Isha Jain

Section 6.6

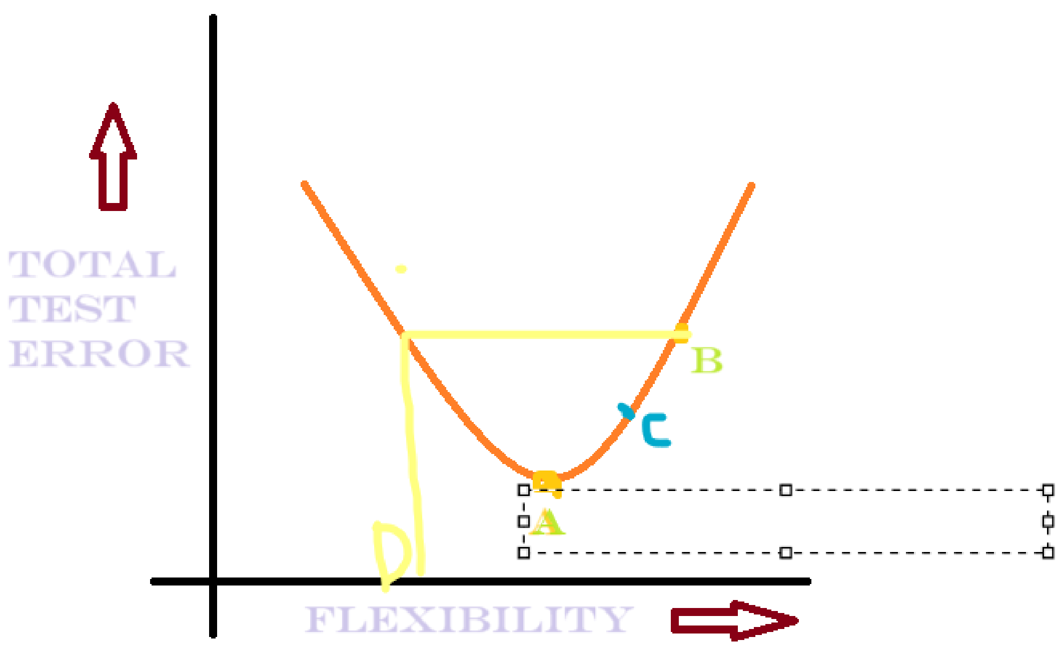
Conceptual

2. (a) i. Incorrect. The lasso is less flexible than least squares as it reduces the number of predictors in the model (uses a revised cost function, penalizing larger coefficients more).

ii. Incorrect. The lasso is less flexible than least squares as it reduces the number of predictors in the model (uses a revised cost function, penalizing larger coefficients more).

iii. Correct. Lasso is less flexible than least squares as it reduces the number of predictors in the model (uses a revised cost function, penalizing larger coefficients more). This would lead to a model with lower variance and higher bias. The increase in bias would have to be lower than the reduction in variance for improved prediction accuracy (accuracy would improve when model was giving higher total error than that at the optimal complexity, and complexity reduces appropriately, to not result in a higher increase in bias than a reduction in variance.

(Improvement in accuracy with Lasso when: -The least squares model lies to the right of A: optimal bias-variance , let's say B and the reduction in flexibility would lead to improved accuracy only in a certain range up to D)



.

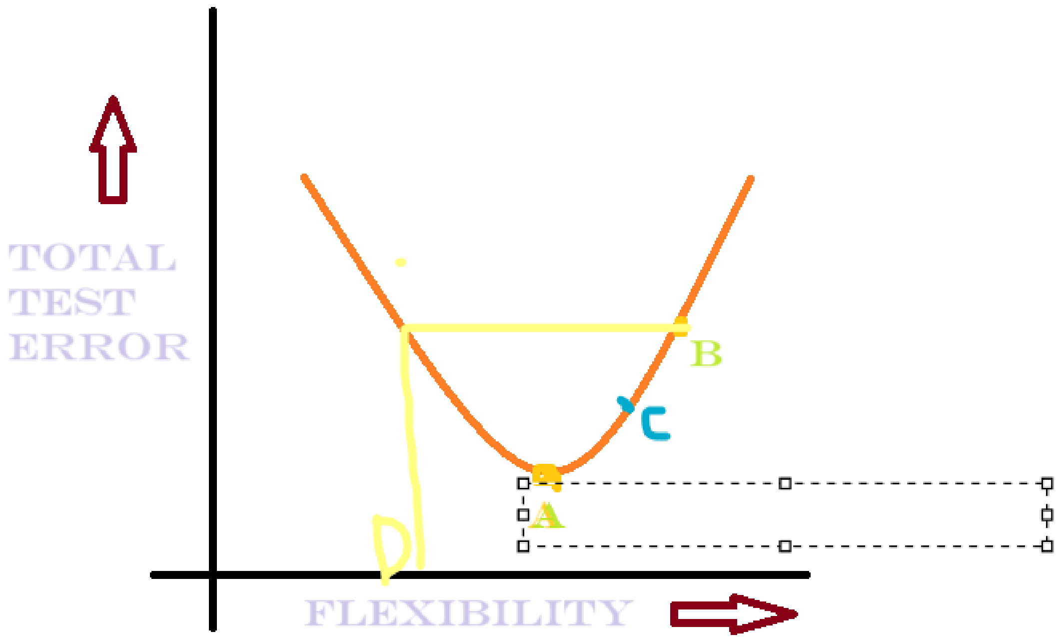
iv. Incorrect. Lasso is less flexible than least squares as it reduces the number of predictors in the model (uses a revised cost function, penalizing larger coefficients more). This would lead to a model with lower variance and higher bias.

(b) i. Incorrect. Ridge Regression is less flexible than least squares as it reduces the number of predictors in the model (uses a revised cost function, penalizing larger coefficients more).

ii. Incorrect. Ridge Regression is less flexible than least squares as it reduces the number of predictors in the model (uses a revised cost function, penalizing larger coefficients more).

iii. Correct. Ridge Regression is less flexible than least squares as it reduces the number of predictors in the model (uses a revised cost function, penalizing larger coefficients more). This would lead to a model with lower variance and higher bias. The increase in bias would have to be lower than the reduction in variance for improved prediction accuracy (accuracy would improve when model was giving higher total error than that at the optimal complexity, and complexity reduces appropriately, to not result in a higher increase in bias than a reduction in variance.

(Improvement in accuracy with Ridge Regression when: -The least squares model lies to the right of A: optimal bias-variance , let's say B and the reduction in flexibility would lead to improved accuracy only in a certain range up to D)



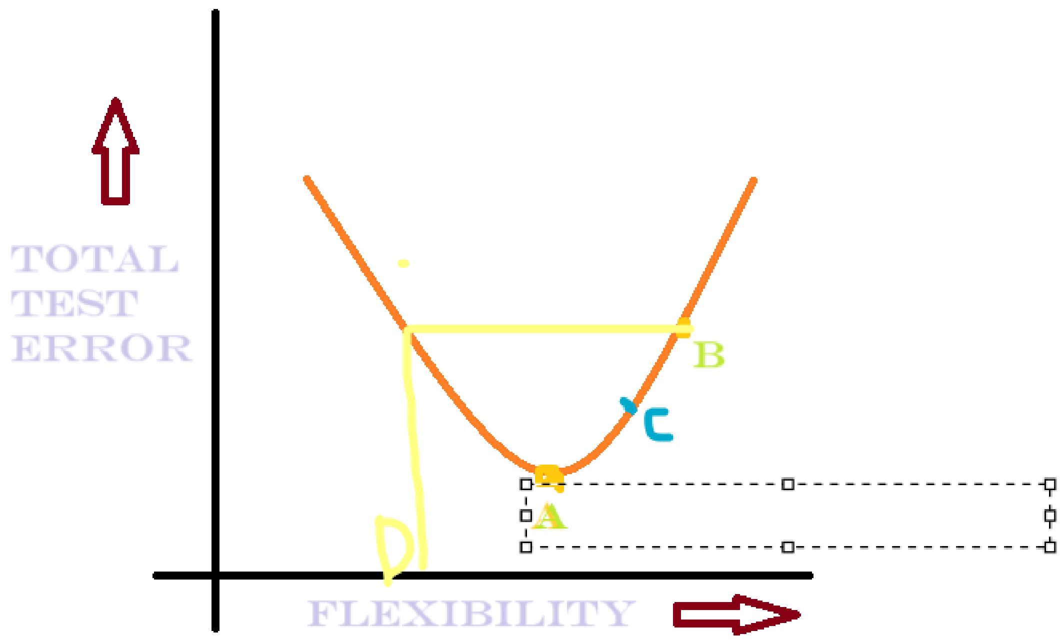
.

iv. Incorrect. Ridge Regression is less flexible than least squares as it reduces the number of predictors in the model (uses a revised cost function, penalizing larger coefficients more). This would lead to a model with lower variance and higher bias.

(c) i. Incorrect. Non-linear methods are more flexible than least squares and would hence lead to an increase in variance and reduction in bias.

ii. Correct. Non-linear methods are more flexible than least squares and would hence lead to an increase in variance and reduction in bias. Hence, they will give improved prediction accuracy when its increase in variance is less than its decrease in bias (overall error going down even with increased variance if offset by decrease in bias is larger).

(Improvement in accuracy with non-linear methods when:-The least squares model lies to the left of A: optimal bias-variance , let's say D and the increase in flexibility would lead to improved accuracy only in a certain range up to B)



iii. Incorrect. Non-linear methods are more flexible than least squares and would hence lead to an increase in variance and reduction in bias.

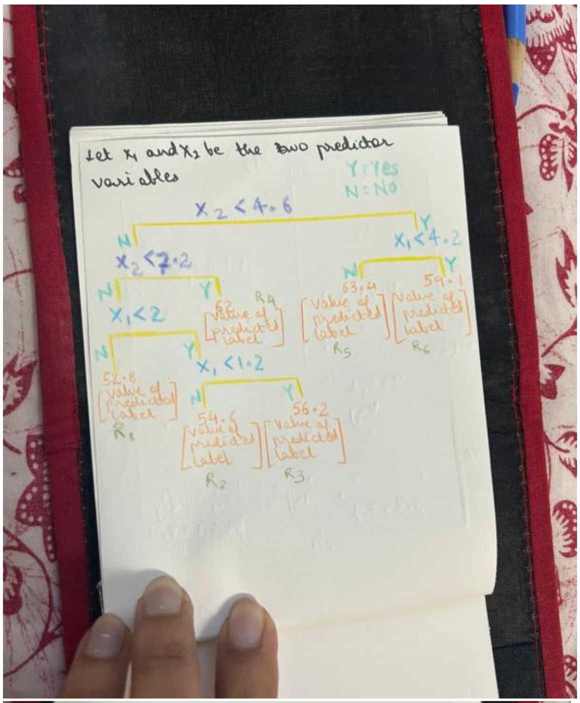
iv. Incorrect. Non-linear methods are more flexible than least squares.

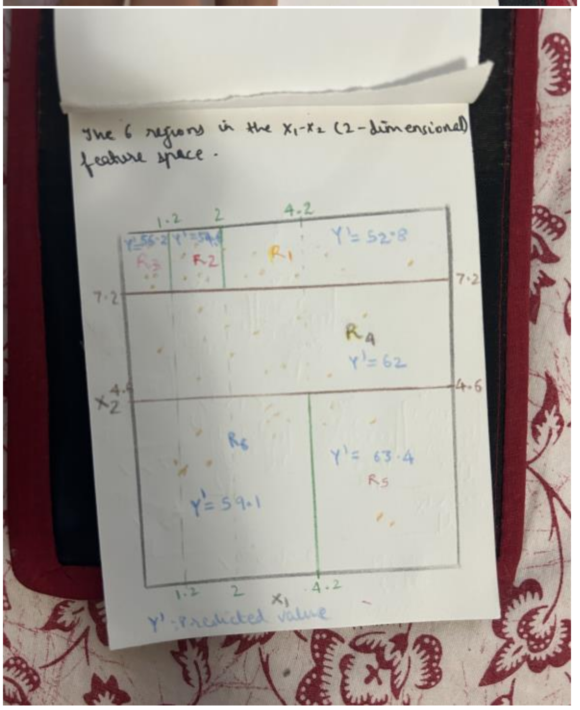
Isha Jain

Section 8.4

Conceptual

1.





Applied

8.

localhost:8888/notebooks/WPI/Stats%20for%20DS/Homework%234_Isha_Jain.ipynb

jupyter Homework#4_Isha_Jain Last Checkpoint: Last Sunday at 11:14 PM (unsaved changes)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

8

In [1]:
```
pip install ISLP;
```

```
Requirement already satisfied: ISLP in c:\users\ishaj\anaconda3\lib\site-packages (0.3.19)
Requirement already satisfied: pandas<=1.9,>=0.20 in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (1.3.4)
Requirement already satisfied: lxml in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (4.6.3)
Requirement already satisfied: numpy<1.25,>=1.7.1 in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (1.20.3)
Requirement already satisfied: scipy>=0.9 in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (1.7.1)
Requirement already satisfied: joblib in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (1.3.2)
Requirement already satisfied: statsmodels>=0.13 in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (0.14.0)
Requirement already satisfied: torch in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (2.0.1)
Requirement already satisfied: torchmetrics in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (1.1.2)
Requirement already satisfied: pytorch-lightning in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (2.0.9)
Requirement already satisfied: pygam in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (0.8.0)
Requirement already satisfied: lifelines in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (0.27.8)
Requirement already satisfied: scikit-learn>=1.2 in c:\users\ishaj\anaconda3\lib\site-packages (from ISLP) (1.3.0)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\ishaj\anaconda3\lib\site-packages (from pandas<=1.9,>=0.20->ISLP) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\ishaj\anaconda3\lib\site-packages (from pandas<=1.9,>=0.20->ISLP) (2021.3)
Requirement already satisfied: six>=1.5 in c:\users\ishaj\anaconda3\lib\site-packages (from python-dateutil>=2.7.3->pandas<=1.9,>=0.20->ISLP) (1.16.0)
```

In [2]:
```python
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
from statsmodels.datasets import get_rdataset
import sklearn.model_selection as skm
from ISLP import load_data, confusion_table
from ISLP.models import ModelSpec as MS
```

In [3]:
```python
from sklearn.tree import (DecisionTreeClassifier as DTC,
      DecisionTreeRegressor as DTR,
      plot_tree,
      export_text)
from sklearn.metrics import (accuracy_score,
      log_loss)
from sklearn.ensemble import \
    (RandomForestRegressor as RF,
    GradientBoostingRegressor as GBR)
from ISLP.bart import BART
```

In [4]:
```python
Carseats = load_data('Carseats')
Carseats.head(5)
```

Out[4]:

	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

In [5]:
```python
model = MS(Carseats.columns.drop('Sales'), intercept=False)
D = model.fit_transform(Carseats)
feature_names = list(D.columns)
X = np.asarray(D)
```

In [7]:
```python
print(feature_names)
```

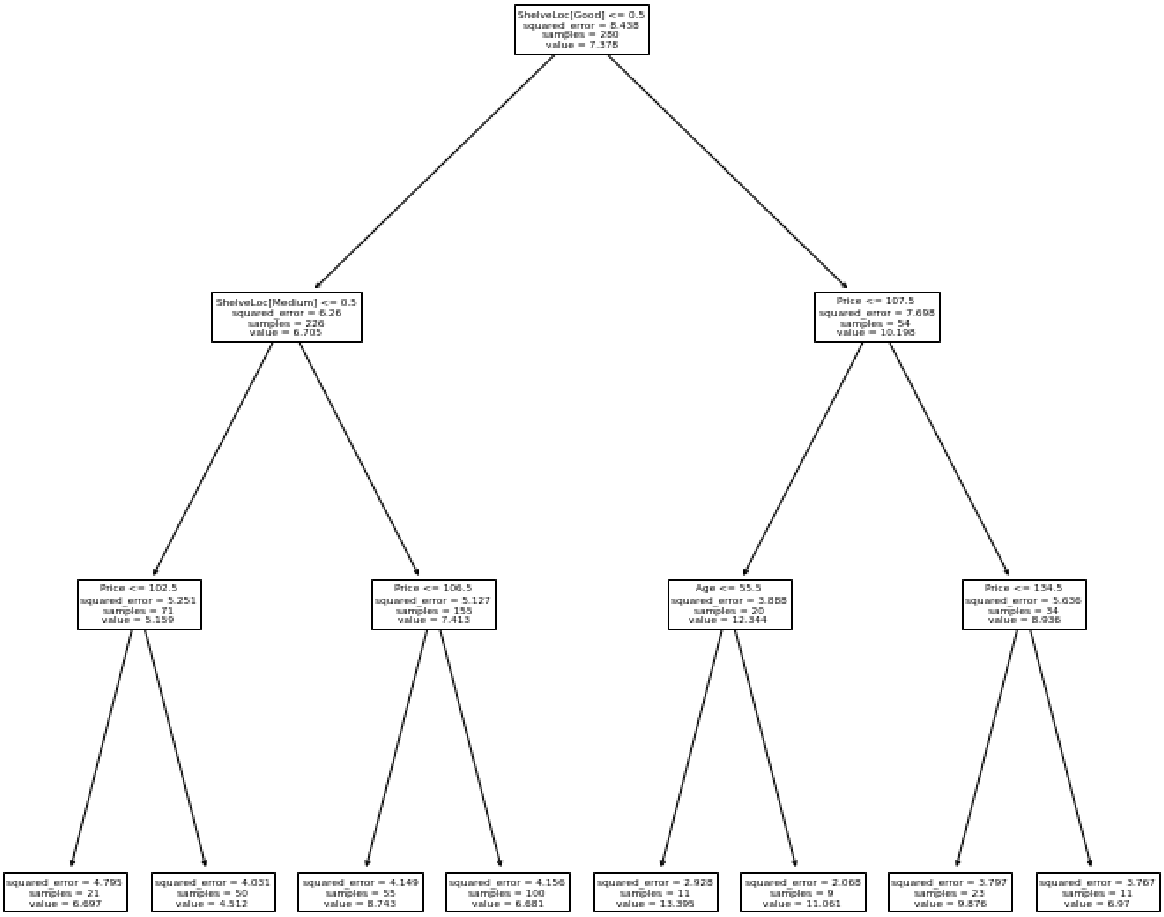
```
['CompPrice', 'Income', 'Advertising', 'Population', 'Price', 'ShelveLoc[Good]', 'ShelveLoc[Medium]', 'Age', 'Education', 'Urban[Yes]', 'US[Yes]']
```

(a)

```
In [8]: #creating test set with 30 percent of the dataset
from sklearn.model_selection import train_test_split

y = Carseats['Sales']

# split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=0)
```

(b)

```
In [13]: reg = DTR(max_depth=3)
reg.fit(X_train, y_train)
ax = subplots(figsize=(12,12))[1]
plot_tree(reg,
          feature_names=feature_names,
          ax=ax);
```



Above is the decision tree obtained specifying maximum depth as 3 (to make the tree plot more legible, as higher depths led to illegible text within the tree). The root decision node is corresponding to the ShelveLoc being good, implying ShelveLoc to the the most important parameter, post which, if ShelveLoc isn't good, it's checked if ShelveLoc is Medium. Post this, we make 2 different predictions based on the price. If the ShelveLoc was good, the next question is whether the price is below a certain value (107.5 in this case). If the price is lesser, we look at age to arrive at a prediction. If the price was greater (than 107.5), we look at another price (134.5), to make predictions.

