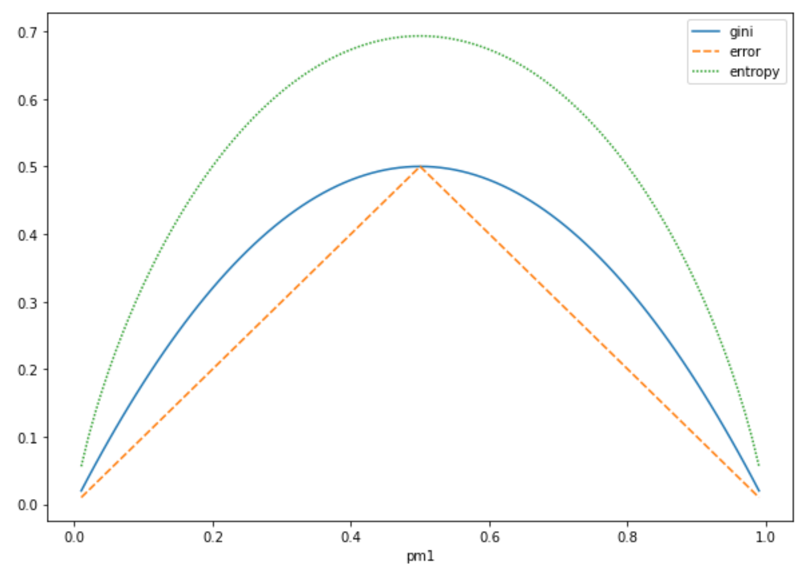
Youssef Mahmoud 905854027

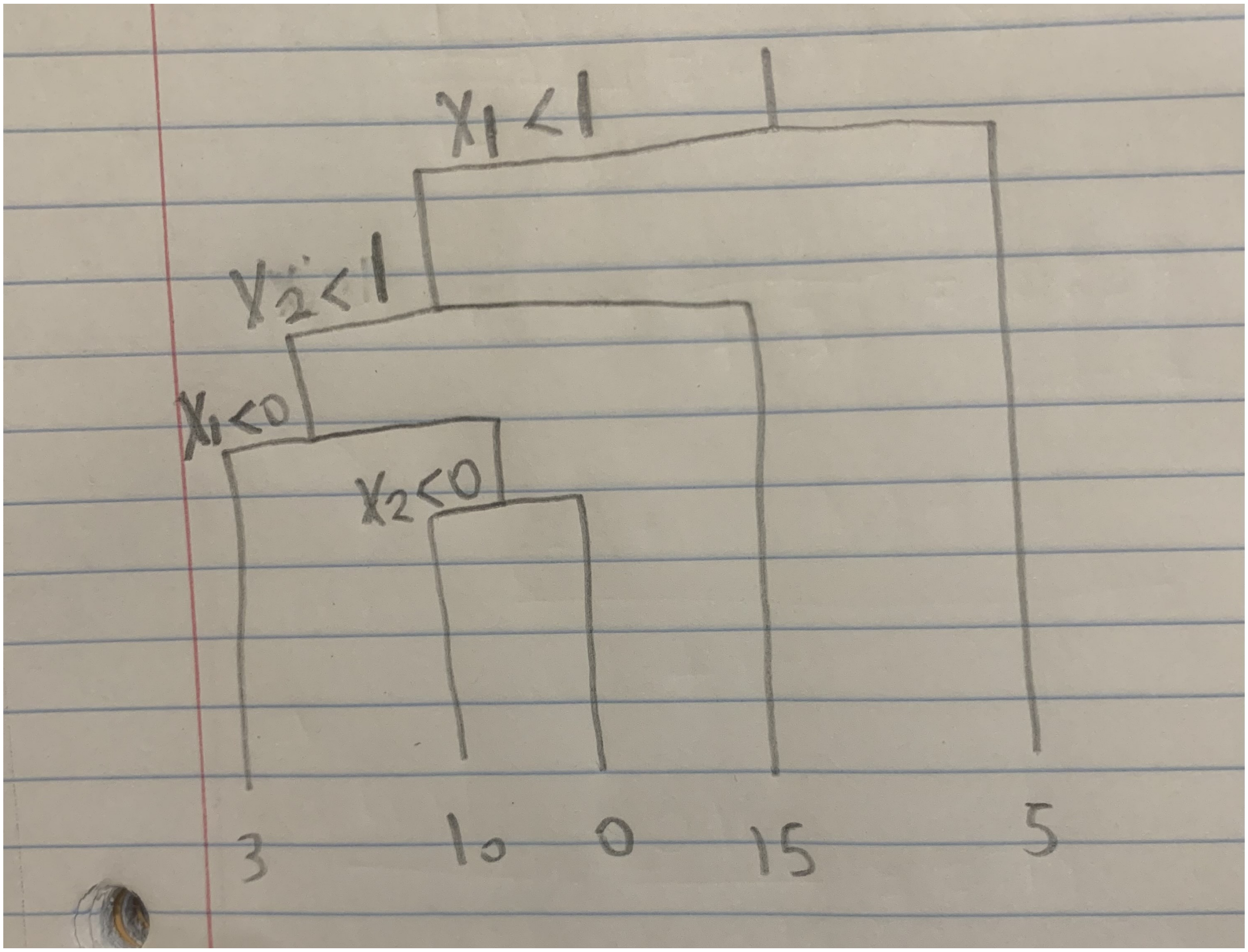
In [140]:
```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt
import graphviz
from sklearn import tree
from sklearn.model_selection import cross_val_score
import sklearn
from sklearn.metrics import accuracy_score, roc_auc_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, export_graphviz, plot_tree
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import confusion_matrix, mean_squared_error
import seaborn as sns
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV,RandomizedSearchCV
from sklearn.ensemble import AdaBoostRegressor
from sklearn.tree import DecisionTreeRegressor
import xgboost as XGB
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
%matplotlib inline
```

