,
          rounded=True,
          fontsize = 10
          )
plt.show()
```

```
Node count = 51
```



```
In [9]: #encoding my Test set, and predicting target variables.
        DummyTest = pd.get_dummies(Xtest, columns = [col for col in Xtest.columns if col != "review_coun

        Ypred = tree.predict(DummyTest)
```

In [10]:

```python
#functions for OSR^2 and MAE
def OSR2(Ypreds, trueTest):
    RSS = np.sum((trueTest - Ypreds)**2 )
    TSS = np.sum((trueTest - np.mean(trueTest))**2)
    return 1 - (RSS/TSS)
def MAE(Ypreds, trueTest):
    return np.sum(np.abs(Ypreds - trueTest))/len(Ypreds)
```

In [11]:

```python
#the following cells are outputs of OSR2 and MAE of decision tree and regression predictions.

MAE(Ypred, Ytest)
```

Out[11]:

```
0.6252132561993073
```

In [12]:

```python
OSR2(Ypred, Ytest)
```

Out[12]:

```
0.15099276886806745
```

In [13]:

```python
regrPred = model.predict(Xtest)
```

In [14]:

```python
MAE(regrPred, Ytest)
```

Out[14]:

```
0.630129081526346
```

In [15]:

```python
OSR2(regrPred, Ytest)
```

Out[15]:

```
0.1625081247047695
```

In [16]:

```python
#make copies so we can establish a new target variable,boolean of above/ below 4 star rating.
trainGT4, testGT4 = train.copy(), test.copy()
```

In [17]:

```python
#make bool columns if a row has a rating of >= 4
trainGT4["fourOrAbove"] = (trainGT4["stars"] >= 4).astype(int)
testGT4["fourOrAbove"] = (testGT4["stars"] >=4).astype(int)
```

In [18]:

```python
#2d ii:
# Code for converting predictions of at least four/ below 4 into boolean outcomes

def regr4orAbove(x):
    x = np.asarray(x)
    return [1 if i >= 4 else 0 for i in x ]

def logitYhat(x):
    x = np.asarray(x)
    return [1 if i>= 0.35272055398 else 0 for i in x]
```

In [19]:

```python
#logistic regression model for Four or above rating classifier

logistic4OA = smf.logit('fourOrAbove ~ review_count + C(GoodForKids, Treatment(reference="(Missin
                + C(BusinessAcceptsCreditCards, Treatment(reference="(Missing)")) + C(WiFi, Treat
                + C(BikeParking, Treatment(reference="(Missing)")) + C(ByAppointmentOnly, Treatme
                + C(WheelechairAccessible, Treatment(reference="(Missing)")) + C(OutdoorSeating, 
                + C(RestaurantsReservations, Treatment(reference="(Missing)")) + C(DogsAllowed, T
                + C(Caters, Treatment(reference="(Missing)"))', data = trainGT4).fit()
```

```
Optimization terminated successfully.
         Current function value: 0.607102
         Iterations 6
```

In [20]:

```python
#predictions of the logistic regression function.
logitPred = logistic4OA.predict(Xtest)
logitPred = 1/(1+ np.e**logitPred)
logitPred
```

Out[20]:

```
0       0.414209
1       0.317227
2       0.308660
3       0.354872
4       0.422787
          ...
2683    0.396007
2684    0.437190
2685    0.319350
2686    0.329078
2687    0.376931
Length: 2688, dtype: float64
```

In [21]:

```python
#dataframe cleaning, etc.
trainGT4X = trainGT4.drop(columns = ["stars","fourOrAbove"])
testGT4X = testGT4.drop(columns = ["stars","fourOrAbove"])
testGT4X = pd.get_dummies(testGT4X, columns = [col for col in Xtest.columns if col != "review_co
testGT4Y = testGT4["fourOrAbove"]
trainGT4Y = trainGT4["fourOrAbove"]
trainGT4X = pd.get_dummies(trainGT4X, columns = [col for col in Xtest.columns if col!= "review_c
```

In [22]:

```python
#Cross validation and model fitting for classification tree
grid_values = {'ccp_alpha': np.linspace(0,0.01,26),
               'min_samples_leaf': [3,4,5,6,7],
               'min_samples_split': [15,20,25],
               'max_depth': np.arange(12,20),
               'class_weight' : ["balanced"],
               }


dtc = DecisionTreeClassifier()
dtcCV = GridSearchCV(dtc, param_grid = grid_values, scoring = 'accuracy', cv=4, verbose=1,n_jobs
dtcCV.fit(trainGT4X, trainGT4Y)
```