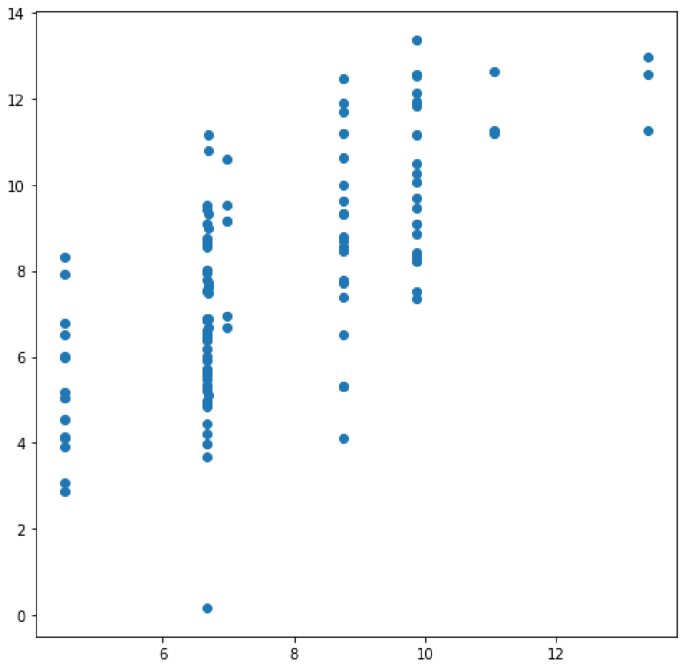
In [15]:
```
#Representing tree with text
print(export_text(reg,
      feature_names=feature_names,
      show_weights=True))
```

```
|--- ShelveLoc[Good] <= 0.50
|   |--- ShelveLoc[Medium] <= 0.50
|   |   |--- Price <= 102.50
|   |   |   |--- value: [6.70]
|   |   |--- Price >  102.50
|   |   |   |--- value: [4.51]
|   |--- ShelveLoc[Medium] >  0.50
|   |   |--- Price <= 106.50
|   |   |   |--- value: [8.74]
|   |   |--- Price >  106.50
|   |   |   |--- value: [6.68]
|--- ShelveLoc[Good] >  0.50
|   |--- Price <= 107.50
|   |   |--- Age <= 55.50
|   |   |   |--- value: [13.39]
|   |   |--- Age >  55.50
|   |   |   |--- value: [11.06]
|   |--- Price >  107.50
|   |   |--- Price <= 134.50
|   |   |   |--- value: [9.88]
|   |   |--- Price >  134.50
|   |   |   |--- value: [6.97]
```

In [14]:
```
#scatterplot of true label vs predicted label and test MSE
ax = subplots(figsize=(8,8))[1]
y_hat = reg.predict(X_test)
ax.scatter(y_hat, y_test)
np.mean((y_test - y_hat)**2)
```

Out[14]: 3.703858275236092

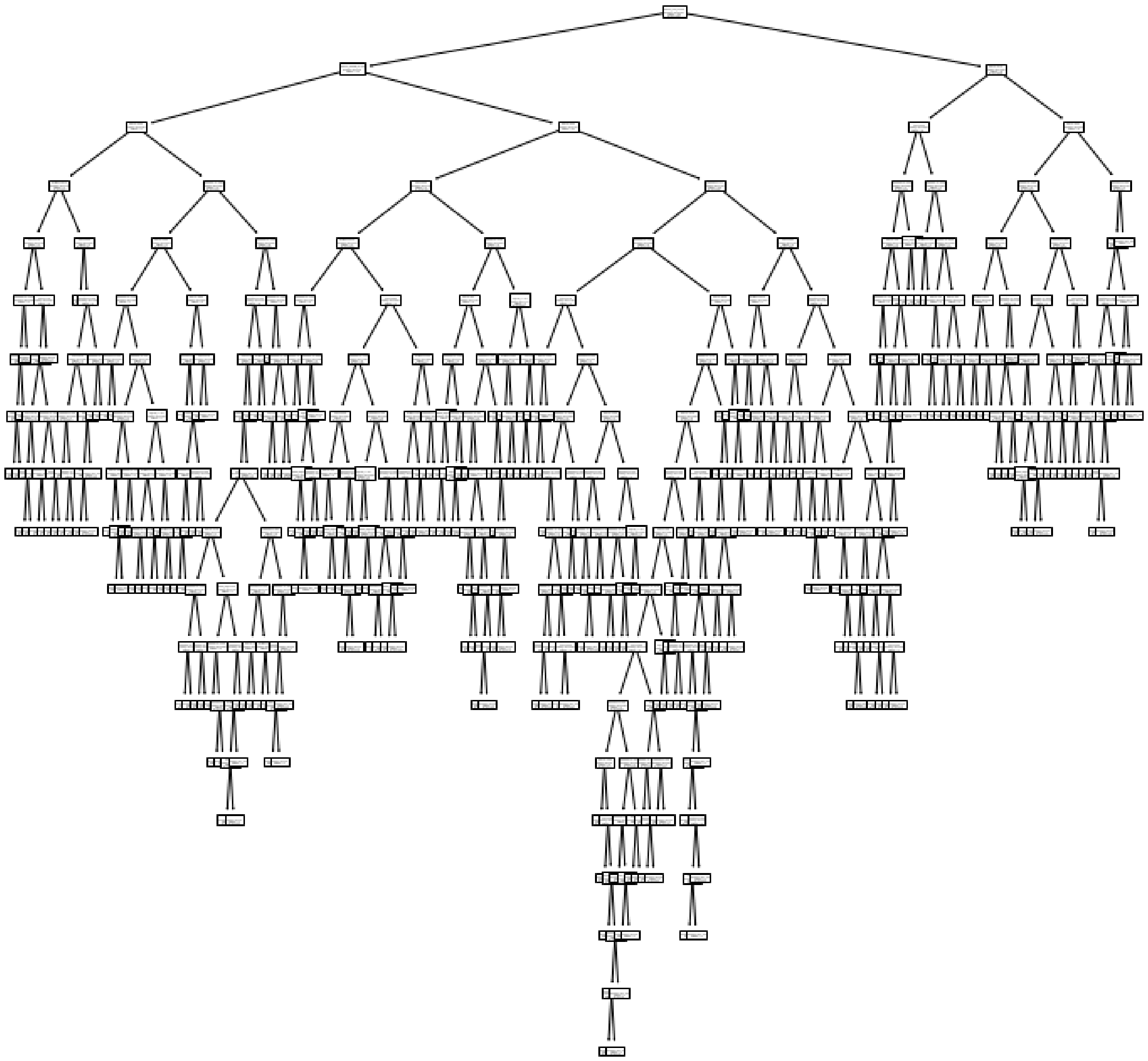
MSE of 3.7 (rounded to 2 decimal places) is obtained.



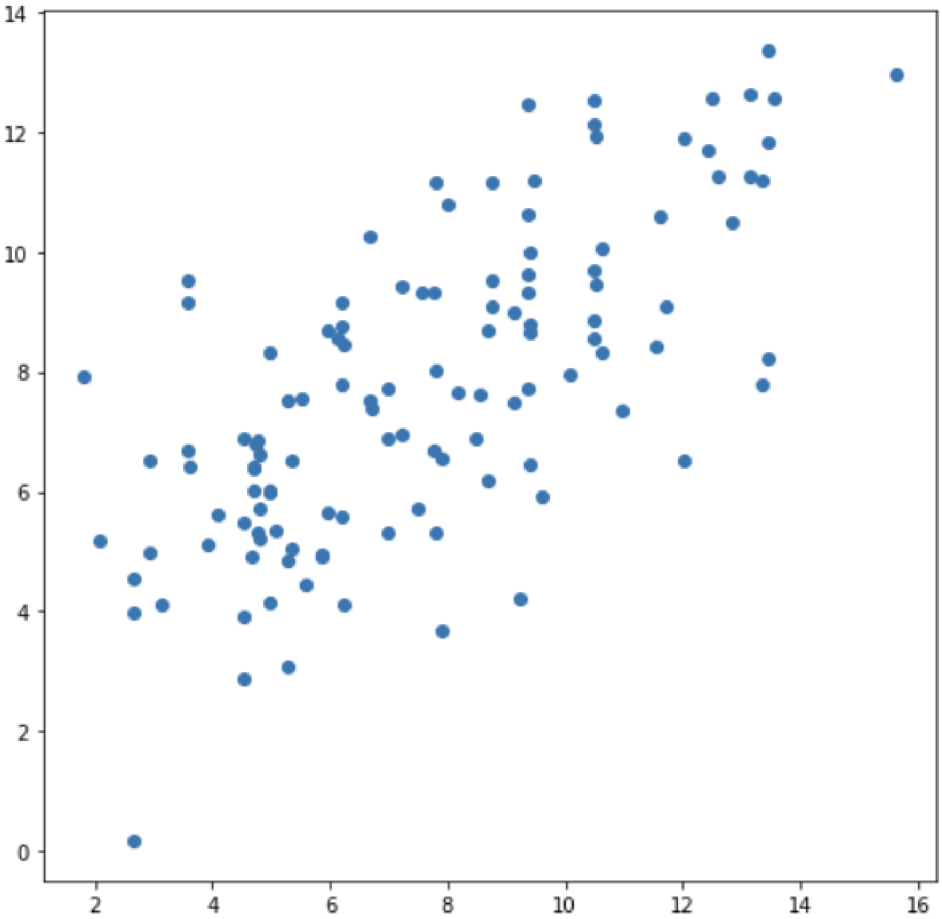
(c)

In [111]:
```
reg = DTR()
reg.fit(X_train, y_train)
ax = subplots(figsize=(12,12))[1]
plot_tree(reg,
          feature_names=feature_names,
          ax=ax);
```

Decision tree with no maximum depth criteria



```
In [125]: #scatterplot of true label vs predicted label and test MSE
          ax = subplots(figsize=(8,8))[1]
          y_hat = reg.predict(X_test)
          ax.scatter(y_hat, y_test)
          np.mean((y_test - y_hat)**2)

Out[125]: 5.031036666666664
```



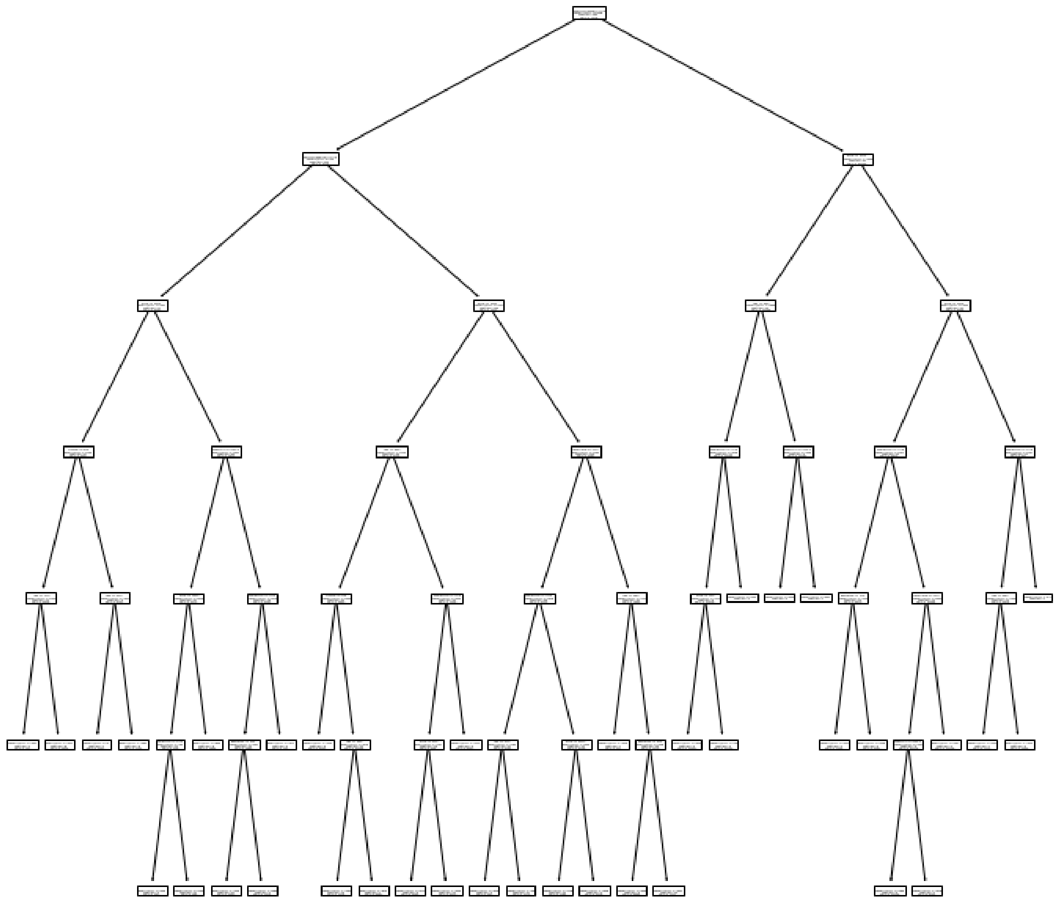
On pruning the above tree:

The test error improves from 5.03 (rounded to 2 decimal places) to 4.47 (rounded to 2 decimal places). The pruned tree has a depth of 6.

```
In [122]: #Pruning
          ccp_path = best_.cost_complexity_pruning_path(X_train, y_train)
          kfold = skm.KFold(5,
                            shuffle=True,
                            random_state=10)
          grid = skm.GridSearchCV(best_,
                                  {'ccp_alpha': ccp_path.ccp_alphas},
                                  refit=True,
                                  cv=kfold,
                                  scoring='neg_mean_squared_error')
          G = grid.fit(X_train, y_train)
```

```
In [123]: best_ = grid.best_estimator_
          np.mean((y_test - best_.predict(X_test))**2)

Out[123]: 4.468305434887279
```

```
In [124]: ax = subplots(figsize=(12,12))[1]
          plot_tree(G.best_estimator_,
                    feature_names=feature_names,
                    ax=ax);
```



Using k-fold (k=5) cross validation to determine optimal complexity (by varying maximum depth):

The test error improves from 5.03 (rounded to 2 decimal places) to 4.33 (rounded to 2 decimal places). The resulting tree has a depth of 6 too.