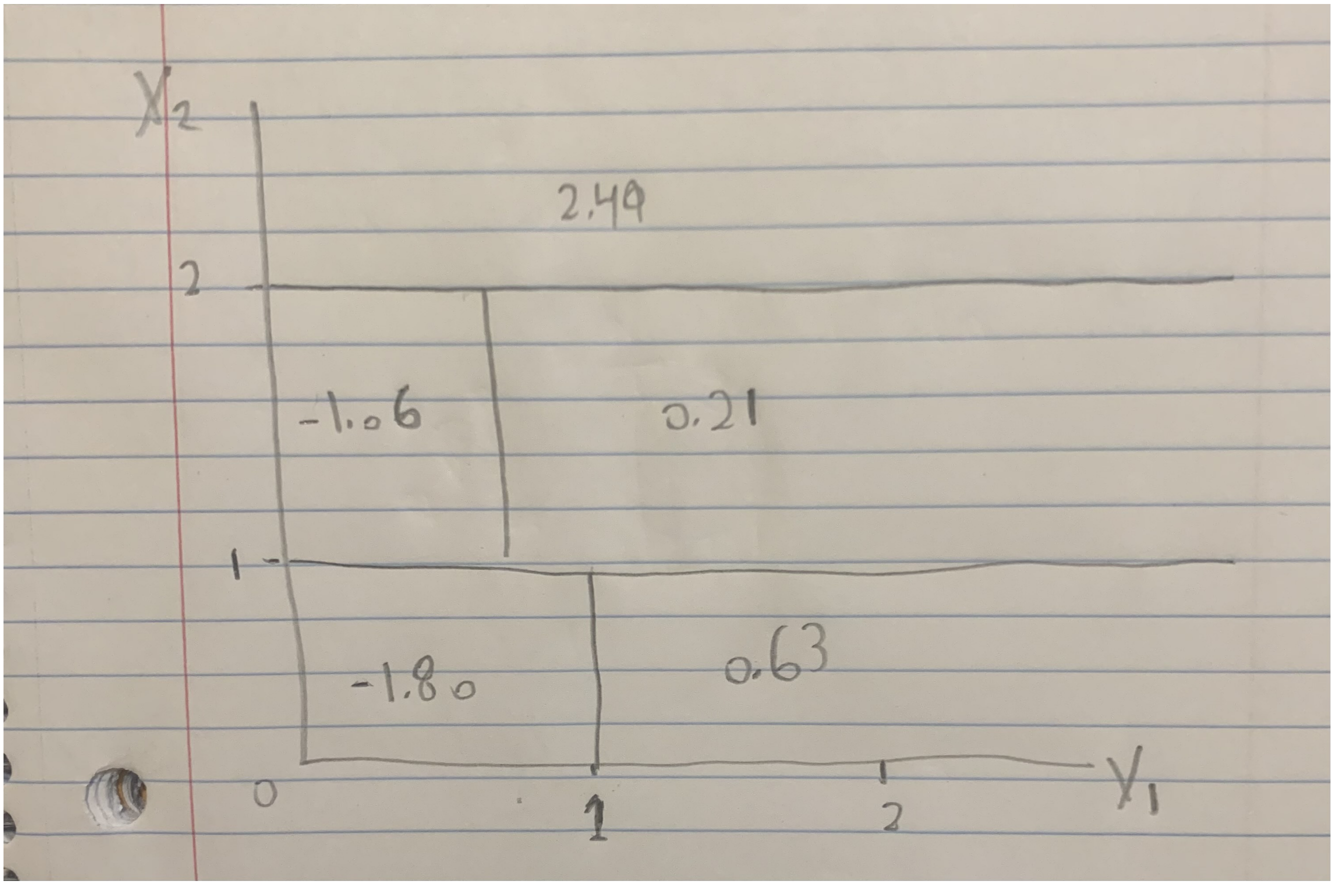
8.4.3

In [2]:
```
pm1 = np.arange(0.01, 1, 0.01)
pm2 = 1 - pm1
gini = (pm1 * (1 - pm1)) + (pm2 * (1 - pm2))


err=[]
for p in pm1:
    if 0.5 <= p:
        err += [1 - p]
    if p < 0.5:
        err += [p]


entropy = -((pm1 * np.log(pm1)) + (pm2 * np.log(pm2)))




df = pd.DataFrame(np.stack([pm1, gini, err, entropy], axis=1),
                  columns=['pm1', 'gini', 'error', 'entropy']).set_index('pm1')
plt.figure(figsize=(10, 7))
sns.lineplot(data=df);
```



8.4.4





8.4.5

In [3]:
```
X_boot = np.array([0.1,0.15,0.2,0.2,0.55,0.6,0.6,0.65,0.7,0.75])

def majority_clf(votes):
    pro_votes = (0.5 < votes).sum()
    majority_is_pro = (len(votes)/2) < pro_votes
    return majority_is_pro, pro_votes

def avg_clf(votes):
    avg = np.mean(votes)
    return (0.5 < avg, avg)

print('Is red (by Majority): {}, votes_red={}'.format(*majority_clf(X_boot)))
print('Is red (by Average) : {}, avg={}'.format(*avg_clf(X_boot)))
```

```
Is red (by Majority): True, votes_red=6
Is red (by Average) : False, avg=0.45
```

8.3.Boston

In [4]:
```
df1 = pd.read_csv("desktop/housing.csv")
```

In [5]:
```
df1
```

Out[5]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
501	0.06263	0.0	11.93	0	0.573	6.593	69.1	2.4786	1	273	21.0	391.99	9.67	22.4
502	0.04527	0.0	11.93	0	0.573	6.120	76.7	2.2875	1	273	21.0	396.90	9.08	20.6
503	0.06076	0.0	11.93	0	0.573	6.976	91.0	2.1675	1	273	21.0	396.90	5.64	23.9
504	0.10959	0.0	11.93	0	0.573	6.794	89.3	2.3889	1	273	21.0	393.45	6.48	22.0
505	0.04741	0.0	11.93	0	0.573	6.030	80.8	2.5050	1	273	21.0	396.90	7.88	11.9

506 rows × 14 columns

In [6]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   CRIM     506 non-null    float64
 1   ZN       506 non-null    float64
 2   INDUS    506 non-null    float64
 3   CHAS     506 non-null    int64
 4   NOX      506 non-null    float64
 5   RM       506 non-null    float64
 6   AGE      506 non-null    float64
 7   DIS      506 non-null    float64
 8   RAD      506 non-null    int64
 9   TAX      506 non-null    int64
 10  PTRATIO  506 non-null    float64
 11  B        506 non-null    float64
 12  LSTAT    506 non-null    float64
 13  MEDV     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

In [7]:
```
df1_train, df1_test = train_test_split(
  df1,
  train_size = 0.5,
  random_state = 425, # seed
  )
df1_test.shape
```

Out[7]: (253, 14)

In [8]:
```
features = ['CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', 'DIS', 'RAD','TAX','PTRATIO','B','LSTAT']
X_train = df1_train[features]
Y_train = np.log(df1_train.MEDV)

X_test = df1_test[features]
Y_test = np.log(df1_test.MEDV)
```

Regression Tree

In [9]:
```
regtree_mod = DecisionTreeRegressor(random_state = 425)
```

In [10]:
```
pipe = Pipeline(steps = [
  ("model", regtree_mod)
  ])
pipe
```

Out[10]: Pipeline(steps=[('model', DecisionTreeRegressor(random_state=425))])

In [11]:
```
ccp_alpha_grid = np.linspace(start = 0.0, stop = 0.1, num = 100)
tuned_parameters = {
  "model__ccp_alpha": ccp_alpha_grid
  }
tuned_parameters
```

Out[11]:
```
{'model__ccp_alpha': array([0.        , 0.0010101 , 0.0020202 , 0.0030303 , 0.0040404 ,
        0.00505051, 0.00606061, 0.00707071, 0.00808081, 0.00909091,
        0.01010101, 0.01111111, 0.01212121, 0.01313131, 0.01414141,
        0.01515152, 0.01616162, 0.01717172, 0.01818182, 0.01919192,
        0.02020202, 0.02121212, 0.02222222, 0.02323232, 0.02424242,
        0.02525253, 0.02626263, 0.02727273, 0.02828283, 0.02929293,
        0.03030303, 0.03131313, 0.03232323, 0.03333333, 0.03434343,
        0.03535354, 0.03636364, 0.03737374, 0.03838384, 0.03939394,
        0.04040404, 0.04141414, 0.04242424, 0.04343434, 0.04444444,
        0.04545455, 0.04646465, 0.04747475, 0.04848485, 0.04949495,
        0.05050505, 0.05151515, 0.05252525, 0.05353535, 0.05454545,
        0.05555556, 0.05656566, 0.05757576, 0.05858586, 0.05959596,
        0.06060606, 0.06161616, 0.06262626, 0.06363636, 0.06464646,
        0.06565657, 0.06666667, 0.06767677, 0.06868687, 0.06969697,
        0.07070707, 0.07171717, 0.07272727, 0.07373737, 0.07474747,
        0.07575758, 0.07676768, 0.07777778, 0.07878788, 0.07979798,
        0.08080808, 0.08181818, 0.08282828, 0.08383838, 0.08484848,
        0.08585859, 0.08686869, 0.08787879, 0.08888889, 0.08989899,
        0.09090909, 0.09191919, 0.09292929, 0.09393939, 0.09494949,
        0.0959596 , 0.0969697 , 0.0979798 , 0.0989899 , 0.1       ])}
```