```
Fitting 4 folds for each of 3120 candidates, totalling 12480 fits
```

Out[22]:

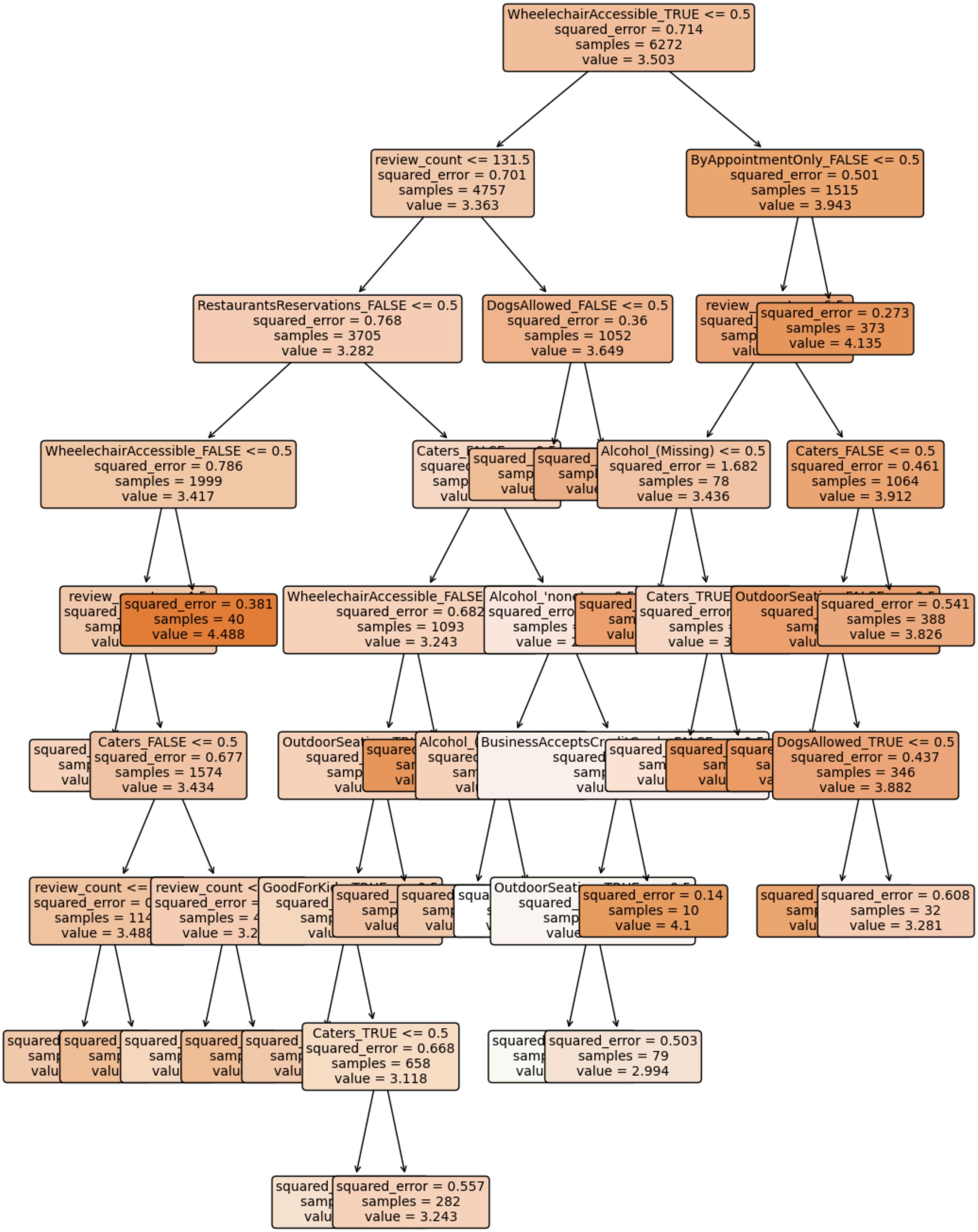
```
▸            GridSearchCV
▸ estimator: DecisionTreeClassifier
      ▸ DecisionTreeClassifier
```

In [23]:

```python
#tree diagram for classifier
print(dtcCV.best_params_)
print('Node count =', dtcCV.best_estimator_.tree_.node_count)
plt.figure(figsize=(12,18))
plot_tree(dtrBest,
          feature_names = trainGT4X.columns,
          filled=True,
          impurity=True,
          rounded=True,
          fontsize = 10
```

```
           )
plt.show()
```

```
{'ccp_alpha': 0.0008, 'class_weight': 'balanced', 'max_depth': 12, 'min_samples_leaf': 3, 'min_sa
mples_split': 15}
Node count = 45
```



In [24]:
```
#grid search's best parameters for my model given the candidates in the cross validation.
dtcCV.best_params_
```

```
Out[24]: {'ccp_alpha': 0.0008,
          'class_weight': 'balanced',
          'max_depth': 12,
          'min_samples_leaf': 3,
          'min_samples_split': 15}
```

```
In [25]: #predict the Above 4 star rating for each estabishment using the classfication tree.
         Ypred2 = dtcCV.best_estimator_.predict(testGT4X)
         Ypred2
```

```
Out[25]: array([0, 1, 1, ..., 1, 1, 1])
```

```
In [26]: #baseline naive classifier, predicts mode of the outcomes.
         y_baseline = testGT4Y.mode()
         modePred = pd.Series([y_baseline[0] for _ in range(len(testGT4Y))])
```

```
In [27]: #function that returns a 3-list with accuracy, TPR, and FPR given a true y values array and a pre
         def AccTFR(y,yhat):
             y = np.asarray(y)
             yhat = np.asarray(yhat)
             return [np.mean(y == yhat), np.sum((y == 1) & (yhat == 1)) / (np.sum((y == 1) & (yhat == 1))
```

```
In [28]: # constructing the dataframe "table" for each of the five models, with accuracy, tpr, and fpr be

         df = pd.DataFrame()

         df["baseline"] = AccTFR(testGT4Y,modePred)
         df["lin. regression Threshold"] = AccTFR(testGT4Y, regr4orAbove(r2d2))
         df["regressor Dec. Tree"] = AccTFR(testGT4Y, regr4orAbove(Ypred))
         df["log. Regression"] = AccTFR(testGT4Y, logitYhat(logitPred))
         df["Classification Tree"] = AccTFR(testGT4Y, Ypred2)

         df.index = ["Accuracy:", "TPR:","FPR:"]
         df
```

Out[28]:

	baseline	lin. regression Threshold	regressor Dec. Tree	log. Regression	Classification Tree
Accuracy:	0.551339	0.616071	0.616443	0.338542	0.643229
TPR:	0.000000	0.203980	0.202322	0.628524	0.538143
FPR:	0.000000	0.048583	0.046559	0.897436	0.271255

```
In [29]: #question 2e- feature importance
         imp = dtcCV.best_estimator_.feature_importances_
         features = Dummy_train.columns
         importance_df = pd.DataFrame({
             'Feature': features,
             'Importance': imp
         })

         # Sort the DataFrame by importance
         importance_df = importance_df.sort_values(by='Importance', ascending=False)

         importance_df
```

Out[29]:

	Feature	Importance
16	WheelechairAccessible_TRUE	0.488146
0	review_count	0.195819
15	WheelechairAccessible_FALSE	0.068601
19	RestaurantsReservations_FALSE	0.065059
23	Caters_FALSE	0.052794
3	Alcohol_'full_bar'	0.024468
13	ByAppointmentOnly_FALSE	0.022670
18	OutdoorSeating_TRUE	0.022189
6	BusinessAcceptsCreditCards_FALSE	0.021529
21	DogsAllowed_FALSE	0.014867
2	GoodForKids_TRUE	0.013512
4	Alcohol_'none'	0.010345
22	DogsAllowed_TRUE	0.000000
20	RestaurantsReservations_TRUE	0.000000
17	OutdoorSeating_FALSE	0.000000
12	BikeParking_TRUE	0.000000
14	ByAppointmentOnly_TRUE	0.000000
1	GoodForKids_FALSE	0.000000
11	BikeParking_FALSE	0.000000
10	WiFi_(Missing)	0.000000
9	WiFi_'paid'	0.000000
8	WiFi_'no'	0.000000
7	BusinessAcceptsCreditCards_TRUE	0.000000
5	Alcohol_(Missing)	0.000000
24	Caters_TRUE	0.000000