In [112]:
```
#Cross validation to test for different max depths
#ccp_path = reg.cost_complexity_pruning_path(X_train, y_train)
kfold = skm.KFold(5,
                  shuffle=True,
                  random_state=10)
tree_para = {'max_depth':[2,3,4,5,6,7,8,9,10,11,12]}
grid = skm.GridSearchCV(reg,
                        tree_para,
                        refit=True,
                        cv=kfold,
                        scoring='neg_mean_squared_error')
G = grid.fit(X_train, y_train)
print(G)
```

```
GridSearchCV(cv=KFold(n_splits=5, random_state=10, shuffle=Tru
e),
             estimator=DecisionTreeRegressor(),
             param_grid={'max_depth': [2, 3, 4, 5, 6, 7, 8, 9, 1
0, 11, 12]},
             scoring='neg_mean_squared_error')
```

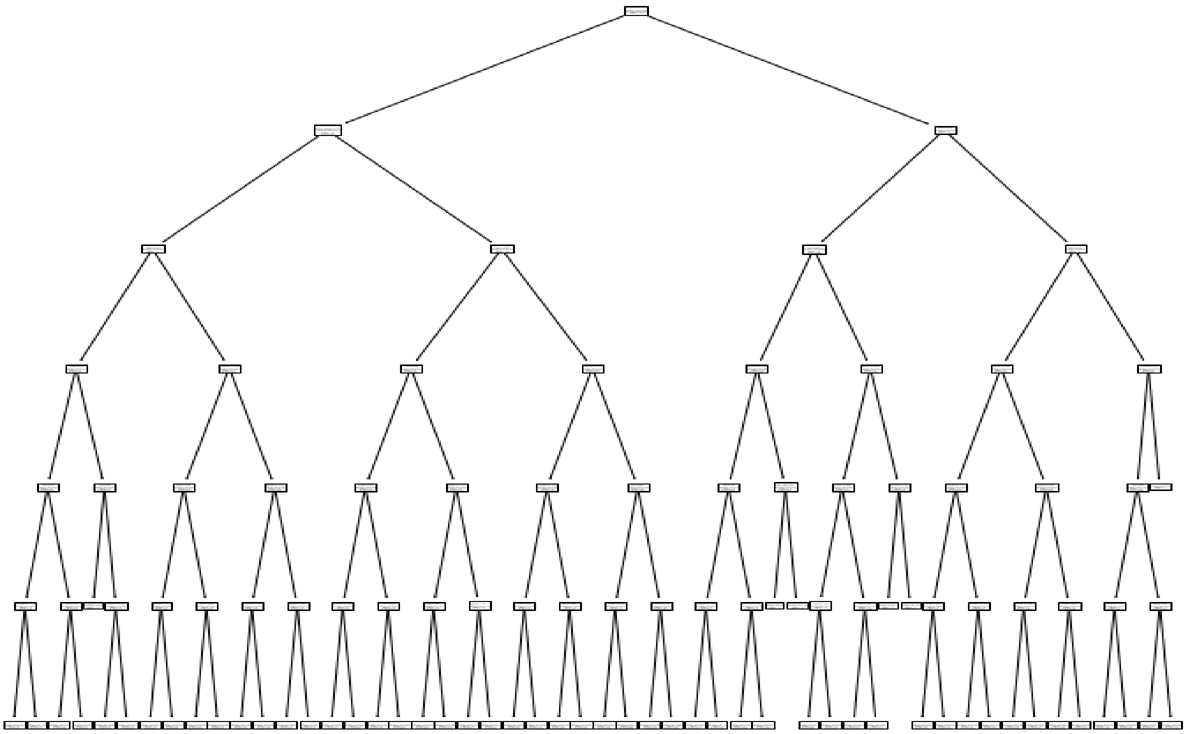
In [113]:
```
best_ = grid.best_estimator_
np.mean((y_test - best_.predict(X_test))**2)
```

Out[113]:
```
4.325673618339898
```

In [114]:
```
print(best_)
```

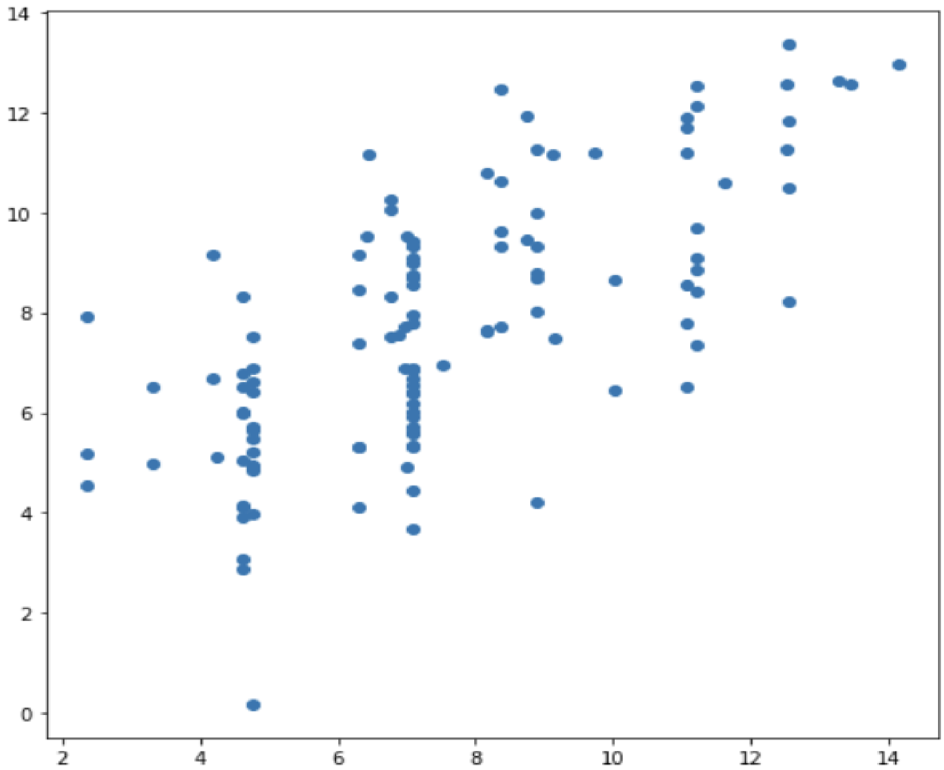
```
DecisionTreeRegressor(max_depth=6)
```

In [115]:
```
ax = subplots(figsize=(12,12))[1]
plot_tree(G.best_estimator_,
          feature_names=feature_names,
          ax=ax);
```



In [116]:
```
ax = subplots(figsize=(8,8))[1]
y_hat = best_.predict(X_test)
ax.scatter(y_hat, y_test)
np.mean((y_test - y_hat)**2)
```

Out[116]: 4.325673618339898



Pruning the above tree leads to a slight increase in the test error.

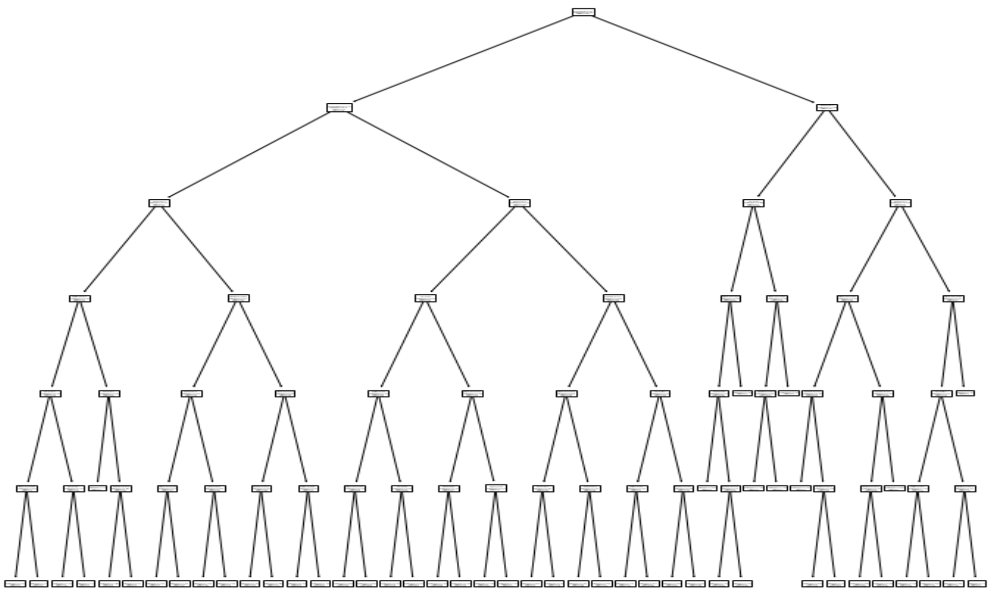
The test error reduces from 4.33 (rounded to 2 decimal places) to 4.40 (rounded to 2 decimal places).

In [117]:
```
#Pruning
ccp_path = best_.cost_complexity_pruning_path(X_train, y_train)
kfold = skm.KFold(5,
                  shuffle=True,
                  random_state=10)
grid = skm.GridSearchCV(best_,
                        {'ccp_alpha': ccp_path.ccp_alphas},
                        refit=True,
                        cv=kfold,
                        scoring='neg_mean_squared_error')
G = grid.fit(X_train, y_train)
```

In [118]:
```
best_ = grid.best_estimator_
np.mean((y_test - best_.predict(X_test))**2)
```

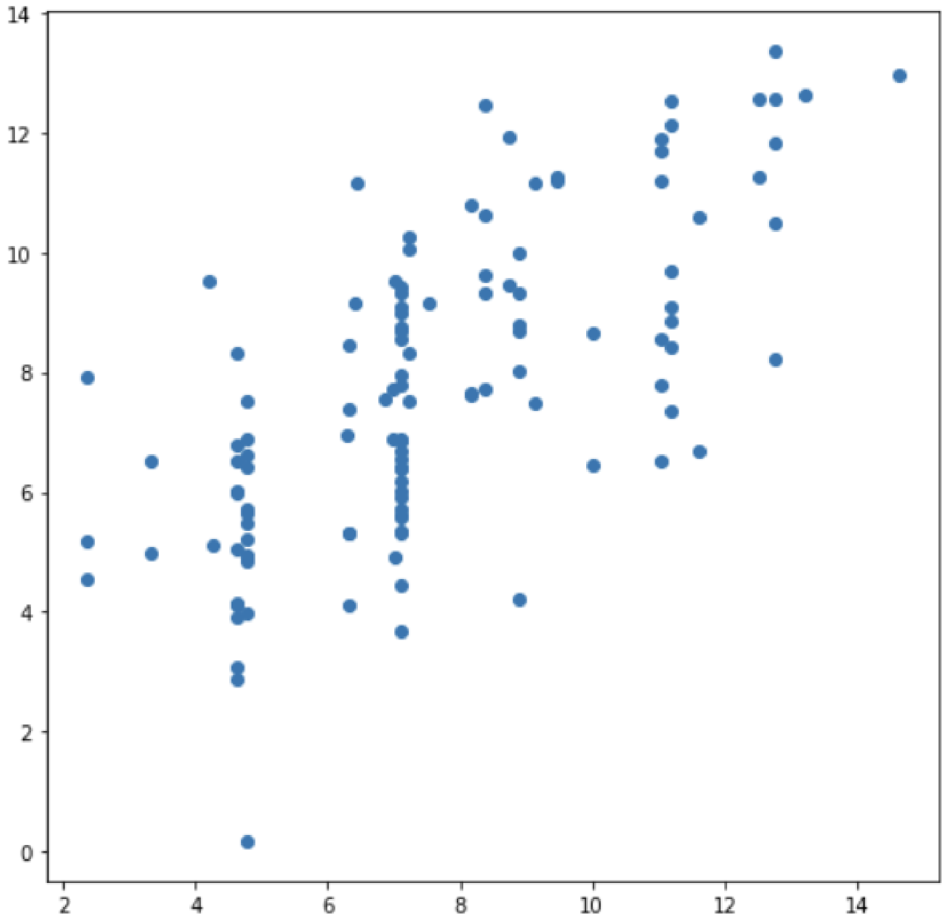
Out[118]: 4.402777712089898

In [119]:
```
ax = subplots(figsize=(12,12))[1]
plot_tree(G.best_estimator_,
          feature_names=feature_names,
          ax=ax);
```



```
In [120]: ax = subplots(figsize=(8,8))[1]
          y_hat = best_.predict(X_test)
          ax.scatter(y_hat, y_test)
          np.mean((y_test - y_hat)**2)
```

```
Out[120]: 4.402777712089898
```



(d)

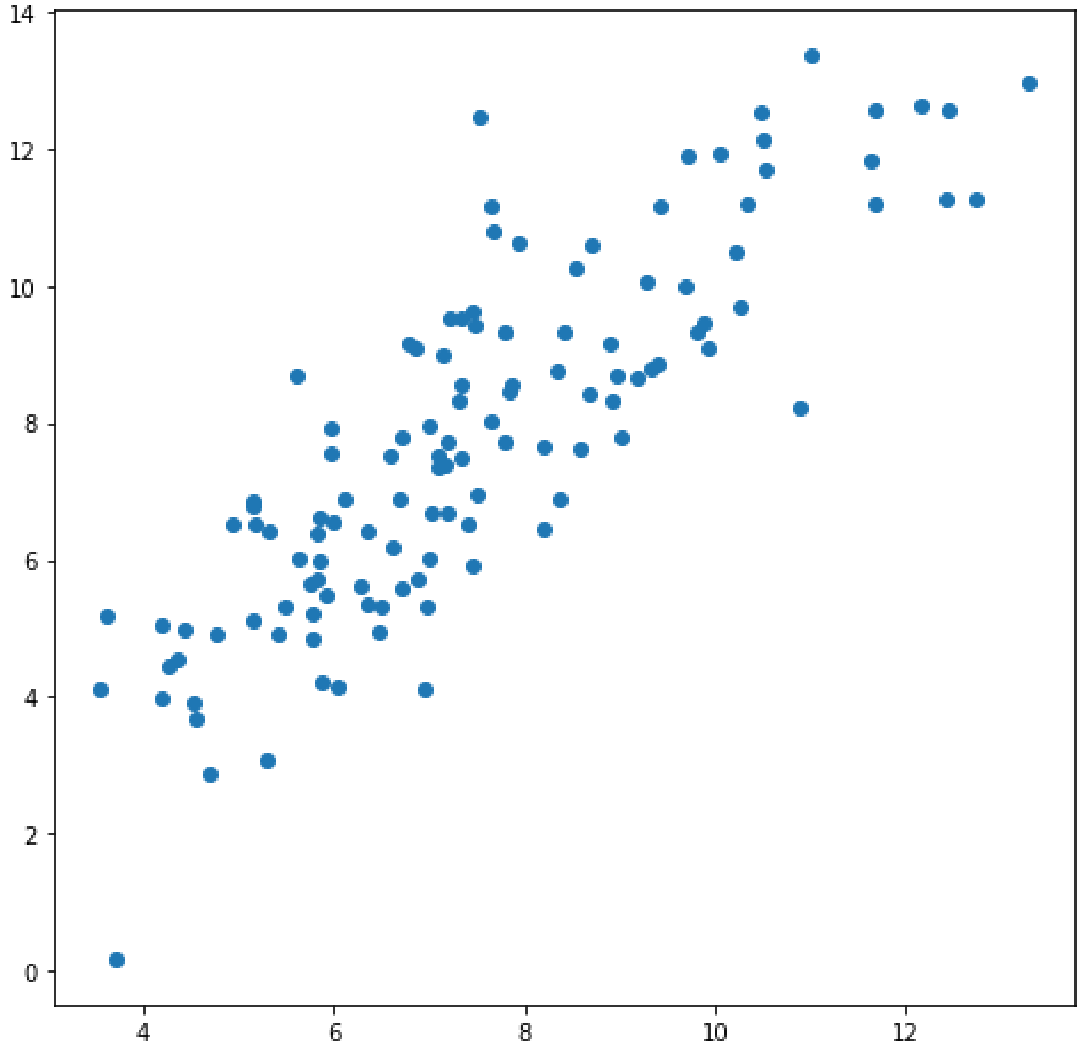
```
In [25]: bag_carseats = RF(max_features=X_train.shape[1], random_state=0)
         bag_carseats.fit(X_train, y_train)
```

```
Out[25]: RandomForestRegressor
         RandomForestRegressor(max_features=11, random_state=0)
```

```
In [26]: ax = subplots(figsize=(8,8))[1]
         y_hat_bag = bag_carseats.predict(X_test)
         ax.scatter(y_hat_bag, y_test)
         np.mean((y_test - y_hat_bag)**2)
```

```
Out[26]: 2.007744519750003
```

Bagging leads to test MSE of 2.01 (rounded to 2 decimal places)



In [27]:
```
#changing number of trees
bag_carseats = RF(max_features=X_train.shape[1],
                  n_estimators=500,
                  random_state=0).fit(X_train, y_train)
y_hat_bag = bag_carseats.predict(X_test)
np.mean((y_test - y_hat_bag)**2)
```
Out[27]: 2.0236328641333428

In [28]:
```
#changing number of trees
bag_carseats = RF(max_features=X_train.shape[1],
                  n_estimators=80,
                  random_state=0).fit(X_train, y_train)
y_hat_bag = bag_carseats.predict(X_test)
np.mean((y_test - y_hat_bag)**2)
```
Out[28]: 2.0299576269531268

In [31]:
```
#changing number of trees
bag_carseats = RF(max_features=X_train.shape[1],
                  n_estimators=120,
                  random_state=0).fit(X_train, y_train)
y_hat_bag = bag_carseats.predict(X_test)
np.mean((y_test - y_hat_bag)**2)
```
Out[31]: 2.0614932343171315

In [34]:
```
feature_imp = pd.DataFrame(
                          {'importance':bag_carseats.feature_importances_},
                          index=feature_names)
feature_imp.sort_values(by='importance', ascending=False)
```

The feature importance is shown below. ‘Price’ being the most important feature, and ‘Urban’ being ‘Yes,’ the least.