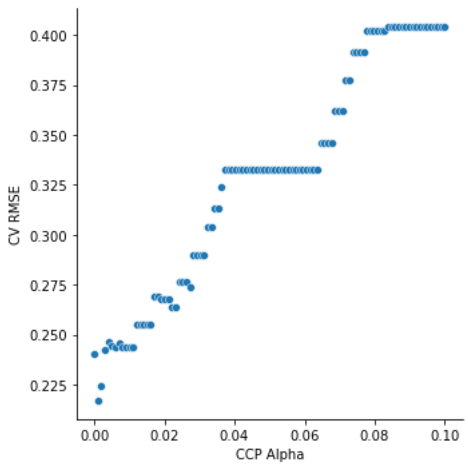
In [12]:
```
n_folds = 6
search = GridSearchCV(
  pipe,
  tuned_parameters,
  cv = n_folds,
  scoring = "neg_root_mean_squared_error",
  # Refit the best model on the whole data set
  refit = True
  )
```

In [13]:
```
search.fit(X_train, Y_train)
```

Out[13]:
```
GridSearchCV(cv=6,
             estimator=Pipeline(steps=[('model',
                                        DecisionTreeRegressor(random_state=425))]),
             param_grid={'model__ccp_alpha': array([0.        , 0.0010101 , 0.0020202 , 0.0030303 , 0.0040404 ,
       0.00505051, 0.00606061, 0.00707071, 0.00808081, 0.00909091,
       0.01010101, 0.01111111, 0.01212121, 0.01313131, 0.01414141,
       0.01515152, 0.01616162, 0.01717172, 0.01818182, 0.01919192,
       0.020202...
       0.07070707, 0.07171717, 0.07272727, 0.07373737, 0.07474747,
       0.07575758, 0.07676768, 0.07777778, 0.07878788, 0.07979798,
       0.08080808, 0.08181818, 0.08282828, 0.08383838, 0.08484848,
       0.08585859, 0.08686869, 0.08787879, 0.08888889, 0.08989899,
       0.09090909, 0.09191919, 0.09292929, 0.09393939, 0.09494949,
       0.0959596 , 0.0969697 , 0.0979798 , 0.0989899 , 0.1       ])},
             scoring='neg_root_mean_squared_error')
```

In [14]:
```
cv_res = pd.DataFrame({
  "ccp_alpha": np.array(search.cv_results_["param_model__ccp_alpha"]),
  "rmse": -search.cv_results_["mean_test_score"]
  })

plt.figure()
sns.relplot(
  data = cv_res,
  x = "ccp_alpha",
  y = "rmse"
  ).set(
    xlabel = "CCP Alpha",
    ylabel = "CV RMSE"
);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



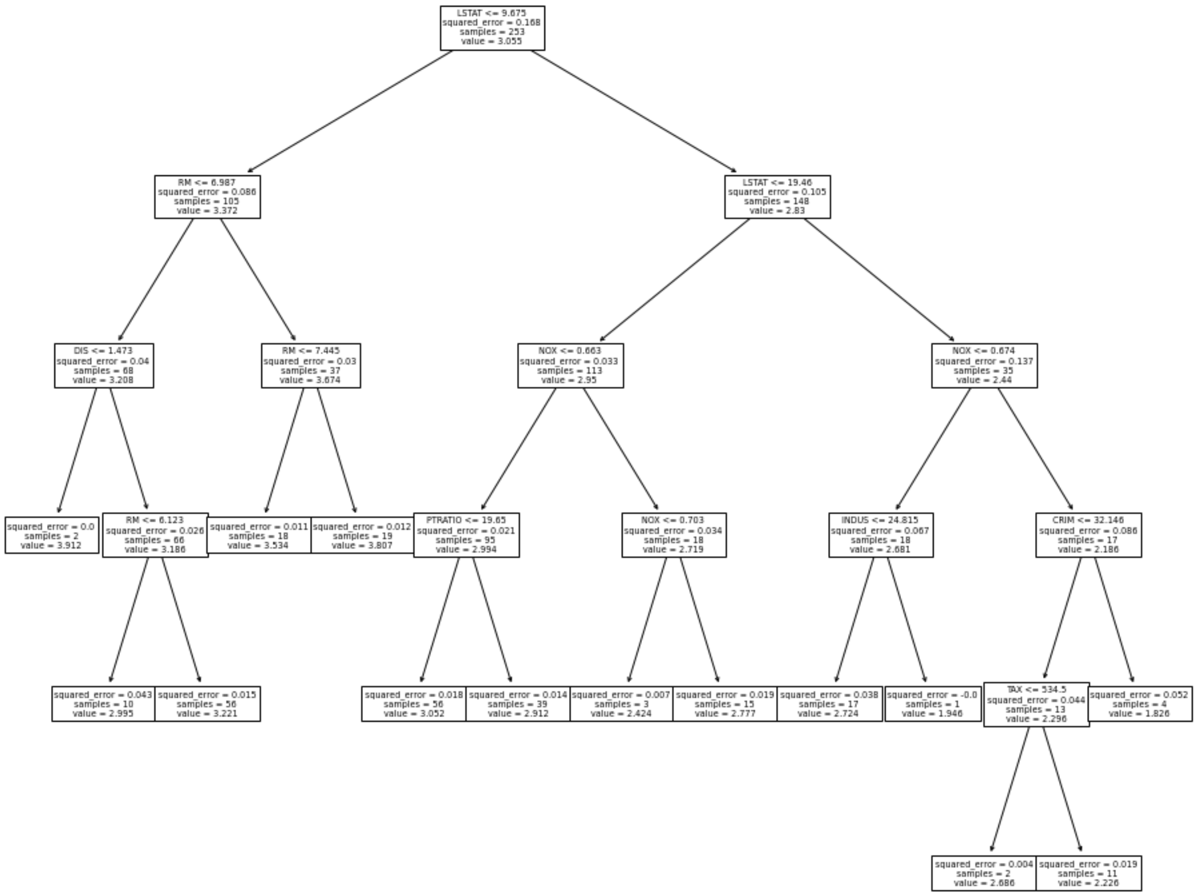
In [15]:
```
-search.best_score_
```

Out[15]:
```
0.2170777866126541
```

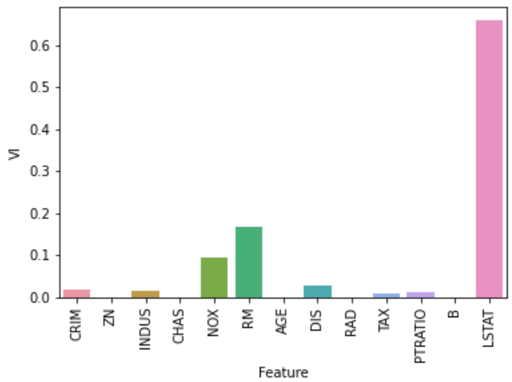
In [16]:
```
search.best_estimator_
```

Out[16]:
```
Pipeline(steps=[('model',
                 DecisionTreeRegressor(ccp_alpha=0.0010101010101010101,
                                       random_state=425))])
```

In [17]:
```
plt.figure(figsize=(16,14))
sklearn.tree.plot_tree(
  search.best_estimator_['model'],
  feature_names = features
  );
plt.show()
```



In [18]:
```
vi_df = pd.DataFrame({
  "feature": features,
  "vi": search.best_estimator_['model'].feature_importances_
  })