Out[34]:

	importance
Price	0.280663
ShelveLoc[Good]	0.220696
Age	0.109453
CompPrice	0.097915
ShelveLoc[Medium]	0.082481
Advertising	0.072299
Income	0.052147
Population	0.041558
Education	0.031427
US[Yes]	0.006551
Urban[Yes]	0.004810

(e)

In [32]:
```
RF_carseats = RF(max_features=6,
                 random_state=0).fit(X_train, y_train)
y_hat_RF = RF_carseats.predict(X_test)
np.mean((y_test - y_hat_RF)**2)
```

Out[32]: 2.010414250999999

In [33]:
```
feature_imp = pd.DataFrame(
                            {'importance':RF_carseats.feature_importances_},
                            index=feature_names)
feature_imp.sort_values(by='importance', ascending=False)
```

Out[33]:

	importance
Price	0.274081
ShelveLoc[Good]	0.185368
Age	0.116413
CompPrice	0.103326
Advertising	0.083759
Income	0.064661
ShelveLoc[Medium]	0.062402
Population	0.054959
Education	0.036506
US[Yes]	0.011148
Urban[Yes]	0.007376

Random forests (similar to bagging but maximum features considered isn’t number of features in the training set) lead to a test MSE of 2.01 (rounded to 2 decimal places).

The feature importance is shown below. ‘Price’ being the most important feature, and ‘Urban’ being ‘Yes,’ the least.

```
In [48]: #changing m/number od features considered at each split
         RF_carseats = RF(max_features=4,
                          random_state=0).fit(X_train, y_train)
         y_hat_RF = RF_carseats.predict(X_test)
         np.mean((y_test - y_hat_RF)**2)

Out[48]: 2.162177176749999
```

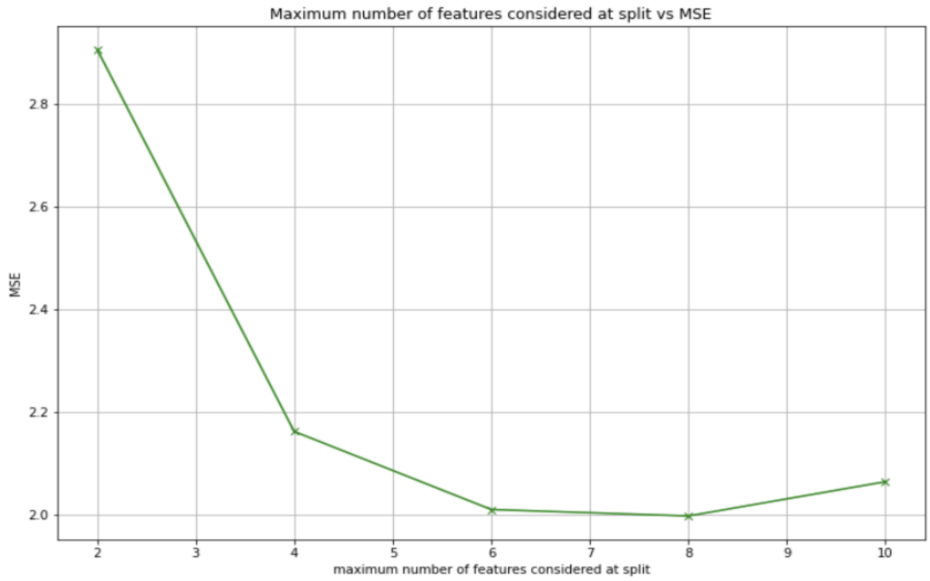
```
In [49]: RF_carseats = RF(max_features=2,
                          random_state=0).fit(X_train, y_train)
         y_hat_RF = RF_carseats.predict(X_test)
         np.mean((y_test - y_hat_RF)**2)

Out[49]: 2.9053707589999993
```

```
In [50]: RF_carseats = RF(max_features=8,
                          random_state=0).fit(X_train, y_train)
         y_hat_RF = RF_carseats.predict(X_test)
         np.mean((y_test - y_hat_RF)**2)

Out[50]: 1.9978001218333337
```

```
In [51]: RF_carseats = RF(max_features=10,
                          random_state=0).fit(X_train, y_train)
         y_hat_RF = RF_carseats.predict(X_test)
         np.mean((y_test - y_hat_RF)**2)

Out[51]: 2.064451973666667
```

Effect of m, the number of features considered at each split on the test error:

```
In [117]: no_of_features=[2,4,6,8,10]
          mse=[2.9053707589999993,2.162177176749999,2.010414250999999,1.9978001218333337,2.064451973666667]
          fig = plt.figure(figsize=(12, 8))
          ax = fig.add_subplot(111)
          plt.plot(no_of_features, mse,marker="x", color='g')
          ax.set_xlabel('maximum number of features considered at split')
          ax.set_ylabel('MSE')
          ax.set_title('Maximum number of features considered at split vs MSE')
          plt.grid()
          plt.show()
```



The test MSE seems to reduce with an increase in the number of features considered at each split to a certain extent (up to 8 in this case)

9.

9

In [37]:
```python
OJ = load_data('OJ')
OJ.head(5)
```

Out[37]:

	Purchase	WeekofPurchase	StoreID	PriceCH	PriceMM	DiscCH	DiscMM	SpecialCH	SpecialMM	LoyalCH	SalePriceMM	SalePriceCH	PriceDiff	Store7	Pc
0	CH	237	1	1.75	1.99	0.00	0.0	0	0	0.500000	1.99	1.75	0.24	No	
1	CH	239	1	1.75	1.99	0.00	0.3	0	1	0.600000	1.69	1.75	-0.06	No	
2	CH	245	1	1.86	2.09	0.17	0.0	0	0	0.680000	2.09	1.69	0.40	No	
3	MM	227	1	1.69	1.69	0.00	0.0	0	0	0.400000	1.69	1.69	0.00	No	
4	CH	228	7	1.69	1.69	0.00	0.0	0	0	0.956535	1.69	1.69	0.00	Yes	

In [38]:
```python
OJ.shape
```

Out[38]: (1070, 18)

In [39]:
```python
test_proportion=(OJ.shape[0]-800)/OJ.shape[0]
print(test_proportion)
```

```
0.2523364485981308
```

(a)

In [40]:
```python
X = OJ.drop('Purchase',axis=1)
X.head()
```

Out[40]:

	WeekofPurchase	StoreID	PriceCH	PriceMM	DiscCH	DiscMM	SpecialCH	SpecialMM	LoyalCH	SalePriceMM	SalePriceCH	PriceDiff	Store7	PctDiscMM	F
0	237	1	1.75	1.99	0.00	0.0	0	0	0.500000	1.99	1.75	0.24	No	0.000000	
1	239	1	1.75	1.99	0.00	0.3	0	1	0.600000	1.69	1.75	-0.06	No	0.150754	

In [41]:
```python
from sklearn import preprocessing
lab_enc = preprocessing.LabelEncoder()
encoded_store7 = lab_enc.fit_transform(X['Store7'])
print(encoded_store7)
```

```
[0 0 0 ... 1 1 0]
```

In [42]:
```python
X['enc_store7']=encoded_store7
X = X.drop('Store7',axis=1)
X.head()
```

Out[42]:

	WeekofPurchase	StoreID	PriceCH	PriceMM	DiscCH	DiscMM	SpecialCH	SpecialMM	L
0	237	1	1.75	1.99	0.00	0.0	0	0	0
1	239	1	1.75	1.99	0.00	0.3	0	1	0
2	245	1	1.86	2.09	0.17	0.0	0	0	0
3	227	1	1.69	1.69	0.00	0.0	0	0	0
4	228	7	1.69	1.69	0.00	0.0	0	0	0

In [43]:
```python
y = OJ['Purchase']
y.head()
```

Out[43]:
```
0    CH
1    CH
2    CH
3    MM
4    CH
Name: Purchase, dtype: object
```

In [44]:
```python
#encoding y
lab_enc = preprocessing.LabelEncoder()
encoded_Y = lab_enc.fit_transform(y)
print(encoded_Y)
```

In [45]:
```
# split the dataset
X_train, X_test, y_train, y_test = train_test_split(
    X, encoded_Y, test_size=test_proportion, random_state=4)
```

(b)

In [46]:
```
#Response is categorical, hence we use a claasification tree
clf1= DTC(criterion='entropy',
      #max_depth=3,
      random_state=0)
clf1.fit(X_train, y_train)
```

Out[46]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [47]:
```
#training accuracy
accuracy_score(y_train, clf1.predict(X_train))
```

Out[47]:
```
0.9925
```

In [48]:
```
print("training error=",100*(1-0.9925),'%')
```

```
training error= 0.7499999999999951 %
```

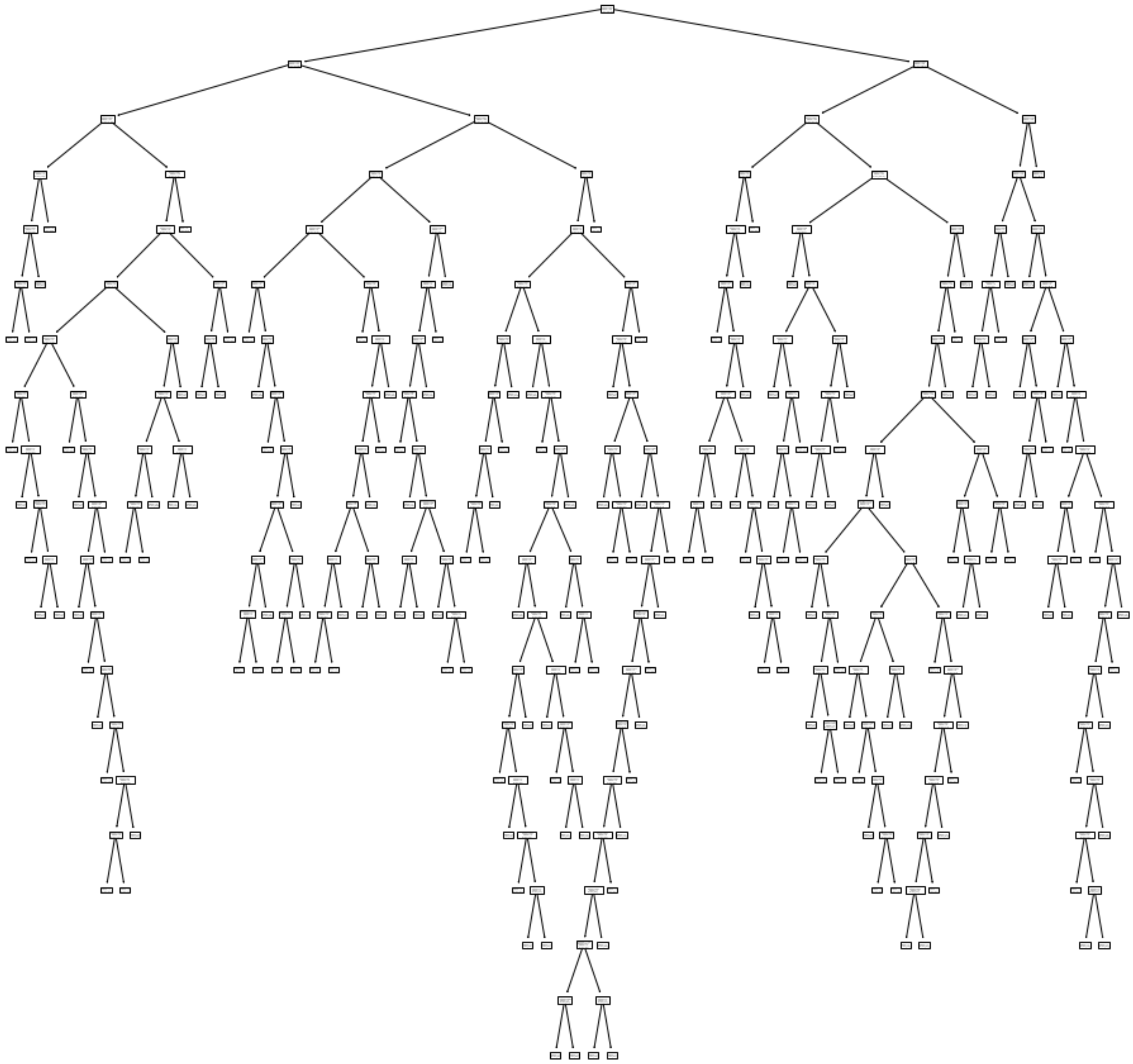
Clf1 has no maximum depth criteria

training error of clf1= 0.7499999999999951 %

(c)

In [49]:
```
feature_names = list(X.columns)
```

In [50]:
```
#plotting
ax = subplots(figsize=(16,16))[1]
plot_tree(clf1,
          feature_names=feature_names,
          ax=ax);
```



In [52]:
```
#number of nodes
print (clf1.tree_.node_count)
```

```
345
```

In [53]:
```
nodedepths=clf1.tree_.compute_node_depths()
print(clf1.tree_.compute_node_depths())
```

```
[ 1  2  3  4  5  6  7  7  6  5  4  5  6  7  8  9  9 10 10 11 11 12 12  8
  9  9 10 10 11 12 12 13 13 14 14 15 15 16 17 17 16 11  7  8  9 10 11 11
 10  9 10 10  8  6  7  8  8  7  5  3  4  5  6  7  7  8  8  9  9 10 11 12
 13 13 12 11 12 13 13 12 10  6  7  7  8  9 10 11 12 13 13 12 11 12 12 10
  9  8  5  6  7  8  9  9 10 10 11 12 12 11 12 12 13 13  8  7  6  4  5  6
  7  8  9 10 11 11 10  9  8  7  8  8  9  9 10 11 12 12 13 14 15 15 16 16
 17 17 18 18 14 13 14 14 15 15 16 16 11 12 12 13 13 10  6  7  8  8  9 10
 10 11 11  9 10 10 11 12 13 14 15 16 17 18 19 20 20 19 20 20 18 17 16 15
 14 13 12 11  7  5  2  3  4  5  6  7  7  8  9 10 11 11 10  9 10 10 11 11
 12 12 13 13  8  6  5  4  5  6  6  7  8  8  9 10 10 11 11  9  7  8  9 10
 10  9  8  5  6  7  8  9 10 11 12 12 13 14 14 15 15 13 11 12 13 14 14 15
 15 16 16 17 17 13 14 14 12 13 13 14 15 16 17 18 18 17 16 15 14 10  9 10
 11 11 12 12 10 11 11  8  7  6  3  4  5  6  7  8  8  7  6  5  6  6  7  8
  8  9 10 10  9  7  8  8  9  9 10 11 12 12 11 10 11 11 12 13 14 15 15 16
 17 17 18 18 16 14 13 12  4]
```

In [54]:
```
print(type(clf1.tree_.compute_node_depths()))
```

```
<class 'numpy.ndarray'>
```

In [55]:
```
terminal_depth=max(nodedepths)
print(terminal_depth)
```

```
20
```

In [56]:
```
#count of nodes with max depth
terminal=[]
for depth in nodedepths:
    if depth==terminal_depth:
        terminal.append(depth)
print(terminal,len(terminal))
```

```
[20, 20, 20, 20] 4
```

In [57]:
```
clf2 = DTC(criterion='entropy',
      max_depth=3,
      random_state=0)
clf2.fit(X_train, y_train)
```

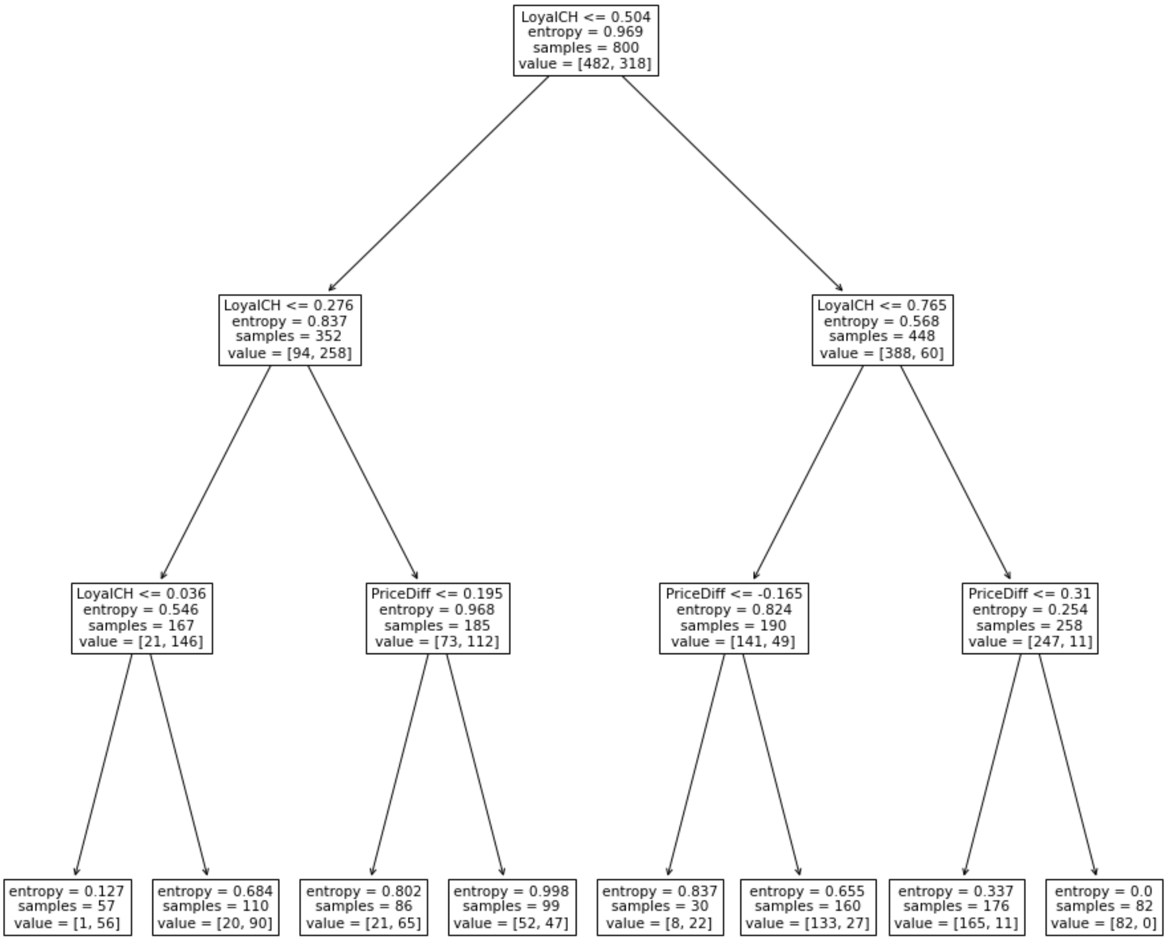
Out[57]:
```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy', max_depth=3, random_state=0)
```

In [58]:
```
#training accuracy
accuracy_score(y_train, clf2.predict(X_train))
```

Out[58]: 0.83125

In [59]:
```
#plotting
ax = subplots(figsize=(16,16))[1]
plot_tree(clf2,
          feature_names=feature_names,
          ax=ax);
```

Creating clf2 with maximum depth 3 since clf1 had depth 20 to allow manual counting of terminal nodes (nodes with no children).