plt.figure()
sns.barplot(
  data = vi_df,
  x = "feature",
  y = "vi"
  ).set(
    xlabel = "Feature",
    ylabel = "VI"
);
plt.xticks(rotation = 90);
plt.show()
```



In [19]:
```
mean_squared_error(
  Y_test,
  search.best_estimator_.predict(X_test),
  squared = False
  )
```

Out[19]: 0.21790720977320982

Random Forest

In [20]:
```
rf_mod =  RandomForestRegressor(
  # Number of trees
  n_estimators = 100,
  criterion = 'squared_error',
  # Number of features to use in each split
  max_features = 'sqrt',
  oob_score = True,
  random_state = 425
  )
```

In [21]:
```
pipe = Pipeline(steps = [
  ("model", rf_mod)
  ])
pipe
```

Out[21]:
```
Pipeline(steps=[('model',
                 RandomForestRegressor(max_features='sqrt', oob_score=True,
                                       random_state=425))])
```

In [22]:
```
B_grid = [50, 100, 150, 200, 250, 300]
m_grid = ['sqrt', 'log2', 1.0] # max_features = 1.0 uses all features
tuned_parameters = {
  "model__n_estimators": B_grid,
  "model__max_features": m_grid
  }
tuned_parameters
```

Out[22]:
```
{'model__n_estimators': [50, 100, 150, 200, 250, 300],
 'model__max_features': ['sqrt', 'log2', 1.0]}
```

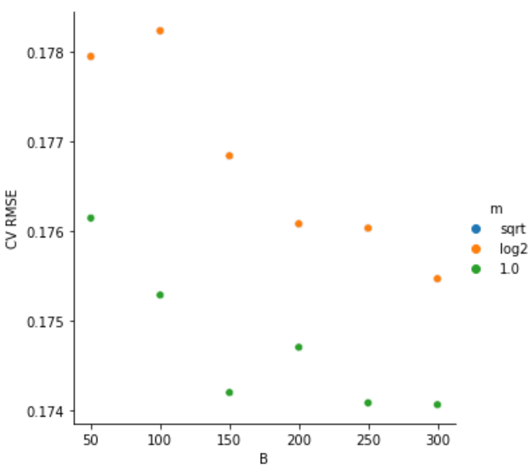
In [23]:
```
# Set up CV
n_folds = 6
search = GridSearchCV(
  pipe,
  tuned_parameters,
  cv = n_folds,
  scoring = "neg_root_mean_squared_error",
  # Refit the best model on the whole data set
  refit = True
  )
```

In [24]:
```
search.fit(X_train,Y_train)
```

Out[24]:
```
GridSearchCV(cv=6,
             estimator=Pipeline(steps=[('model',
                                        RandomForestRegressor(max_features='sqrt',
                                                              oob_score=True,
                                                              random_state=425))]),
             param_grid={'model__max_features': ['sqrt', 'log2', 1.0],
                         'model__n_estimators': [50, 100, 150, 200, 250, 300]},
             scoring='neg_root_mean_squared_error')
```

In [25]:
```
cv_res = pd.DataFrame({
  "B": np.array(search.cv_results_["param_model__n_estimators"]),
  "rmse": -search.cv_results_["mean_test_score"],
  "m": search.cv_results_["param_model__max_features"]
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "B",
  y = "rmse",
  hue = "m",
  ).set(
    xlabel = "B",
    ylabel = "CV RMSE"
);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [26]:
```
-search.best_score_
```

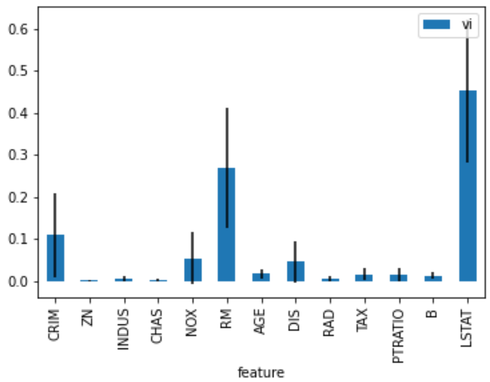
Out[26]:
```
0.17406330114074656
```

In [27]:
```
search.best_estimator_
```

Out[27]:
```
Pipeline(steps=[('model',
                 RandomForestRegressor(max_features=1.0, n_estimators=300,
                                       oob_score=True, random_state=425))])
```

In [28]:
```
vi_df = pd.DataFrame({
  "feature": features,
  "vi": search.best_estimator_['model'].feature_importances_,
  "vi_std": np.std([tree.feature_importances_ for tree in search.best_estimator_['model'].estimators_], axis = 0)
  })

plt.figure()
vi_df.plot.bar(x = "feature", y = "vi", yerr = "vi_std")
plt.xticks(rotation = 90);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [29]:
```
mean_squared_error(
  Y_test,
  search.best_estimator_.predict(X_test),
  squared = False
  )
```

Out[29]:
```
0.153092179332972
```

Boosting

In [30]:
```
bst_mod =  AdaBoostRegressor(
  # Default base estimator is DecisionTreeRegressor with max_depth = 3
  base_estimator = DecisionTreeRegressor(max_depth = 3),
  # Number of trees (to be tuned)
  n_estimators = 50,
  # Learning rate (to be tuned)
  learning_rate = 1.0,
  random_state = 425
  )
```

In [31]:
```
pipe = Pipeline(steps = [
  ("model", bst_mod)
  ])
pipe
```

Out[31]:
```
Pipeline(steps=[('model',
                 AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3),
                                   random_state=425))])
```

In [32]:

```
d_grid = [
  DecisionTreeRegressor(max_depth = 1),
  DecisionTreeRegressor(max_depth = 2),
  DecisionTreeRegressor(max_depth = 3),
  DecisionTreeRegressor(max_depth = 4)
  ]
B_grid = [50, 100, 150, 200, 250, 300, 350, 400]
lambda_grid = [0.2, 0.4, 0.6, 0.8, 1.0]
tuned_parameters = {
  "model__base_estimator": d_grid,
  "model__n_estimators": B_grid,
  "model__learning_rate": lambda_grid
  }
tuned_parameters
```

Out[32]:

```
{'model__base_estimator': [DecisionTreeRegressor(max_depth=1),
  DecisionTreeRegressor(max_depth=2),
  DecisionTreeRegressor(max_depth=3),
  DecisionTreeRegressor(max_depth=4)],
 'model__n_estimators': [50, 100, 150, 200, 250, 300, 350, 400],
 'model__learning_rate': [0.2, 0.4, 0.6, 0.8, 1.0]}
```

In [33]:

```
n_folds = 6
search = GridSearchCV(
  pipe,
  tuned_parameters,
  cv = n_folds,
  scoring = "neg_root_mean_squared_error",
  # Refit the best model on the whole data set
  refit = True
  )
```

In [34]:

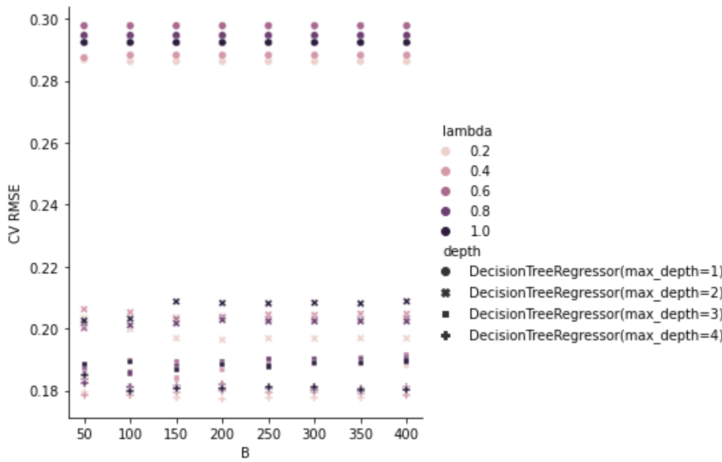
```
search.fit(X_train,Y_train)

```

Out[34]:

```
GridSearchCV(cv=6,
             estimator=Pipeline(steps=[('model',
                                        AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=3),
                                                          random_state=425))]),
             param_grid={'model__base_estimator': [DecisionTreeRegressor(max_depth=1),
                                                   DecisionTreeRegressor(max_depth=2),
                                                   DecisionTreeRegressor(max_depth=3),
                                                   DecisionTreeRegressor(max_depth=4)],
                         'model__learning_rate': [0.2, 0.4, 0.6, 0.8, 1.0],
                         'model__n_estimators': [50, 100, 150, 200, 250, 300,
                                                 350, 400]},
             scoring='neg_root_mean_squared_error')
```

In [35]:
```
cv_res = pd.DataFrame({
  "B": np.array(search.cv_results_["param_model__n_estimators"]),
  "rmse": -search.cv_results_["mean_test_score"],
  "lambda": search.cv_results_["param_model__learning_rate"],
  "depth": search.cv_results_["param_model__base_estimator"],
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "B",
  y = "rmse",
  hue = "lambda",
  style = "depth"
  ).set(
    xlabel = "B",
    ylabel = "CV RMSE"
);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [36]:
```
-search.best_score_
```

Out[36]:
```
0.17728787774924967
```

In [37]:
```
search.best_estimator_
```

Out[37]:
```
Pipeline(steps=[('model',
                 AdaBoostRegressor(base_estimator=DecisionTreeRegressor(max_depth=4),
                                   learning_rate=0.2, n_estimators=200,
                                   random_state=425))])
```

In [38]:
```
mean_squared_error(
  Y_test,
  search.best_estimator_.predict(X_test),
  squared = False
  )
```

Out[38]:
```
0.15874890293823835
```

Random forest performed best.