Above is the decision tree obtained specifying maximum depth as 3 (to make the tree plot more legible, as higher depths led to illegible text within the tree). The root decision node is corresponding to the LoyalCH being used as the feature with value below 0.504 as the decision criteria, implying LoyalCH to the the most important parameter, post which, if LoyalCH is below 0.504, it’s checked if LoyalCH is less than 0.276. If yes, we check if LoyalCH is less than 0.036 to make a prediction. If not, we check the PriceDiff to make a prediction. If the value of LoyalCH was above 0.504, we check if the value of LoyalCH is above 0.765. Then based on the response, we consider the PriceDiff to arrive at a prediction.

The tree created using clf2 has 8 terminal nodes.

(d)

```
In [51]: print(export_text(clf1,
                feature_names=feature_names,
                show_weights=True))
```

```
|--- LoyalCH <= 0.50
|   |--- LoyalCH <= 0.28
|   |   |--- LoyalCH <= 0.04
|   |   |   |--- StoreID <= 2.50
|   |   |   |   |--- LoyalCH <= 0.00
|   |   |   |   |   |--- LoyalCH <= 0.00
|   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |--- LoyalCH >  0.00
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- LoyalCH >  0.00
|   |   |   |   |   |--- weights: [0.00, 7.00] class: 1
|   |   |   |--- StoreID >  2.50
|   |   |   |   |--- weights: [0.00, 47.00] class: 1
|   |   |--- LoyalCH >  0.04
|   |   |   |--- WeekofPurchase <= 273.50
|   |   |   |   |--- WeekofPurchase <= 263.50
|   |   |   |   |   |--- STORE <= 1.50
|   |   |   |   |   |   |--- ListPriceDiff <= 0.18
|   |   |   |   |   |   |   |--- LoyalCH <= 0.15
|   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- LoyalCH >  0.15
|   |   |   |   |   |   |   |   |--- WeekofPurchase <= 228.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- WeekofPurchase >  228.50
|   |   |   |   |   |   |   |   |   |--- StoreID <= 4.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- StoreID >  4.00
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.16
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.16
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- ListPriceDiff >  0.18
|   |   |   |   |   |   |   |--- ListPriceDiff <= 0.23
|   |   |   |   |   |   |   |   |--- weights: [0.00, 9.00] class: 1
|   |   |   |   |   |   |   |--- ListPriceDiff >  0.23
|   |   |   |   |   |   |   |   |--- LoyalCH <= 0.08
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |   |--- LoyalCH >  0.08
```

```
|   |   |   |   |   |   |   |   |   |--- WeekofPurchase <= 256.50
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.09
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.09
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 6
|   |   |   |   |   |   |   |   |   |--- WeekofPurchase >  256.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- STORE >  1.50
|   |   |   |   |   |   |--- StoreID <= 3.50
|   |   |   |   |   |   |   |--- SalePriceMM <= 2.26
|   |   |   |   |   |   |   |   |--- LoyalCH <= 0.07
|   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.06
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 6.00] class: 1
|   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.06
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- LoyalCH >  0.07
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 40.00] class: 1
|   |   |   |   |   |   |   |--- SalePriceMM >  2.26
|   |   |   |   |   |   |   |   |--- WeekofPurchase <= 257.00
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- WeekofPurchase >  257.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- StoreID >  3.50
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- WeekofPurchase >  263.50
|   |   |   |   |   |--- LoyalCH <= 0.27
|   |   |   |   |   |   |--- StoreID <= 1.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- StoreID >  1.50
|   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |--- LoyalCH >  0.27
|   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |--- WeekofPurchase >  273.50
|   |   |   |   |--- weights: [0.00, 15.00] class: 1
|   |--- LoyalCH >  0.28
|   |   |--- PriceDiff <= 0.20
|   |   |   |--- SpecialCH <= 0.50
|   |   |   |   |--- SalePriceMM <= 1.74
|   |   |   |   |   |--- LoyalCH <= 0.28
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- LoyalCH >  0.28
|   |   |   |   |   |   |--- StoreID <= 1.50
|   |   |   |   |   |   |   |--- weights: [0.00, 21.00] class: 1
|   |   |   |   |   |   |--- StoreID >  1.50
|   |   |   |   |   |   |   |--- PriceDiff <= -0.24
|   |   |   |   |   |   |   |   |--- weights: [0.00, 10.00] class: 1
|   |   |   |   |   |   |   |--- PriceDiff >  -0.24
```

```
|   |   |   |   |   |   |   |   |--- STORE <= 3.50
|   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.47
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.33
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.33
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 9.00] class: 1
|   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.47
|   |   |   |   |   |   |   |   |   |   |--- PriceMM <= 1.84
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- PriceMM >  1.84
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- STORE >  3.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- SalePriceMM >  1.74
|   |   |   |   |   |--- LoyalCH <= 0.39
|   |   |   |   |   |   |--- weights: [0.00, 6.00] class: 1
|   |   |   |   |   |--- LoyalCH >  0.39
|   |   |   |   |   |   |--- WeekofPurchase <= 263.50
|   |   |   |   |   |   |   |--- SalePriceMM <= 1.84
|   |   |   |   |   |   |   |   |--- LoyalCH <= 0.49
|   |   |   |   |   |   |   |   |   |--- STORE <= 3.50
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.44
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.44
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- STORE >  3.50
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.44
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.44
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- LoyalCH >  0.49
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- SalePriceMM >  1.84
|   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |--- WeekofPurchase >  263.50
|   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |--- SpecialCH >  0.50
|   |   |   |   |--- PctDiscCH <= 0.11
|   |   |   |   |   |--- SpecialMM <= 0.50
|   |   |   |   |   |   |--- STORE <= 3.50
|   |   |   |   |   |   |   |--- LoyalCH <= 0.37
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- LoyalCH >  0.37
|   |   |   |   |   |   |   |   |--- LoyalCH <= 0.39
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- LoyalCH >  0.39
|   |   |   |   |   |   |   |   |   |--- PctDiscMM <= 0.20
```

```
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.47
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.47
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |--- PctDiscMM >  0.20
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.45
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.45
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |--- STORE >  3.50
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- SpecialMM >  0.50
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- PctDiscCH >  0.11
|   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |--- PriceDiff >  0.20
|   |   |   |--- PriceDiff <= 0.49
|   |   |   |   |--- LoyalCH <= 0.45
|   |   |   |   |   |--- ListPriceDiff <= 0.26
|   |   |   |   |   |   |--- PriceDiff <= 0.23
|   |   |   |   |   |   |   |--- StoreID <= 3.00
|   |   |   |   |   |   |   |   |--- StoreID <= 1.50
|   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.36
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.36
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |--- StoreID >  1.50
|   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- StoreID >  3.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |--- PriceDiff >  0.23
|   |   |   |   |   |   |   |--- weights: [0.00, 11.00] class: 1
|   |   |   |   |   |--- ListPriceDiff >  0.26
|   |   |   |   |   |   |--- WeekofPurchase <= 234.00
|   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- WeekofPurchase >  234.00
|   |   |   |   |   |   |   |--- WeekofPurchase <= 255.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |--- WeekofPurchase >  255.50
|   |   |   |   |   |   |   |   |--- LoyalCH <= 0.41
|   |   |   |   |   |   |   |   |   |--- PriceCH <= 1.88
|   |   |   |   |   |   |   |   |   |   |--- SalePriceMM <= 2.16
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- SalePriceMM >  2.16
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 7
|   |   |   |   |   |   |   |   |   |--- PriceCH >  1.88
|   |   |   |   |   |   |   |   |   |   |--- StoreID <= 2.50
```

```
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- StoreID >  2.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |--- LoyalCH >  0.41
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |--- LoyalCH >  0.45
|   |   |   |   |   |--- PriceDiff <= 0.38
|   |   |   |   |   |   |--- WeekofPurchase <= 234.00
|   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |--- WeekofPurchase >  234.00
|   |   |   |   |   |   |   |--- PriceDiff <= 0.22
|   |   |   |   |   |   |   |   |--- PctDiscCH <= 0.03
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |   |--- PctDiscCH >  0.03
|   |   |   |   |   |   |   |   |   |--- WeekofPurchase <= 271.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- WeekofPurchase >  271.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 1.00] class: 0
|   |   |   |   |   |   |   |--- PriceDiff >  0.22
|   |   |   |   |   |   |   |   |--- PriceCH <= 1.77
|   |   |   |   |   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- PriceCH >  1.77
|   |   |   |   |   |   |   |   |   |--- WeekofPurchase <= 267.50
|   |   |   |   |   |   |   |   |   |   |--- WeekofPurchase <= 262.00
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 9
|   |   |   |   |   |   |   |   |   |   |--- WeekofPurchase >  262.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- WeekofPurchase >  267.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |--- PriceDiff >  0.38
|   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |--- PriceDiff >  0.49
|   |   |   |   |--- weights: [5.00, 0.00] class: 0
|--- LoyalCH >  0.50
|   |--- LoyalCH <= 0.76
|   |   |--- PriceDiff <= -0.16
|   |   |   |--- STORE <= 3.50
|   |   |   |   |--- WeekofPurchase <= 276.50
|   |   |   |   |   |--- LoyalCH <= 0.59
|   |   |   |   |   |   |--- weights: [0.00, 6.00] class: 1
|   |   |   |   |   |--- LoyalCH >  0.59
|   |   |   |   |   |   |--- LoyalCH <= 0.69
|   |   |   |   |   |   |   |--- WeekofPurchase <= 271.50
|   |   |   |   |   |   |   |   |--- PctDiscMM <= 0.15
|   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.64
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.64
```

```
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- PctDiscMM >  0.15
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |--- WeekofPurchase >  271.50
|   |   |   |   |   |   |   |   |--- WeekofPurchase <= 274.50
|   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- WeekofPurchase >  274.50
|   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.64
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.64
|   |   |   |   |   |   |   |   |   |   |--- PriceMM <= 2.16
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- PriceMM >  2.16
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |--- LoyalCH >  0.69
|   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |--- WeekofPurchase >  276.50
|   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |--- STORE >  3.50
|   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |--- PriceDiff >  -0.16
|   |   |   |--- ListPriceDiff <= 0.13
|   |   |   |   |--- WeekofPurchase <= 228.50
|   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |--- WeekofPurchase >  228.50
|   |   |   |   |   |--- StoreID <= 3.50
|   |   |   |   |   |   |--- WeekofPurchase <= 265.00
|   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |--- WeekofPurchase >  265.00
|   |   |   |   |   |   |   |--- SpecialMM <= 0.50
|   |   |   |   |   |   |   |   |--- StoreID <= 2.00
|   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- StoreID >  2.00
|   |   |   |   |   |   |   |   |   |--- DiscCH <= 0.15
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- DiscCH >  0.15
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- SpecialMM >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |--- StoreID >  3.50
|   |   |   |   |   |   |--- LoyalCH <= 0.62
|   |   |   |   |   |   |   |--- WeekofPurchase <= 261.00
|   |   |   |   |   |   |   |   |--- WeekofPurchase <= 230.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- WeekofPurchase >  230.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- WeekofPurchase >  261.00
```

```
|   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- LoyalCH >  0.62
|   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |--- ListPriceDiff >  0.13
|   |   |   |   |--- PriceDiff <= 0.36
|   |   |   |   |   |--- LoyalCH <= 0.76
|   |   |   |   |   |   |--- LoyalCH <= 0.74
|   |   |   |   |   |   |   |--- LoyalCH <= 0.68
|   |   |   |   |   |   |   |   |--- WeekofPurchase <= 273.50
|   |   |   |   |   |   |   |   |   |--- ListPriceDiff <= 0.23
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.55
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.55
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |--- ListPriceDiff >  0.23
|   |   |   |   |   |   |   |   |   |   |--- StoreID <= 2.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 6
|   |   |   |   |   |   |   |   |   |   |--- StoreID >  2.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 7
|   |   |   |   |   |   |   |   |--- WeekofPurchase >  273.50
|   |   |   |   |   |   |   |   |   |--- weights: [8.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- LoyalCH >  0.68
|   |   |   |   |   |   |   |   |--- LoyalCH <= 0.71
|   |   |   |   |   |   |   |   |   |--- StoreID <= 5.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |   |   |--- StoreID >  5.50
|   |   |   |   |   |   |   |   |   |   |--- ListPriceDiff <= 0.26
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- ListPriceDiff >  0.26
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- LoyalCH >  0.71
|   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.74
|   |   |   |   |   |   |   |   |   |   |--- weights: [8.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.74
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- LoyalCH >  0.74
|   |   |   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |   |--- LoyalCH >  0.76
|   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- PriceDiff >  0.36
|   |   |   |   |   |--- weights: [20.00, 0.00] class: 0
|   |--- LoyalCH >  0.76
|   |   |--- PriceDiff <= 0.31
|   |   |   |--- PriceDiff <= -0.39
|   |   |   |   |--- STORE <= 1.50
|   |   |   |   |   |--- WeekofPurchase <= 275.50
|   |   |   |   |   |   |--- LoyalCH <= 1.00
```

```
|   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |--- LoyalCH >  1.00
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- WeekofPurchase >  275.50
|   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |--- STORE >  1.50
|   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |--- PriceDiff >  -0.39
|   |   |   |   |--- STORE <= 1.50
|   |   |   |   |   |--- weights: [71.00, 0.00] class: 0
|   |   |   |   |--- STORE >  1.50
|   |   |   |   |   |--- ListPriceDiff <= 0.23
|   |   |   |   |   |   |--- PriceDiff <= 0.20
|   |   |   |   |   |   |   |--- weights: [30.00, 0.00] class: 0
|   |   |   |   |   |   |--- PriceDiff >  0.20
|   |   |   |   |   |   |   |--- LoyalCH <= 0.95
|   |   |   |   |   |   |   |   |--- LoyalCH <= 0.94
|   |   |   |   |   |   |   |   |   |--- weights: [8.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- LoyalCH >  0.94
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- LoyalCH >  0.95
|   |   |   |   |   |   |   |   |--- weights: [11.00, 0.00] class: 0
|   |   |   |   |   |--- ListPriceDiff >  0.23
|   |   |   |   |   |   |--- PriceDiff <= -0.07
|   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |--- PriceDiff >  -0.07
|   |   |   |   |   |   |   |--- WeekofPurchase <= 236.50
|   |   |   |   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- WeekofPurchase >  236.50
|   |   |   |   |   |   |   |   |--- WeekofPurchase <= 241.50
|   |   |   |   |   |   |   |   |   |--- STORE <= 2.50
|   |   |   |   |   |   |   |   |   |   |--- WeekofPurchase <= 240.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- WeekofPurchase >  240.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- STORE >  2.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |--- WeekofPurchase >  241.50
|   |   |   |   |   |   |   |   |   |--- WeekofPurchase <= 247.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- WeekofPurchase >  247.50
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH <= 0.99
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 7
|   |   |   |   |   |   |   |   |   |   |--- LoyalCH >  0.99
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |--- PriceDiff >  0.31
|   |   |   |--- weights: [82.00, 0.00] class: 0
```

(d)

In [60]:
```
print(export_text(clf2,
      feature_names=feature_names,
      show_weights=True))
```

```
|--- LoyalCH <= 0.50
|   |--- LoyalCH <= 0.28
|   |   |--- LoyalCH <= 0.04
|   |   |   |--- weights: [1.00, 56.00] class: 1
|   |   |--- LoyalCH >  0.04
|   |   |   |--- weights: [20.00, 90.00] class: 1
|   |--- LoyalCH >  0.28
|   |   |--- PriceDiff <= 0.20
|   |   |   |--- weights: [21.00, 65.00] class: 1
|   |   |--- PriceDiff >  0.20
|   |   |   |--- weights: [52.00, 47.00] class: 0
|--- LoyalCH >  0.50
|   |--- LoyalCH <= 0.76
|   |   |--- PriceDiff <= -0.16
|   |   |   |--- weights: [8.00, 22.00] class: 1
|   |   |--- PriceDiff >  -0.16
|   |   |   |--- weights: [133.00, 27.00] class: 0
|   |--- LoyalCH >  0.76
|   |   |--- PriceDiff <= 0.31
|   |   |   |--- weights: [165.00, 11.00] class: 0
|   |   |--- PriceDiff >  0.31
|   |   |   |--- weights: [82.00, 0.00] class: 0
```