8.4.Carseat

In [39]:
```
df2 = pd.read_csv("Desktop/Cars.csv")
```

In [40]:
```
df2
```

Out[40]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US
0	9.50	138	73	11	276	120	Bad	42	17	Yes	Yes
1	11.22	111	48	16	260	83	Good	65	10	Yes	Yes
2	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes
3	7.40	117	100	4	466	97	Medium	55	14	Yes	Yes
4	4.15	141	64	3	340	128	Bad	38	13	Yes	No
...	...	...	...	...	...	...	...	...	...	...	...
395	12.57	138	108	17	203	128	Good	33	14	Yes	Yes
396	6.14	139	23	3	37	120	Medium	55	11	No	Yes
397	7.41	162	26	12	368	159	Medium	40	18	Yes	Yes
398	5.94	100	79	7	284	95	Bad	50	12	Yes	Yes
399	9.71	134	37	0	27	120	Good	49	16	Yes	Yes

400 rows × 11 columns

In [41]:
```
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   Sales        400 non-null    float64
 1   CompPrice    400 non-null    int64  
 2   Income       400 non-null    int64  
 3   Advertising  400 non-null    int64  
 4   Population   400 non-null    int64  
 5   Price        400 non-null    int64  
 6   ShelveLoc    400 non-null    object 
 7   Age          400 non-null    int64  
 8   Education    400 non-null    int64  
 9   Urban        400 non-null    object 
 10  US           400 non-null    object 
dtypes: float64(1), int64(7), object(3)
memory usage: 34.5+ KB
```

In [42]:
```
df2['Sales1'] = df2.Sales.map(lambda x: 1 if x>8 else 0)
```

In [43]:
```
df2
```

Out[43]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US	Sales1
0	9.50	138	73	11	276	120	Bad	42	17	Yes	Yes	1
1	11.22	111	48	16	260	83	Good	65	10	Yes	Yes	1
2	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes	1
3	7.40	117	100	4	466	97	Medium	55	14	Yes	Yes	0
4	4.15	141	64	3	340	128	Bad	38	13	Yes	No	0
...	...	...	...	...	...	...	...	...	...	...	...	...
395	12.57	138	108	17	203	128	Good	33	14	Yes	Yes	1
396	6.14	139	23	3	37	120	Medium	55	11	No	Yes	0
397	7.41	162	26	12	368	159	Medium	40	18	Yes	Yes	0
398	5.94	100	79	7	284	95	Bad	50	12	Yes	Yes	0
399	9.71	134	37	0	27	120	Good	49	16	Yes	Yes	1

400 rows × 12 columns

In [44]:
```
X1 = df2.drop(['Sales', 'Sales1'], axis=1)
Y1 = df2['Sales1']
X1_train, X1_test, Y1_train, Y1_test = train_test_split(X1, Y1, test_size=0.25, random_state=101)
```

In [45]:
```
num_features = ['CompPrice', 'Income', 'Advertising', 'Population', 'Price', 'Age', 'Education']
cat_features = ['Urban', 'US','ShelveLoc']
features = np.concatenate([num_features, cat_features])
```

In [46]:
```
categorical_tf = Pipeline(steps = [
  ("cat_impute", SimpleImputer(strategy = 'most_frequent')),
  ("encoder", OneHotEncoder())
])

# Transformer for continuous variables
numeric_tf = Pipeline(steps = [
  ("num_impute", SimpleImputer(strategy = 'mean')),
])

# Column transformer
col_tf = ColumnTransformer(transformers = [
  ('num', numeric_tf, num_features),
  ('cat', categorical_tf, cat_features)
])
```

In [47]:
```
classtree_mod = DecisionTreeClassifier(
  criterion = 'gini',
  random_state = 425
  )
```

In [48]:
```
pipe = Pipeline(steps = [
  ("col_tf", col_tf),
  ("model", classtree_mod)
  ])
pipe
```

Out[48]:
```
Pipeline(steps=[('col_tf',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('num_impute',
                                                                   SimpleImputer())]),
                                                  ['CompPrice', 'Income',
                                                   'Advertising', 'Population',
                                                   'Price', 'Age',
                                                   'Education']),
                                                 ('cat',
                                                  Pipeline(steps=[('cat_impute',
                                                                   SimpleImputer(strategy='most_frequent')),
                                                                  ('encoder',
                                                                   OneHotEncoder())]),
                                                  ['Urban', 'US',
                                                   'ShelveLoc'])])),
                ('model', DecisionTreeClassifier(random_state=425))])
```

In [49]:
```
ccp_alpha_grid = np.linspace(start = 0.0, stop = 0.05, num = 100)
tuned_parameters = {
  "model__ccp_alpha": ccp_alpha_grid
  }
tuned_parameters
```

Out[49]:
```
{'model__ccp_alpha': array([0.        , 0.00050505, 0.0010101 , 0.00151515, 0.0020202 ,
        0.00252525, 0.0030303 , 0.00353535, 0.0040404 , 0.00454545,
        0.00505051, 0.00555556, 0.00606061, 0.00656566, 0.00707071,
        0.00757576, 0.00808081, 0.00858586, 0.00909091, 0.00959596,
        0.01010101, 0.01060606, 0.01111111, 0.01161616, 0.01212121,
        0.01262626, 0.01313131, 0.01363636, 0.01414141, 0.01464646,
        0.01515152, 0.01565657, 0.01616162, 0.01666667, 0.01717172,
        0.01767677, 0.01818182, 0.01868687, 0.01919192, 0.01969697,
        0.02020202, 0.02070707, 0.02121212, 0.02171717, 0.02222222,
        0.02272727, 0.02323232, 0.02373737, 0.02424242, 0.02474747,
        0.02525253, 0.02575758, 0.02626263, 0.02676768, 0.02727273,
        0.02777778, 0.02828283, 0.02878788, 0.02929293, 0.02979798,
        0.03030303, 0.03080808, 0.03131313, 0.03181818, 0.03232323,
        0.03282828, 0.03333333, 0.03383838, 0.03434343, 0.03484848,
        0.03535354, 0.03585859, 0.03636364, 0.03686869, 0.03737374,
        0.03787879, 0.03838384, 0.03888889, 0.03939394, 0.03989899,
        0.04040404, 0.04090909, 0.04141414, 0.04191919, 0.04242424,
        0.04292929, 0.04343434, 0.04393939, 0.04444444, 0.04494949,
        0.04545455, 0.0459596 , 0.04646465, 0.0469697 , 0.04747475,
        0.0479798 , 0.04848485, 0.0489899 , 0.04949495, 0.05      ])}
```

In [50]:

```
n_folds = 5
search = GridSearchCV(
  pipe,
  tuned_parameters,
  cv = n_folds,
  scoring = "roc_auc",
  # Refit the best model on the whole data set
  refit = True
  )
```

In [51]:

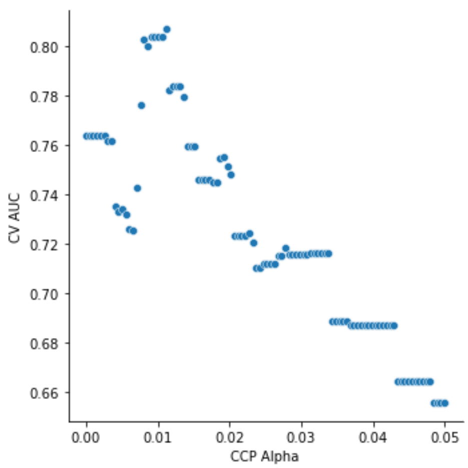
```
search.fit(X1_train,Y1_train)
```

Out[51]:

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('col_tf',
                                        ColumnTransformer(transformers=[('num',
                                                                         Pipeline(steps=[('num_impute',
                                                                                          SimpleImputer())]),
                                                                         ['CompPrice',
                                                                          'Income',
                                                                          'Advertising',
                                                                          'Population',
                                                                          'Price',
                                                                          'Age',
                                                                          'Education']),
                                                                        ('cat',
                                                                         Pipeline(steps=[('cat_impute',
                                                                                          SimpleImputer(strategy='mos
t_frequent')),
                                                                                         ('encoder',
                                                                                          OneHotEncoder())]),
                                                                         ['Urban',
                                                                          'US',
                                                                          'ShelveLo...
       0.03282828, 0.03333333, 0.03383838, 0.03434343, 0.03484848,
       0.03535354, 0.03585859, 0.03636364, 0.03686869, 0.03737374,
       0.03787879, 0.03838384, 0.03888889, 0.03939394, 0.03989899,
       0.04040404, 0.04090909, 0.04141414, 0.04191919, 0.04242424,
       0.04292929, 0.04343434, 0.04393939, 0.04444444, 0.04494949,
       0.04545455, 0.0459596 , 0.04646465, 0.0469697 , 0.04747475,
       0.0479798 , 0.04848485, 0.0489899 , 0.04949495, 0.05      ])},
             scoring='roc_auc')
```

In [52]:
```
cv_res = pd.DataFrame({
  "ccp_alpha": np.array(search.cv_results_["param_model__ccp_alpha"]),
  "auc": search.cv_results_["mean_test_score"]
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "ccp_alpha",
  y = "auc"
  ).set(
    xlabel = "CCP Alpha",
    ylabel = "CV AUC"
);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [53]:
```
search.best_score_
```

Out[53]:
```
0.806889592760181
```

In [57]:
```
accuracy_score(
  Y1_test,
  search.best_estimator_.predict(X1_test)
  )
```

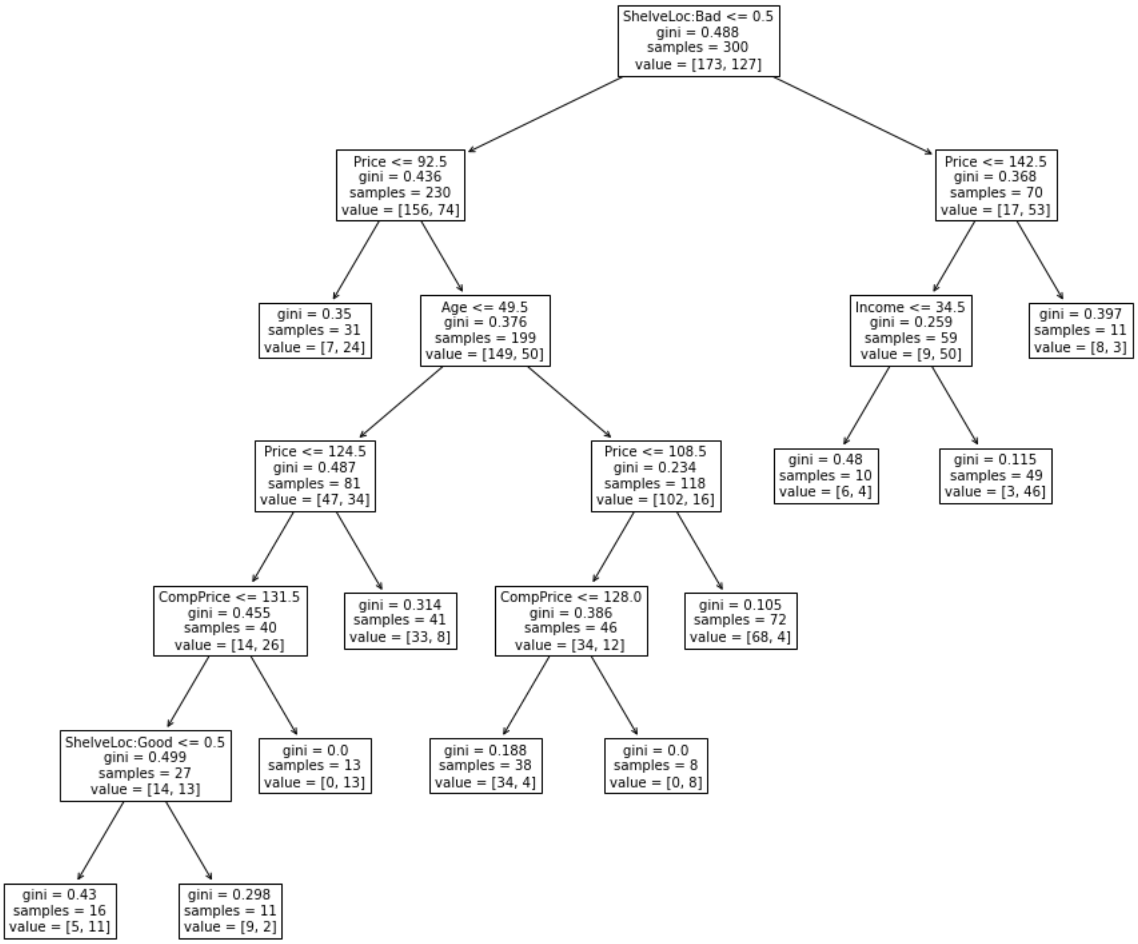
Out[57]:
```
0.77
```

In [58]:
```
search.best_estimator_
```

Out[58]:
```
Pipeline(steps=[('col_tf',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('num_impute',
                                                                   SimpleImputer())]),
                                                  ['CompPrice', 'Income',
                                                   'Advertising', 'Population',
                                                   'Price', 'Age',
                                                   'Education']),
                                                 ('cat',
                                                  Pipeline(steps=[('cat_impute',
                                                                   SimpleImputer(strategy='most_frequent')),
                                                                  ('encoder',
                                                                   OneHotEncoder())]),
                                                  ['Urban', 'US',
                                                   'ShelveLoc'])])),
                ('model',
                 DecisionTreeClassifier(ccp_alpha=0.011111111111111112,
                                        random_state=425))])
```

In [61]:

```
features = np.concatenate([
    features[:-3],
    ['ShelveLoc:Bad', 'ShelveLoc:Good', 'ShelveLoc:Medium'],
    ['Urban:Yes', 'Urban:No'],
    ['US:Yes' , 'US:No']
    ])

plt.figure(figsize=(16,14))
plot_tree(
  search.best_estimator_['model'],
  feature_names = features
  );
plt.show()
```