In [61]:
```
print (clf2.tree_.node_count)
```

```
15
```

Interpretation of the terminal node making a prediction of class 0 (weights 82,0):

This prediction corresponds to datapoints which would-

1. Have LoyalCH>0.5
2. Also, LoyalCH>0.76 (next criteria)
3. Lastly, PriceDiff>0.31

(e)

In [62]:
```
#test accuracy of tree 1
accuracy_score(y_test, clf1.predict(X_test))
```
Out[62]: 0.7851851851851852

In [63]:
```
#test accuracy of tree 2
accuracy_score(y_test, clf2.predict(X_test))
```
Out[63]: 0.8481481481481481

In [64]:
```
#confusin matrix for test data corresponding to tree 1
print(accuracy_score(y_test,
      clf1.predict(X_test)))
confusion = confusion_table(clf1.predict(X_test),
      y_test)
confusion
```
0.7851851851851852

Out[64]:

Truth	0	1
Predicted		
0	143	30
1	28	69

In [65]:
```
#confusin matrix for test data corresponding to tree 2
print(accuracy_score(y_test,
      clf2.predict(X_test)))
confusion = confusion_table(clf2.predict(X_test),
      y_test)
confusion
```
0.8481481481481481

Out[65]:

Truth	0	1
Predicted		
0	155	25
1	16	74

Test accuracy corresponding to:

Clf1 is 78.52 % (Rounded to two decimal places)

Clf2 is 84.81 % (Rounded to two decimal places)

(f)

In [66]:
```
#Cross validation to test for different max depths
#ccp_path = reg.cost_complexity_pruning_path(X_train, y_train)
kfold = skm.KFold(5,
                  shuffle=True,
                  random_state=10)
tree_para = {'max_depth':[3,5,6,7,8,9,11,12,14,16,18]}
grid = skm.GridSearchCV(clf1,
                        tree_para,
                        refit=True,
                        cv=kfold,
                        scoring='neg_mean_squared_error')
G = grid.fit(X_train, y_train)
print(G)
```

```
GridSearchCV(cv=KFold(n_splits=5, random_state=10, shuffle=True),
             estimator=DecisionTreeClassifier(criterion='entropy',
                                              random_state=0),
             param_grid={'max_depth': [3, 5, 6, 7, 8, 9, 11, 12, 14, 16,
18]},
             scoring='neg_mean_squared_error')
```

In [70]:
```
print(accuracy_score(y_test, best_.predict(X_test)))
confusion = confusion_table(best_.predict(X_test),
      y_test)
confusion
```

```
0.8222222222222222
```

Out[70]:

Truth	0	1
Predicted		
0	152	29
1	19	70

(g)

In [77]:
```
res=grid.cv_results_
```

In [78]:
```
print(res)
```

```
{'mean_fit_time': array([0.00347443, 0.00686069, 0.00499115, 0.00647893, 0.00935354,
       0.00314651, 0.00641084, 0.00616207, 0.00640774, 0.01299024,
       0.00790968]), 'std_fit_time': array([0.00304182, 0.00648788, 0.00317198, 0.00794308, 0.00765318,
       0.00629301, 0.00785403, 0.00755115, 0.00785353, 0.00893738,
       0.00679341]), 'mean_score_time': array([0.0034184 , 0.0020474 , 0.        , 0.        , 0.        ,
       0.00622077, 0.00329313, 0.        , 0.        , 0.        ,
       0.        ]), 'std_score_time': array([0.00423801, 0.00310602, 0.        , 0.        , 0.        ,
       0.00725411, 0.00658627, 0.        , 0.        , 0.        ,
       0.        ]), 'param_max_depth': masked_array(data=[3, 5, 6, 7, 8, 9, 11, 12, 14, 16, 18],
             mask=[False, False, False, False, False, False, False, False,
                   False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'max_depth': 3}, {'max_depth': 5}, {'max_depth': 6}, {'max_depth': 7}, {'max_depth': 8},
{'max_depth': 9}, {'max_depth': 11}, {'max_depth': 12}, {'max_depth': 14}, {'max_depth': 16}, {'max_depth': 18}], 'split0_test_
score': array([-0.16875, -0.16875, -0.1625 , -0.2    , -0.20625, -0.2125 ,
       -0.20625, -0.2125 , -0.2125 , -0.20625, -0.2125 ]), 'split1_test_score': array([-0.21875, -0.24375, -0.2375 , -0.24375,
-0.2375 , -0.225  ,
       -0.23125, -0.24375, -0.2375 , -0.26875, -0.24375]), 'split2_test_score': array([-0.225  , -0.2    , -0.175  , -0.2    ,
-0.18125, -0.2    ,
       -0.21875, -0.2125 , -0.2125 , -0.2125 , -0.2125 ]), 'split3_test_score': array([-0.175  , -0.16875, -0.1875 , -0.18125,
-0.16875, -0.2    ,
       -0.2125 , -0.21875, -0.225  , -0.2125 , -0.2125 ]), 'split4_test_score': array([-0.24375, -0.175  , -0.18125, -0.18125,
-0.1875 , -0.18125,
       -0.1875 , -0.19375, -0.15   , -0.1625 , -0.1625 ]), 'mean_test_score': array([-0.20625, -0.19125, -0.18875, -0.20125, -
0.19625, -0.20375,
       -0.21125, -0.21625, -0.2075 , -0.2125 , -0.20875]), 'std_test_score': array([0.0293151 , 0.02866836, 0.02573908, 0.02284
458, 0.02391391,
       0.01457738, 0.0144698 , 0.01610512, 0.03020761, 0.03377314,
       0.02610077]), 'rank_test_score': array([ 6,  2,  1,  4,  3,  5,  9, 11,  7, 10,  8])}
```

In [79]:
```
type(res)
```

Out[79]: dict

In [109]:
```
params=res['params']
print(params)
```

```
[{'max_depth': 3}, {'max_depth': 5}, {'max_depth': 6}, {'max_depth': 7}, {'max_depth': 8}, {'max_depth': 9}, {'max_depth': 11},
{'max_depth': 12}, {'max_depth': 14}, {'max_depth': 16}, {'max_depth': 18}]
```

In [114]:
```
std_test_score=res['std_test_score']
print(std_test_score)
```

```
[0.0293151  0.02866836 0.02573908 0.02284458 0.02391391 0.01457738
 0.0144698  0.01610512 0.03020761 0.03377314 0.02610077]
```

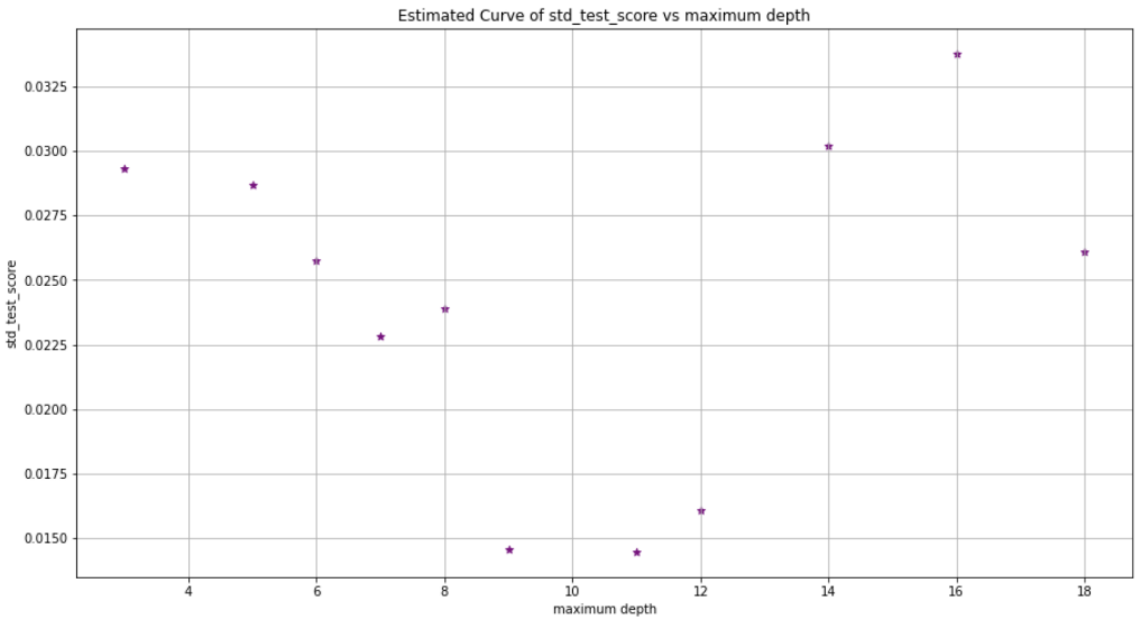
In [115]:
```
max_depth=res['param_max_depth']
print(max_depth)
```

```
[3 5 6 7 8 9 11 12 14 16 18]
```

In [116]:
```
y=res['mean_test_score']
print(y)
```

```
[-0.20625 -0.19125 -0.18875 -0.20125 -0.19625 -0.20375 -0.21125 -0.21625
 -0.2075  -0.2125  -0.20875]
```

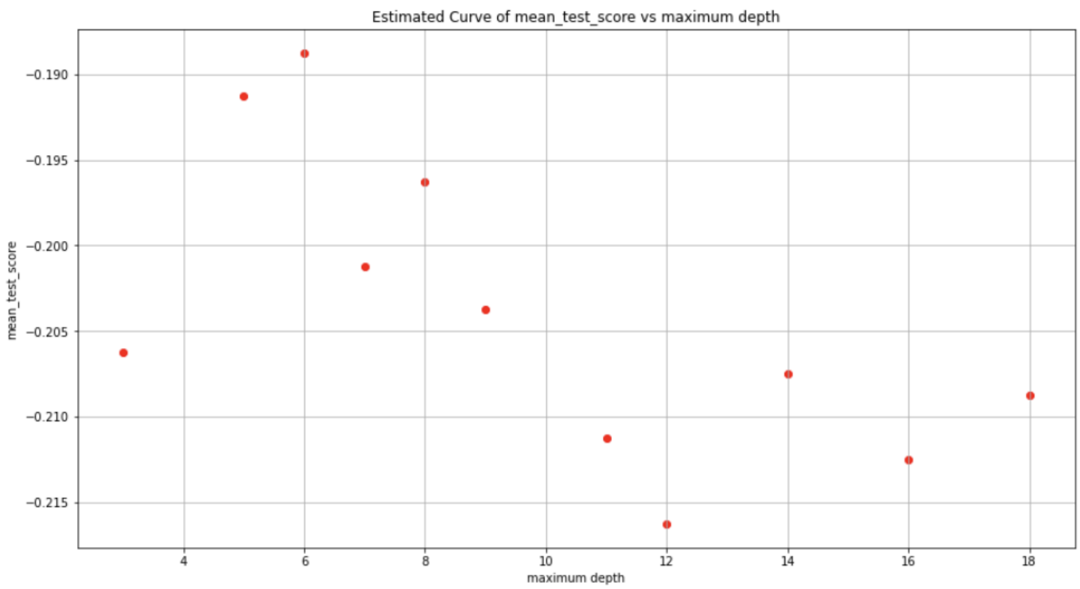
In [119]:
```
fig = plt.figure(figsize=(15, 8))
ax = fig.add_subplot(111)
plt.scatter(max_depth, std_test_score, marker="*", color='purple')
ax.set_xlabel('maximum depth')
ax.set_ylabel('std_test_score')
ax.set_title('Estimated Curve of std_test_score vs maximum depth')
plt.grid()
plt.show()
```



Using std_test_score as the metric we arrive at the plot above

In [120]:

```
fig = plt.figure(figsize=(15, 8))
ax = fig.add_subplot(111)
plt.scatter(max_depth, y, marker="o", color='r')
ax.set_xlabel('maximum depth')
ax.set_ylabel('mean_test_score')
ax.set_title('Estimated Curve of mean_test_score vs maximum depth')
plt.grid()
plt.show()
```



Using mean_test_score as the metric we arrive at the plot above

(h)

```
In [68]: print(best_)

DecisionTreeClassifier(criterion='entropy', max_depth=6, random_state=0)
```

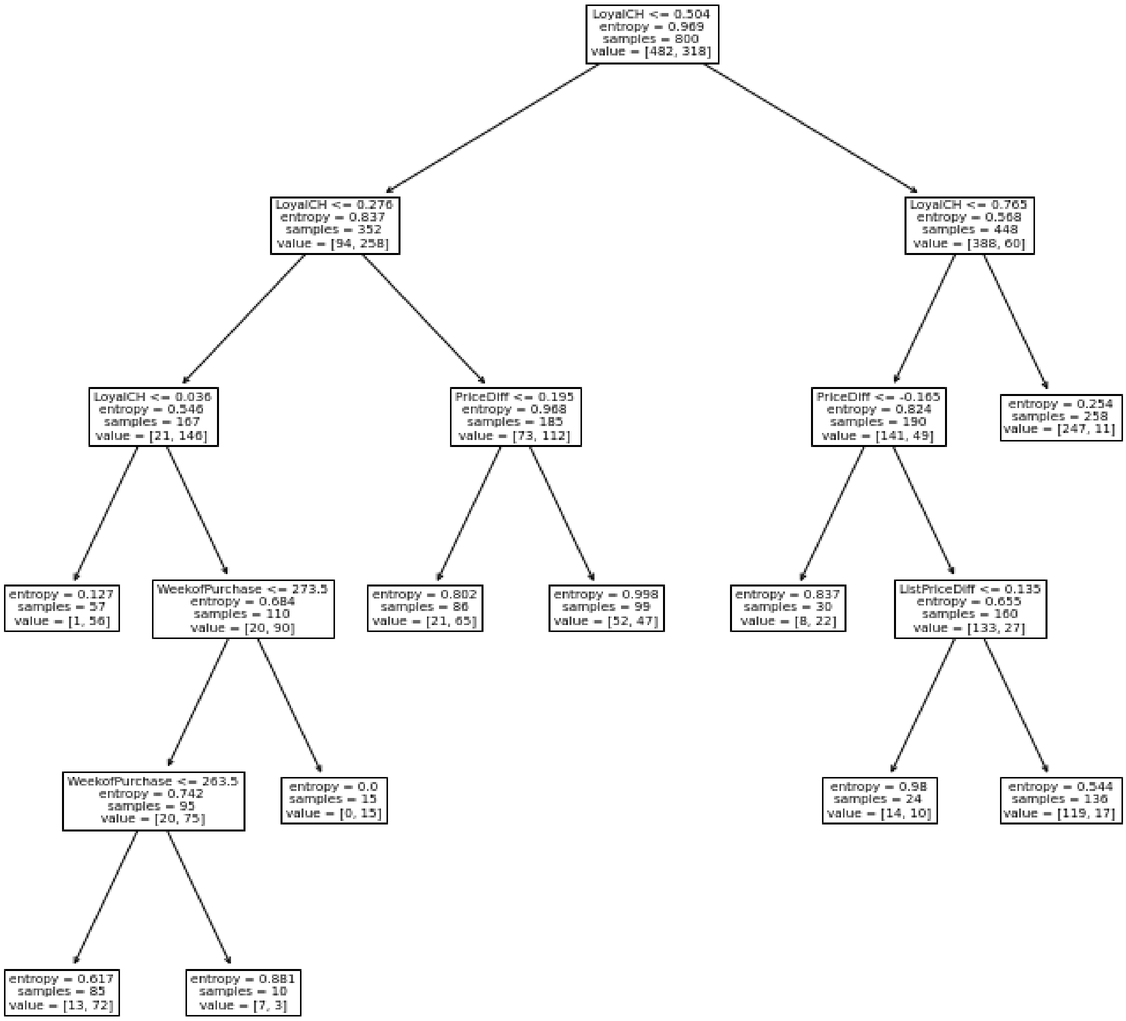
Using k-fold (k=5) cross validation and varying maximum depth, we arrive at maximum depth=6 as the optimal tree size.