In [68]:

```
df3 = df2.copy()
```

In [69]:
```
df3
```

Out[69]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urban	US	Sales1
0	9.50	138	73	11	276	120	Bad	42	17	Yes	Yes	1
1	11.22	111	48	16	260	83	Good	65	10	Yes	Yes	1
2	10.06	113	35	10	269	80	Medium	59	12	Yes	Yes	1
3	7.40	117	100	4	466	97	Medium	55	14	Yes	Yes	0
4	4.15	141	64	3	340	128	Bad	38	13	Yes	No	0
...	...	...	...	...	...	...	...	...	...	...	...	...
395	12.57	138	108	17	203	128	Good	33	14	Yes	Yes	1
396	6.14	139	23	3	37	120	Medium	55	11	No	Yes	0
397	7.41	162	26	12	368	159	Medium	40	18	Yes	Yes	0
398	5.94	100	79	7	284	95	Bad	50	12	Yes	Yes	0
399	9.71	134	37	0	27	120	Good	49	16	Yes	Yes	1

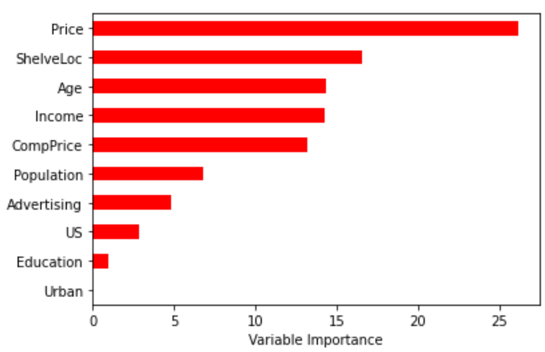
400 rows × 12 columns

In [70]:
```
df3['ShelveLoc'] = df3['ShelveLoc'].map({'Bad':0, 'Medium':1,'Good':2})
df3['Urban'] = df3['Urban'].map({'No':0, 'Yes':1})
df3['US'] = df3['US'].map({'No':0, 'Yes':1})
```

In [93]:
```
X2 = df3.drop(['Sales', 'Sales1'], axis=1)
Y2 = df3['Sales1']
X2_train, X2_test, Y2_train, Y2_test = train_test_split(X2, Y2, test_size=0.25, random_state=101)
```

In [138]:
```
mod1 = classtree_mod.fit(X2_train,Y2_train)
```

In [139]:
```
Importance = pd.DataFrame({'Importance':mod1.feature_importances_*100}, index=X2.columns)
Importance.sort_values(by='Importance', axis=0, ascending=True).plot(kind='barh', color='r', )
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```



In [74]:
```
roc_auc_score(
  Y1_test,
  search.best_estimator_.predict_proba(X1_test)[:, 1]
  )
```

Out[74]: 0.7867867867867868

In [77]:
```
accuracy_score(
  Y1_test,
  search.best_estimator_.predict(X1_test)
  )
```

Out[77]: 0.77

Random Forest Classifier

In [80]:
```
rf_mod = RandomForestClassifier(
  # Number of trees
  n_estimators = 100,
  criterion = 'gini',
  # Number of features to use in each split
  max_features = 'sqrt',
  oob_score = True,
  random_state = 425
  )
```

In [81]:
```
pipe = Pipeline(steps = [
  ("col_tf", col_tf),
  ("model", rf_mod)
  ])
pipe
```

Out[81]:
```
Pipeline(steps=[('col_tf',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('num_impute',
                                                                   SimpleImputer())]),
                                                  ['CompPrice', 'Income',
                                                   'Advertising', 'Population',
                                                   'Price', 'Age',
                                                   'Education']),
                                                 ('cat',
                                                  Pipeline(steps=[('cat_impute',
                                                                   SimpleImputer(strategy='most_frequent')),
                                                                  ('encoder',
                                                                   OneHotEncoder())]),
                                                  ['Urban', 'US',
                                                   'ShelveLoc'])])),
                ('model',
                 RandomForestClassifier(max_features='sqrt', oob_score=True,
                                        random_state=425))])
```

In [82]:
```
B_grid = [50, 100, 150, 200, 250, 300]
m_grid = ['sqrt', 'log2', 1.0] # max_features = 1.0 uses all features
tuned_parameters = {
  "model__n_estimators": B_grid,
  "model__max_features": m_grid
  }
tuned_parameters
```

Out[82]:
```
{'model__n_estimators': [50, 100, 150, 200, 250, 300],
 'model__max_features': ['sqrt', 'log2', 1.0]}
```

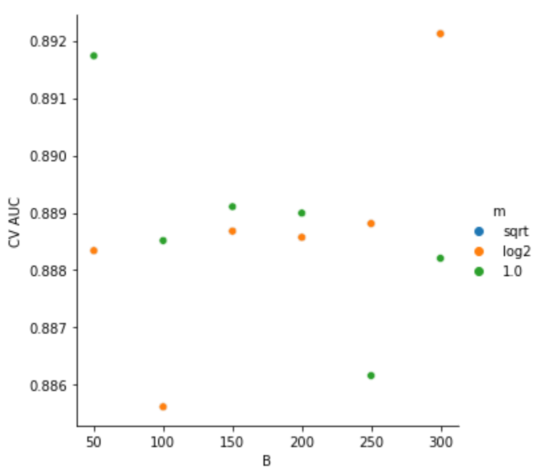
In [83]:
```
n_folds = 5
search = GridSearchCV(
  pipe,
  tuned_parameters,
  cv = n_folds,
  scoring = "roc_auc",
  # Refit the best model on the whole data set
  refit = True
  )
```

In [84]:
```
search.fit(X1_train, Y1_train)
```

Out[84]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('col_tf',
                                        ColumnTransformer(transformers=[('num',
                                                                         Pipeline(steps=[('num_impute',
                                                                                          SimpleImputer())]),
                                                                         ['CompPrice',
                                                                          'Income',
                                                                          'Advertising',
                                                                          'Population',
                                                                          'Price',
                                                                          'Age',
                                                                          'Education']),
                                                                        ('cat',
                                                                         Pipeline(steps=[('cat_impute',
                                                                                          SimpleImputer(strategy='mos
t_frequent')),
                                                                                         ('encoder',
                                                                                          OneHotEncoder())]),
                                                                         ['Urban',
                                                                          'US',
                                                                          'ShelveLoc'])])),
                                       ('model',
                                        RandomForestClassifier(max_features='sqrt',
                                                               oob_score=True,
                                                               random_state=425))]),
             param_grid={'model__max_features': ['sqrt', 'log2', 1.0],
                         'model__n_estimators': [50, 100, 150, 200, 250, 300]},
             scoring='roc_auc')
```

In [85]:
```
cv_res = pd.DataFrame({
  "B": np.array(search.cv_results_["param_model__n_estimators"]),
  "auc": search.cv_results_["mean_test_score"],
  "m": search.cv_results_["param_model__max_features"]
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "B",
  y = "auc",
  hue = "m"
  ).set(
    xlabel = "B",
    ylabel = "CV AUC"
);
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



In [86]:
```
search.best_score_
```

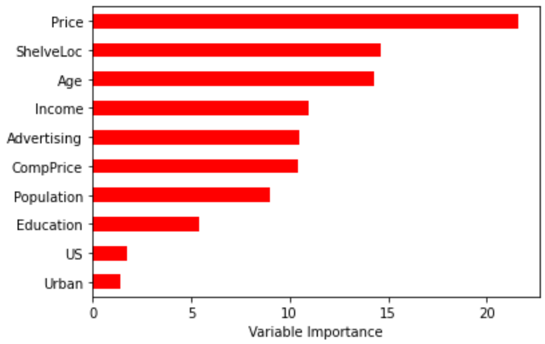
Out[86]:
```
0.8921239819004525
```

In [88]:
```
accuracy_score(
  Y1_train,
  search.best_estimator_.predict(X1_train)
)
```

Out[88]: 1.0

In [136]:
```
mod2 = rf_mod.fit(X2_train , Y2_train)
```

In [137]:
```
Importance = pd.DataFrame({'Importance':mod2.feature_importances_*100}, index=X2.columns)
Importance.sort_values(by='Importance', axis=0, ascending=True).plot(kind='barh', color='r', )
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```