(i)

```
In [90]: ccp_path = clf1.cost_complexity_pruning_path(X_train, y_train)
         kfold = skm.KFold(10,
                           random_state=1,
                           shuffle=True)
```

```
In [91]: grid = skm.GridSearchCV(clf1,
                                 {'ccp_alpha': ccp_path.ccp_alphas},
                                 refit=True,
                                   cv=kfold,
                                 scoring='accuracy')
         grid.fit(X_train, y_train)
         grid.best_score_

Out[91]: 0.8099999999999999
```

```
In [92]: ax = subplots(figsize=(12, 12))[1]
         best_ = grid.best_estimator_
         plot_tree(best_,
                   feature_names=feature_names,
                   ax=ax);
```


LoyalCH <= 0.504
entropy = 0.969
samples = 800
value = [482, 318]
LoyalCH <= 0.276
entropy = 0.837
samples = 352
value = [94, 258]
LoyalCH <= 0.765
entropy = 0.568
samples = 448
value = [388, 60]
LoyalCH <= 0.036
entropy = 0.546
samples = 167
value = [21, 146]
PriceDiff <= 0.195
entropy = 0.968
samples = 185
value = [73, 112]
PriceDiff <= -0.165
entropy = 0.824
samples = 190
value = [141, 49]
entropy = 0.254
samples = 258
value = [247, 11]
entropy = 0.127
samples = 57
value = [1, 56]
WeekofPurchase <= 273.5
entropy = 0.684
samples = 110
value = [20, 90]
entropy = 0.802
samples = 86
value = [21, 65]
entropy = 0.998
samples = 99
value = [52, 47]
entropy = 0.837
samples = 30
value = [8, 22]
ListPriceDiff <= 0.135
entropy = 0.655
samples = 160
value = [133, 27]
WeekofPurchase <= 263.5
entropy = 0.742
samples = 95
value = [20, 75]
entropy = 0.0
samples = 15
value = [0, 15]
entropy = 0.98
samples = 24
value = [14, 10]
entropy = 0.544
samples = 136
value = [119, 17]
entropy = 0.617
samples = 85
value = [13, 72]
entropy = 0.881
samples = 10
value = [7, 3]

In [93]:
```
print(accuracy_score(y_test,
      best_.predict(X_test)))
confusion = confusion_table(best_.predict(X_test),
      y_test)
confusion
```

0.8407407407407408

Out[93]:

Truth	0	1
Predicted		
0	155	27
1	16	72

In [94]:
```
print(best_)
```

DecisionTreeClassifier(ccp_alpha=0.007791407710224338, criterion='entropy',
random_state=0)

(j)

In [86]:
```
#unpruned tree training accuracy and error
print(accuracy_score(y_train,
      clf1.predict(X_train)))
confusion = confusion_table(clf1.predict(X_train),
      y_train)
confusion
```

0.9925

Out[86]:

Truth	0	1
Predicted		
0	481	5
1	1	313

In [87]:
```
print('Error of unpruned tree on training data=',(1-accuracy_score(y_train,clf1.predict(X_train)))*100,'%')
```

Error of unpruned tree on training data= 0.7499999999999951 %

In [88]:
```
#pruned tree training accuracy and error
print(accuracy_score(y_train,
      best_.predict(X_train)))
confusion = confusion_table(best_.predict(X_train),
      y_train)
confusion
```

0.83625

Out[88]:

Truth	0	1
Predicted		
0	439	88
1	43	230

In [89]:
```
print('Error of pruned tree on training data=',(1-accuracy_score(y_train,best_.predict(X_train)))*100,'%')
```

Error of pruned tree on training data= 16.374999999999996 %

Training error is higher for the pruned tree

(k)

In [90]:
```
#unpruned tree test accuracy and error
print(accuracy_score(y_test,
      clf1.predict(X_test)))
confusion = confusion_table(clf1.predict(X_test),
      y_test)
confusion
```

```
0.7851851851851852
```

Out[90]:

Truth	0	1
Predicted		
0	143	30
1	28	69

In [91]:
```
print('Error of unpruned tree on test data=',(1-accuracy_score(y_test,clf1.predict(X_test)))*100,'%')
```

```
Error of unpruned tree on test data= 21.48148148148148 %
```

In [92]:
```
#pruned tree test accuracy and error
print(accuracy_score(y_test,
      best_.predict(X_test)))
confusion = confusion_table(best_.predict(X_test),
      y_test)
confusion
```

```
0.8407407407407408
```

Out[92]:

Truth	0	1
Predicted		
0	155	27
1	16	72

In [93]:
```
print('Error of pruned tree on test data=',(1-accuracy_score(y_test,best_.predict(X_test)))*100,'%')
```

```
Error of pruned tree on test data= 15.92592592592592 %
```

Test error is higher for the unpruned tree (clf1)

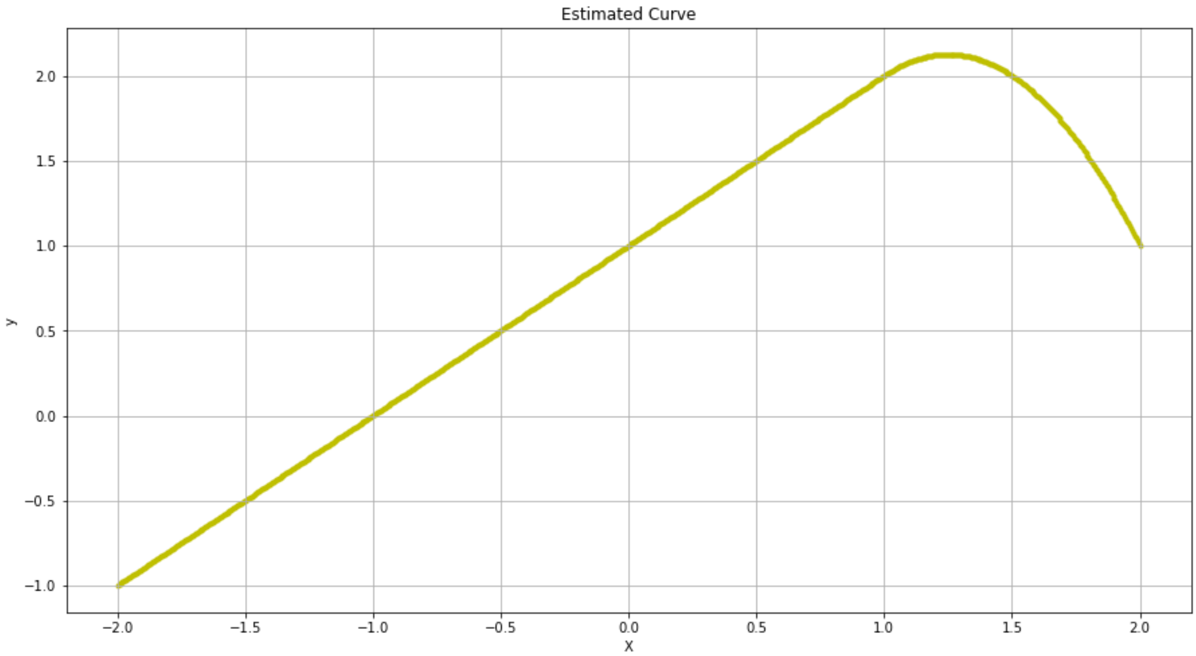
Section 7.9

Conceptual

7.9

3

In [78]:
```
def curve(X):
    β0=1
    β1=1
    β2=-2
    b1=X
    if X>=1:
        b2=(X-1)**2
    else:
        b2=0
    return β0+β1*b1+β2*b2


X = np.linspace(-2,2,800)

y=[]
for i in X:
    element=curve(i)
    y.append(element)

fig = plt.figure(figsize=(15, 8))
ax = fig.add_subplot(111)
plt.scatter(X, y, marker=".", color='y')
ax.set_xlabel('X')
ax.set_ylabel('y')
ax.set_title('Estimated Curve')
plt.grid()
plt.show()
```



Applied

6

In [79]:
```
Wage = load_data('Wage')
Wage.head(5)
```

Out[79]:

	year	age	maritl	race	education	region	jobclass	health	health_ins	logwage	wage
0	2006	18	1. Never Married	1. White	1. < HS Grad	2. Middle Atlantic	1. Industrial	1. <=Good	2. No	4.318063	75.043154
1	2004	24	1. Never Married	1. White	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	2. No	4.255273	70.476020
2	2003	45	2. Married	1. White	3. Some College	2. Middle Atlantic	1. Industrial	1. <=Good	1. Yes	4.875061	130.982177
3	2003	43	2. Married	3. Asian	4. College Grad	2. Middle Atlantic	2. Information	2. >=Very Good	1. Yes	5.041393	154.685293
4	2005	50	4. Divorced	1. White	2. HS Grad	2. Middle Atlantic	2. Information	1. <=Good	1. Yes	4.318063	75.043154

In [80]:
```
X = Wage['age']
X.head()
```

Out[80]:
```
0    18
1    24
2    45
3    43
4    50
Name: age, dtype: int64
```

In [81]:
```
y=Wage['wage']
y.head()
```

Out[81]:
```
0     75.043154
1     70.476020
2    130.982177
3    154.685293
4     75.043154
Name: wage, dtype: float64
```

In [83]:
```
# split the dataset, 25% data in the test set
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=4)
```

(a)

In [128]:
```
import statsmodels.api as sm
from ISLP.models import (summarize, poly, ModelSpec as MS)
from statsmodels.stats.anova import anova_lm
poly_age = MS([poly('age', degree=4)]).fit(Wage)
M = sm.OLS(y, poly_age.transform(Wage)).fit()
summarize(M)
```

Out[128]:

	coef	std err	t	P>|t|
intercept	111.7036	0.729	153.283	0.000
poly(age, degree=4)[0]	447.0679	39.915	11.201	0.000
poly(age, degree=4)[1]	-478.3158	39.915	-11.983	0.000
poly(age, degree=4)[2]	125.5217	39.915	3.145	0.002
poly(age, degree=4)[3]	-77.9112	39.915	-1.952	0.051

In [115]:
```

poly_age2 = MS([poly('age')]).fit(Wage)
M2 = sm.OLS(y, poly_age2.transform(Wage)).fit()
summarize(M2)
```

Out[115]:

	coef	std err	t	P>|t|
intercept	111.7036	0.747	149.484	0.0
poly(age, degree=1)	447.0679	40.929	10.923	0.0

In [116]:
```
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression()),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': [1,2,3,4,5]},
    scoring='neg_mean_squared_error',
    cv=5,
)

search.fit(Wage[['age']], Wage.wage)

first = -search.cv_results_['mean_test_score']
```

In [117]:
```
print(first)
```

```
[1675.01423805 1599.59822685 1594.72621626 1593.91410391 1595.43313111]
```

In [118]:
```
best=search.best_estimator_
```

In [119]:
```
print(best)
```

```
Pipeline(steps=[('poly', PolynomialFeatures(degree=4, include_bias=False)),
                ('model', LinearRegression())])
```

In [120]:
```
models = [MS([poly('age', degree=d)])
for d in range(1, 6)]
Xs = [model.fit_transform(Wage) for model in models]
anova_lm(*[sm.OLS(y, X_).fit()
for X_ in Xs])
```

Out[120]:

	df_resid	ssr	df_diff	ss_diff	F	Pr(>F)
0	2998.0	5.022216e+06	0.0	NaN	NaN	NaN
1	2997.0	4.793430e+06	1.0	228786.010128	143.593107	2.363850e-32
2	2996.0	4.777674e+06	1.0	15755.693664	9.888756	1.679202e-03
3	2995.0	4.771604e+06	1.0	6070.152124	3.809813	5.104620e-02
4	2994.0	4.770322e+06	1.0	1282.563017	0.804976	3.696820e-01

In [121]:
```
age_grid = np.linspace(age.min(),
age.max(),100)
age_df = pd.DataFrame({'age': age_grid})
age_df.head()
```

Out[121]:

	age
0	18.000000
1	18.626263
2	19.252525
3	19.878788
4	20.505051

In [122]:
```
X_test.head()
```

Out[122]:
```
571     39
1674    35
2858    42
331     35
1295    58
Name: age, dtype: int64
```

In [124]:
```
X_test_df = pd.DataFrame({'age': X_test})
X_test_df.head()
```

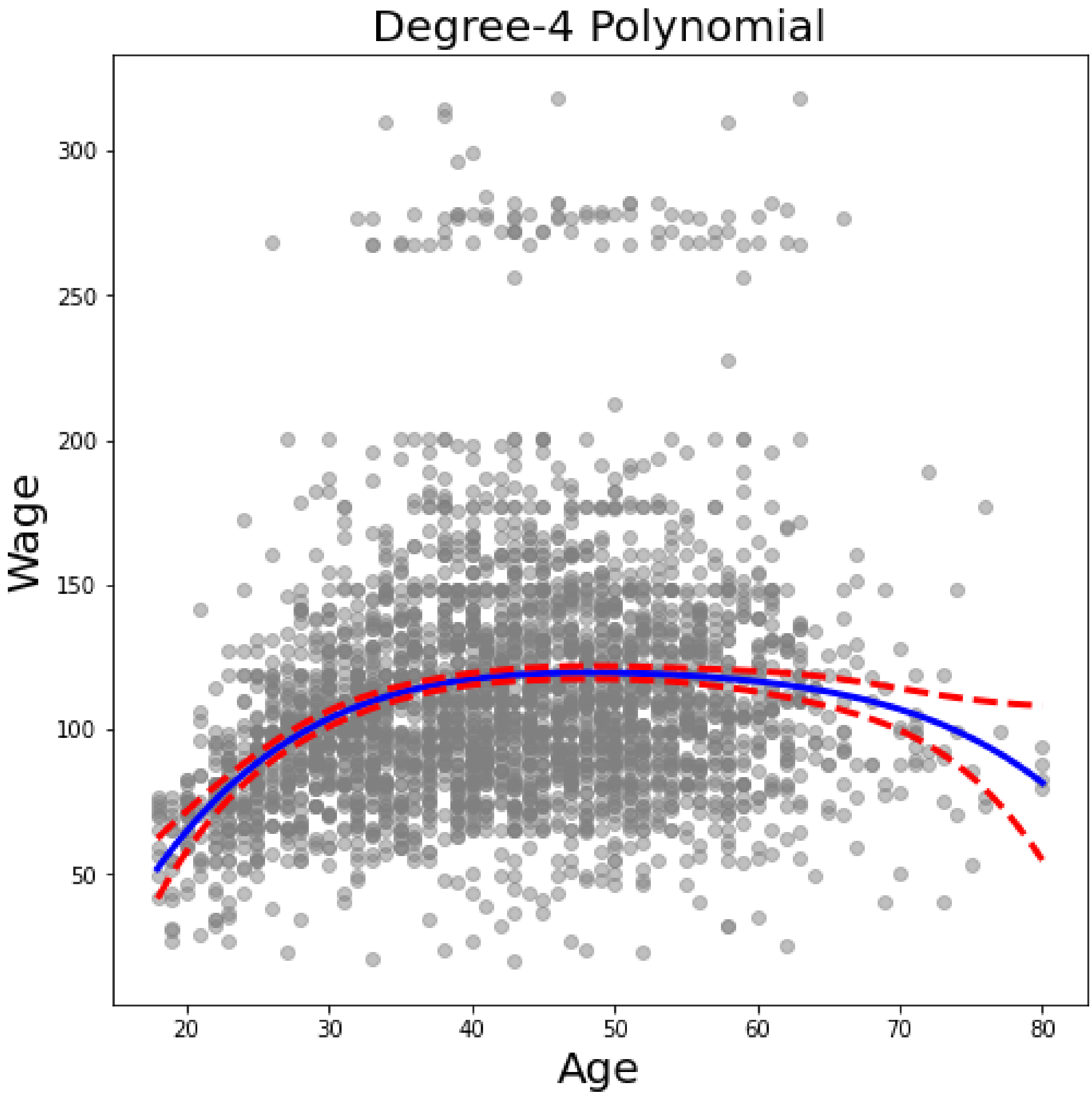
Out[124]:

	age
571	39
1674	35
2858	42
331	35
1295	58

In [140]:
```
newX = poly_age.transform(age_df)
```

In [141]:
```
preds = M.get_prediction(newX)
bands = preds.conf_int(alpha=0.05)
```