In [100]:
```
roc_auc_score(
  Y1_test,
  search.best_estimator_.predict_proba(X1_test)[:, 1]
  )
```

Out[100]: 0.8983268983268984

In [101]:
```
accuracy_score(
  Y1_test,
  search.best_estimator_.predict(X1_test)
  )


```

Out[101]: 0.83

Boosting

In [106]:
```
bst_mod = AdaBoostClassifier(
  base_estimator = DecisionTreeClassifier(max_depth = 3),
  # Number of trees (to be tuned)
  n_estimators = 50,
  # Learning rate (to be tuned)
  learning_rate = 1.0,
  random_state = 425
  )
```

In [107]:
```
pipe = Pipeline(steps = [
  ("col_tf", col_tf),
  ("model", bst_mod)
  ])
pipe
```

Out[107]:
```
Pipeline(steps=[('col_tf',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('num_impute',
                                                                   SimpleImputer())]),
                                                  ['CompPrice', 'Income',
                                                   'Advertising', 'Population',
                                                   'Price', 'Age',
                                                   'Education']),
                                                 ('cat',
                                                  Pipeline(steps=[('cat_impute',
                                                                   SimpleImputer(strategy='most_frequent')),
                                                                  ('encoder',
                                                                   OneHotEncoder())]),
                                                  ['Urban', 'US',
                                                   'ShelveLoc'])])),
                ('model',
                 AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                                    random_state=425))])
```

In [111]:
```
d_grid = [
  DecisionTreeClassifier(max_depth = 1),
  DecisionTreeClassifier(max_depth = 2),
  DecisionTreeClassifier(max_depth = 3),
  DecisionTreeClassifier(max_depth = 4)
  ]
B_grid = np.linspace(10, 100, 10).astype(int)
lambda_grid = [0.2, 0.4, 0.6, 0.8, 1.0]
tuned_parameters = {
  "model__base_estimator": d_grid,
  "model__n_estimators": B_grid,
  "model__learning_rate": lambda_grid
  }
tuned_parameters
```

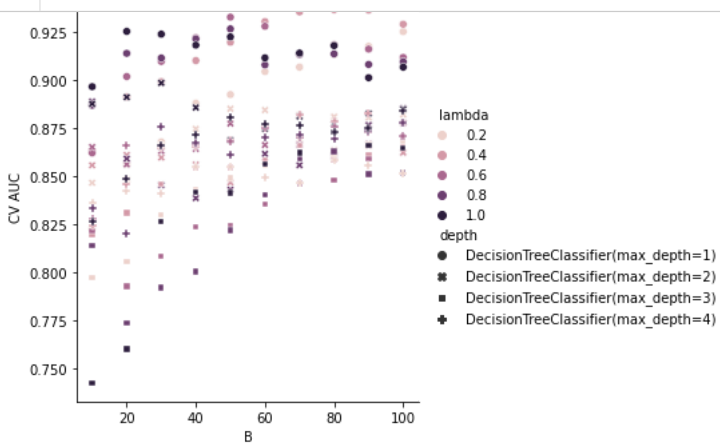
Out[111]:
```
{'model__base_estimator': [DecisionTreeClassifier(max_depth=1),
  DecisionTreeClassifier(max_depth=2),
  DecisionTreeClassifier(max_depth=3),
  DecisionTreeClassifier(max_depth=4)],
 'model__n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100]),
 'model__learning_rate': [0.2, 0.4, 0.6, 0.8, 1.0]}
```

In [112]:
```
n_folds = 5
search = GridSearchCV(
  pipe,
  tuned_parameters,
  cv = n_folds,
  scoring = "roc_auc",
  # Refit the best model on the whole data set
  refit = True
  )
```

In [113]:
```
search.fit(X1_train,Y1_train)
```

Out[113]:
```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('col_tf',
                                        ColumnTransformer(transformers=[('num',
                                                                         Pipeline(steps=[('num_impute',
                                                                                          SimpleImputer())]),
                                                                         ['CompPrice',
                                                                          'Income',
                                                                          'Advertising',
                                                                          'Population',
                                                                          'Price',
                                                                          'Age',
                                                                          'Education']),
                                                                        ('cat',
                                                                         Pipeline(steps=[('cat_impute',
                                                                                          SimpleImputer(strategy='mos
t_frequent')),
                                                                                         ('encoder',
                                                                                          OneHotEncoder())]),
                                                                         ['Urban',
                                                                          'US',
                                                                          'ShelveLo...
                                        AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=3),
                                                           random_state=425))]),
             param_grid={'model__base_estimator': [DecisionTreeClassifier(max_depth=1),
                                                   DecisionTreeClassifier(max_depth=2),
                                                   DecisionTreeClassifier(max_depth=3),
                                                   DecisionTreeClassifier(max_depth=4)],
                         'model__learning_rate': [0.2, 0.4, 0.6, 0.8, 1.0],
                         'model__n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100])},
             scoring='roc_auc')
```

In [115]:
```
cv_res = pd.DataFrame({
  "B": np.array(search.cv_results_["param_model__n_estimators"]),
  "auc": search.cv_results_["mean_test_score"],
  "lambda": search.cv_results_["param_model__learning_rate"],
  "depth": search.cv_results_["param_model__base_estimator"],
  })

plt.figure()
sns.relplot(
  # kind = "line",
  data = cv_res,
  x = "B",
  y = "auc",
  hue = "lambda",
  style = "depth"
  ).set(
    xlabel = "B",
    ylabel = "CV AUC"
);
plt.show()
```



In [125]:
```
search.best_score_
```

Out[125]:
```
0.9366921784098257
```

In [117]:
```
accuracy_score(
  Y1_train,
  search.best_estimator_.predict(X1_train)
  )
```

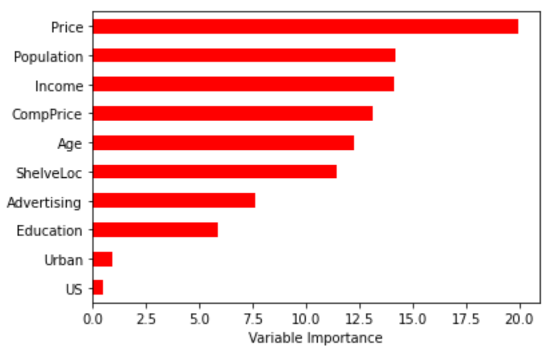
Out[117]: 0.94

In [126]:
```
search.best_estimator_
```

Out[126]:
```
Pipeline(steps=[('col_tf',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('num_impute',
                                                                   SimpleImputer())]),
                                                  ['CompPrice', 'Income',
                                                   'Advertising', 'Population',
                                                   'Price', 'Age',
                                                   'Education']),
                                                 ('cat',
                                                  Pipeline(steps=[('cat_impute',
                                                                   SimpleImputer(strategy='most_frequent')),
                                                                  ('encoder',
                                                                   OneHotEncoder())]),
                                                  ['Urban', 'US',
                                                   'ShelveLoc'])])),
                ('model',
                 AdaBoostClassifier(base_estimator=DecisionTreeClassifier(max_depth=1),
                                    learning_rate=0.4, n_estimators=80,
                                    random_state=425))])
```

In [128]:
```
mod3 = bst_mod.fit(X2_train,Y2_train)
```

In [135]:
```
Importance = pd.DataFrame({'Importance':mod3.feature_importances_*100}, index=X2.columns)
Importance.sort_values(by='Importance', axis=0, ascending=True).plot(kind='barh', color='r', )
plt.xlabel('Variable Importance')
plt.gca().legend_ = None
```



In [127]:
```
roc_auc_score(
  Y1_test,
  search.best_estimator_.predict_proba(X1_test)[:, 1]
  )
```

Out[127]: 0.9223509223509224

In [119]:
```
accuracy_score(
  Y1_test,
  search.best_estimator_.predict(X1_test)
  )
```

Out[119]: 0.86

Boosting performed best.