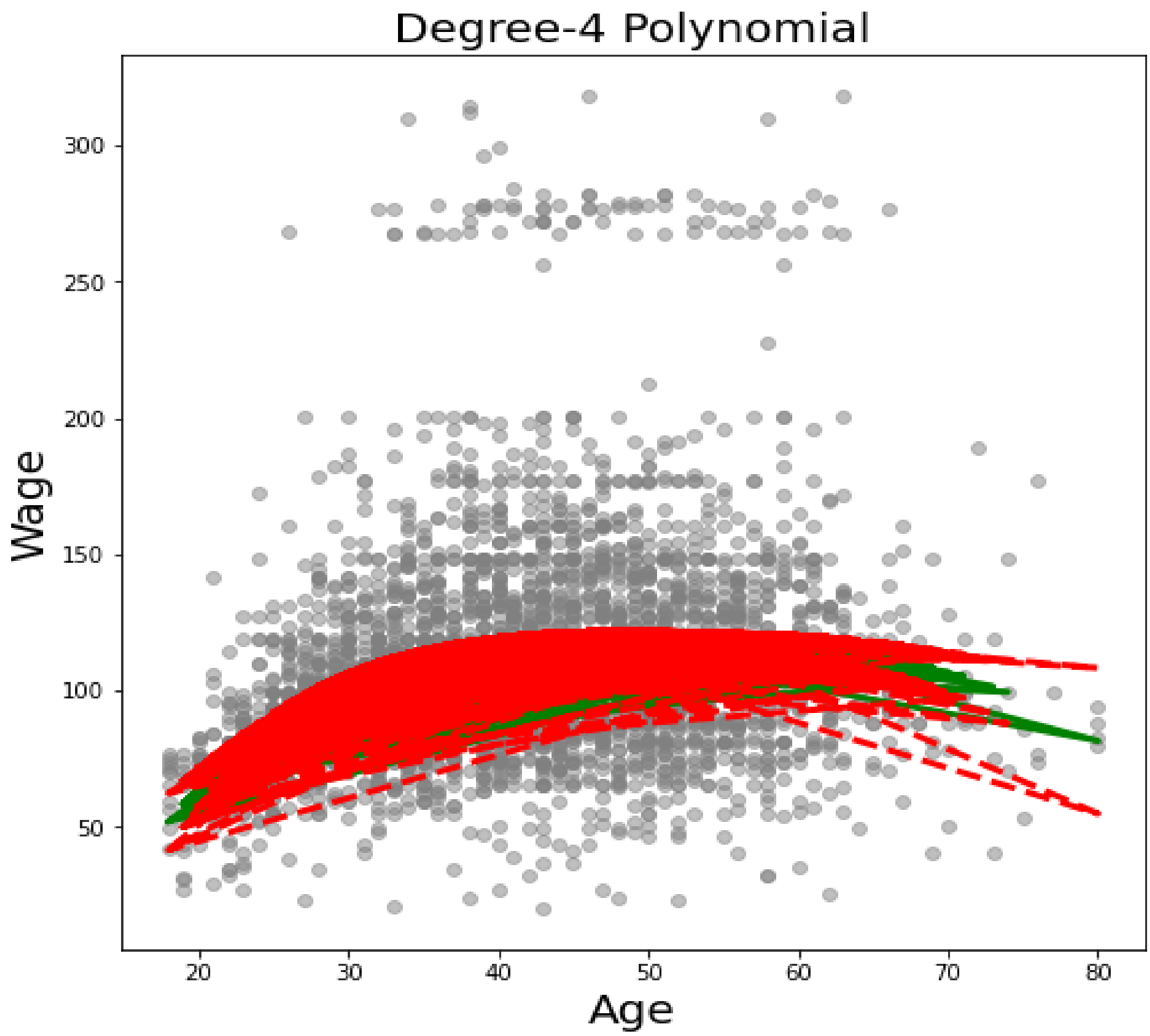
In [144]:
```
fig, ax = subplots(figsize=(8,8))
ax.scatter(age, y, facecolor='gray', alpha=0.5)
for val, ls in zip([preds.predicted_mean,bands[:,0],bands[:,1]],['b','r--','r--']):
    ax.plot(age_df.values, val, ls, linewidth=3)
ax.set_title( 'Degree-4 Polynomial', fontsize=20)
ax.set_xlabel('Age', fontsize=20)
ax.set_ylabel('Wage', fontsize=20);
```



In [145]:
```
newX = poly_age.transform(X_test_df)
```

In [146]:
```
preds = M.get_prediction(newX)
bands = preds.conf_int(alpha=0.05)
```

In [147]:
```
fig, ax = subplots(figsize=(8,8))
ax.scatter(age, y, facecolor='gray', alpha=0.5)
for val, ls in zip([preds.predicted_mean,bands[:,0],bands[:,1]],['g','r--','r--']):
    ax.plot(X_test_df.values, val, ls, linewidth=3)
ax.set_title( 'Degree-4 Polynomial', fontsize=20)
ax.set_xlabel('Age', fontsize=20)
ax.set_ylabel('Wage', fontsize=20);
```



The above plot is obtained using X_test. It is messier than that using age_df since age_df has 100 evenly spaced points in the age range unlike randomly selected X_test points (25% of 3000 i.e ,750 in count)

Also, both cross validation and ANOVA lead to selecting the optimal degree as 4. (Cross validation-Degree 4 turns out to be the best estimator. ANOVA-When comparing the linear models [0] to the quadratic models [1], the p-value is almost zero, suggesting that a linear fit is insufficient. In a similar vein, the p-value between the cubic and quadratic models is extremely low(0.0017), thus the squared fit is likewise insufficient.P-value comparison for the degree-four cubic polynomials, models [2] and [3], are approximately roughly 5%, but the degree-five polynomial models[4]don't seem to be necessary due to its p-value of 0.37. Hence a quartic-degree 4 fit seems to be a good choice using ANOVA).

(b)

In [150]:
```
cut_age = pd.qcut(age, 4)

summarize(sm.OLS(y, pd.get_dummies(cut_age)).fit())
```

Out[150]:

	coef	std err	t	P>|t|
(17.999, 33.75]	94.1584	1.478	63.692	0.0
(33.75, 42.0]	116.6608	1.470	79.385	0.0
(42.0, 51.0]	119.1887	1.416	84.147	0.0
(51.0, 80.0]	116.5717	1.559	74.751	0.0

In [151]:
```
cut_age = pd.qcut(age, 3)

summarize(sm.OLS(y, pd.get_dummies(cut_age)).fit())
```

Out[151]:

	coef	std err	t	P>|t|
(17.999, 37.0]	99.2685	1.260	78.762	0.0
(37.0, 48.0]	119.7563	1.274	94.011	0.0
(48.0, 80.0]	116.7918	1.333	87.648	0.0

In [152]:
```
cut_age = pd.qcut(age, 2)

summarize(sm.OLS(y, pd.get_dummies(cut_age)).fit())
```

Out[152]:

	coef	std err	t	P>|t|
(17.999, 42.0]	105.4767	1.062	99.299	0.0
(42.0, 80.0]	118.0057	1.069	110.430	0.0

In [153]:
```
cut_age = pd.qcut(age, 5)

summarize(sm.OLS(y, pd.get_dummies(cut_age)).fit())
```

Out[153]:

	coef	std err	t	P>|t|
(17.999, 32.0]	92.3224	1.570	58.792	0.0
(32.0, 39.0]	115.2538	1.719	67.030	0.0
(39.0, 46.0]	119.5162	1.575	75.879	0.0
(46.0, 53.0]	116.7134	1.676	69.636	0.0
(53.0, 80.0]	116.9012	1.735	67.371	0.0

In [154]:
```
cut_age = pd.qcut(age, 6)

summarize(sm.OLS(y, pd.get_dummies(cut_age)).fit())
```

Out[154]:

	coef	std err	t	P>|t|
(17.999, 30.0]	89.5371	1.765	50.716	0.0
(30.0, 37.0]	108.9999	1.765	61.740	0.0
(37.0, 42.0]	119.4151	1.871	63.840	0.0
(42.0, 48.0]	120.0411	1.709	70.237	0.0
(48.0, 54.0]	117.0123	1.883	62.151	0.0
(54.0, 80.0]	116.5787	1.851	62.990	0.0

In [155]:
```
cut_age = pd.qcut(age, 7)

summarize(sm.OLS(y, pd.get_dummies(cut_age)).fit())
```

Out[155]:

	coef	std err	t	P>|t|
(17.999, 29.0]	86.7282	1.901	45.619	0.0
(29.0, 35.0]	107.1893	1.891	56.695	0.0
(35.0, 40.0]	118.6607	1.945	61.006	0.0
(40.0, 45.0]	119.4403	1.864	64.075	0.0
(45.0, 49.0]	119.0404	2.139	55.659	0.0
(49.0, 55.0]	116.4898	1.916	60.792	0.0
(55.0, 80.0]	116.4429	1.987	58.593	0.0

In [156]:
```
cut_age = pd.qcut(age, 8)

summarize(sm.OLS(y, pd.get_dummies(cut_age)).fit())
```

Out[156]:

	coef	std err	t	P>|t|
(17.999, 28.0]	84.7401	2.035	41.632	0.0
(28.0, 33.75]	104.3616	2.119	49.260	0.0
(33.75, 38.0]	116.6870	2.070	56.363	0.0
(38.0, 42.0]	116.6349	2.057	56.710	0.0
(42.0, 46.375]	121.6993	2.101	57.920	0.0
(46.375, 51.0]	117.1512	1.893	61.892	0.0
(51.0, 56.0]	116.6067	2.226	52.376	0.0
(56.0, 80.0]	116.5390	2.155	54.083	0.0

In [183]:
```
pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression()),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': [4]},
    scoring='neg_mean_squared_error',
    cv=5,
)
#cut_age = pd.qcut(age,cuts )
cuts2=search.fit(pd.get_dummies(pd.qcut(age,2 )), Wage.wage)
```

In [184]:
```
pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression()),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': [4]},
    scoring='neg_mean_squared_error',
    cv=5,
)
#cut_age = pd.qcut(age,cuts )
cuts3=search.fit(pd.get_dummies(pd.qcut(age,3 )), Wage.wage)
```

In [185]:

```
pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression()),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': [4]},
    scoring='neg_mean_squared_error',
    cv=5,
)
#cut_age = pd.qcut(age,cuts )
cuts4=search.fit(pd.get_dummies(pd.qcut(age,4 )), Wage.wage)
```

In [186]:

```
pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression()),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': [4]},
    scoring='neg_mean_squared_error',
    cv=5,
)
#cut_age = pd.qcut(age,cuts )
cuts5=search.fit(pd.get_dummies(pd.qcut(age,5 )), Wage.wage)
```

In [187]:
```
pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression()),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': [4]},
    scoring='neg_mean_squared_error',
    cv=5,
)
#cut_age = pd.qcut(age,cuts )
cuts6=search.fit(pd.get_dummies(pd.qcut(age,6 )), Wage.wage)
```

In [188]:
```
pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression()),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': [4]},
    scoring='neg_mean_squared_error',
    cv=5,
)
#cut_age = pd.qcut(age,cuts )
cuts7=search.fit(pd.get_dummies(pd.qcut(age,7 )), Wage.wage)
```

In [189]:
```
pipe = Pipeline(steps=[
    ('poly', PolynomialFeatures(include_bias=False)),
    ('model', LinearRegression()),
])

search = GridSearchCV(
    estimator=pipe,
    param_grid={'poly__degree': [4]},
    scoring='neg_mean_squared_error',
    cv=5,
)
#cut_age = pd.qcut(age,cuts )
cuts8=search.fit(pd.get_dummies(pd.qcut(age,8 )), Wage.wage)
```

In [190]:
```
print(cuts2)
```

```
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('poly',
                                        PolynomialFeatures(include_bias=False)),
                                       ('model', LinearRegression())]),
             param_grid={'poly__degree': [4]},
             scoring='neg_mean_squared_error')
```

In [191]:
```
cuts2.cv_results_
```

Out[191]:
```
{'mean_fit_time': array([0.01474643]),
 'std_fit_time': array([0.00342521]),
 'mean_score_time': array([0.00464735]),
 'std_score_time': array([0.00375517]),
 'param_poly__degree': masked_array(data=[4],
              mask=[False],
        fill_value='?',
             dtype=object),
 'params': [{'poly__degree': 4}],
 'split0_test_score': array([-1886.90351799]),
 'split1_test_score': array([-1579.72929867]),
 'split2_test_score': array([-1572.26547524]),
 'split3_test_score': array([-1713.02492424]),
 'split4_test_score': array([-1766.01620479]),
 'mean_test_score': array([-1703.58788419]),
 'std_test_score': array([118.47151458]),
 'rank_test_score': array([1])}
```

In [192]:
```
cuts3.cv_results_
```

Out[192]:
```
{'mean_fit_time': array([0.01806049]),
 'std_fit_time': array([0.00263596]),
 'mean_score_time': array([0.00892191]),
 'std_score_time': array([0.00330277]),
 'param_poly__degree': masked_array(data=[4],
              mask=[False],
        fill_value='?',
             dtype=object),
 'params': [{'poly__degree': 4}],
 'split0_test_score': array([-1825.89301155]),
 'split1_test_score': array([-1556.88763826]),
 'split2_test_score': array([-1520.3200526]),
 'split3_test_score': array([-1673.90166593]),
 'split4_test_score': array([-1717.30001627]),
 'mean_test_score': array([-1658.86047692]),
 'std_test_score': array([110.57413237]),
 'rank_test_score': array([1])}
```

In [195]:
```
cuts4.cv_results_
```

Out[195]:
```
{'mean_fit_time': array([0.02366061]),
 'std_fit_time': array([0.00629885]),
 'mean_score_time': array([0.00557742]),
 'std_score_time': array([0.00562271]),
 'param_poly__degree': masked_array(data=[4],
              mask=[False],
        fill_value='?',
             dtype=object),
 'params': [{'poly__degree': 4}],
 'split0_test_score': array([-1805.90479158]),
 'split1_test_score': array([-1561.84864391]),
 'split2_test_score': array([-1512.13688721]),
 'split3_test_score': array([-1653.63393418]),
 'split4_test_score': array([-1678.35673299]),
 'mean_test_score': array([-1642.37619798]),
 'std_test_score': array([101.59714709]),
 'rank_test_score': array([1])}
```

In [196]:
```
cuts5.cv_results_
```

Out[196]:
```
{'mean_fit_time': array([0.04660759]),
 'std_fit_time': array([0.00152269]),
 'mean_score_time': array([0.00941267]),
 'std_score_time': array([0.00542365]),
 'param_poly__degree': masked_array(data=[4],
              mask=[False],
        fill_value='?',
             dtype=object),
 'params': [{'poly__degree': 4}],
 'split0_test_score': array([-1819.62860462]),
 'split1_test_score': array([-1539.17359768]),
 'split2_test_score': array([-1492.60207789]),
 'split3_test_score': array([-1644.64827807]),
 'split4_test_score': array([-1691.30724629]),
 'mean_test_score': array([-1637.47196091]),
 'std_test_score': array([115.56870818]),
 'rank_test_score': array([1])}
```

In [197]:
```
cuts6.cv_results_
```

Out[197]:
```
{'mean_fit_time': array([0.09362712]),
 'std_fit_time': array([0.00233327]),
 'mean_score_time': array([0.01126738]),
 'std_score_time': array([0.00633288]),
 'param_poly__degree': masked_array(data=[4],
              mask=[False],
        fill_value='?',
             dtype=object),
 'params': [{'poly__degree': 4}],
 'split0_test_score': array([-1806.0001232]),
 'split1_test_score': array([-1532.63202151]),
 'split2_test_score': array([-1480.21158007]),
 'split3_test_score': array([-1643.62675461]),
 'split4_test_score': array([-1685.85144673]),
 'mean_test_score': array([-1629.66438522]),
 'std_test_score': array([115.06315371]),
 'rank_test_score': array([1])}
```

In [198]:
```
cuts7.cv_results_
```

Out[198]:
```
{'mean_fit_time': array([0.18239222]),
 'std_fit_time': array([0.00887817]),
 'mean_score_time': array([0.01597924]),
 'std_score_time': array([0.00090419]),
 'param_poly__degree': masked_array(data=[4],
              mask=[False],
        fill_value='?',
             dtype=object),
 'params': [{'poly__degree': 4}],
 'split0_test_score': array([-1799.85128857]),
 'split1_test_score': array([-1544.75779302]),
 'split2_test_score': array([-1467.00738265]),
 'split3_test_score': array([-1644.16318374]),
 'split4_test_score': array([-1673.85259813]),
 'mean_test_score': array([-1625.92644922]),
 'std_test_score': array([113.77430681]),
 'rank_test_score': array([1])}
```

```
In [199]: cuts8.cv_results_

Out[199]: {'mean_fit_time': array([0.18422732]),
           'std_fit_time': array([0.01338262]),
           'mean_score_time': array([0.00964222]),
           'std_score_time': array([0.00788229]),
           'param_poly__degree': masked_array(data=[4],
                        mask=[False],
                  fill_value='?',
                       dtype=object),
           'params': [{'poly__degree': 4}],
           'split0_test_score': array([-1786.84511619]),
           'split1_test_score': array([-1541.70917902]),
           'split2_test_score': array([-1468.70470278]),
           'split3_test_score': array([-1633.32368356]),
           'split4_test_score': array([-1666.61695611]),
           'mean_test_score': array([-1619.43992753]),
           'std_test_score': array([108.81737466]),
           'rank_test_score': array([1])}

In [210]: newX = poly_age.transform(age_df)

In [211]: cut_age = pd.qcut(age, 8)

          cut8M=sm.OLS(y, pd.get_dummies(cut_age)).fit()
```

In [228]:
```
pd.get_dummies(cut_age)
```

Out[228]:

	(17.999, 28.0]	(28.0, 33.75]	(33.75, 38.0]	(38.0, 42.0]	(42.0, 46.375]	(46.375, 51.0]	(51.0, 56.0]	(56.0, 80.0]
0	1	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0
3	0	0	0	0	1	0	0	0
4	0	0	0	0	0	1	0	0
...	...	...	...	...	...	...	...	...
2995	0	0	0	0	1	0	0	0
2996	0	1	0	0	0	0	0	0
2997	1	0	0	0	0	0	0	0
2998	1	0	0	0	0	0	0	0
2999	0	0	0	0	0	0	1	0

3000 rows × 8 columns

In [217]:
```
newX
```

Out[217]:

	intercept	poly(age, degree=4)[0]	poly(age, degree=4)[1]	poly(age, degree=4)[2]	poly(age, degree=4)[3]
0	1.0	-0.038625	0.055909	-0.071741	0.086730
1	1.0	-0.037634	0.052445	-0.064068	0.072513
2	1.0	-0.036643	0.049068	-0.056818	0.059599
3	1.0	-0.035652	0.045779	-0.049979	0.047921
4	1.0	-0.034662	0.042576	-0.043544	0.037415
...	...	...	...	...	...
95	1.0	0.055498	0.114987	0.152588	0.134426
96	1.0	0.056489	0.119781	0.164684	0.155650
97	1.0	0.057480	0.124662	0.177306	0.178605
98	1.0	0.058470	0.129630	0.190463	0.203368
99	1.0	0.059461	0.134685	0.204166	0.230017

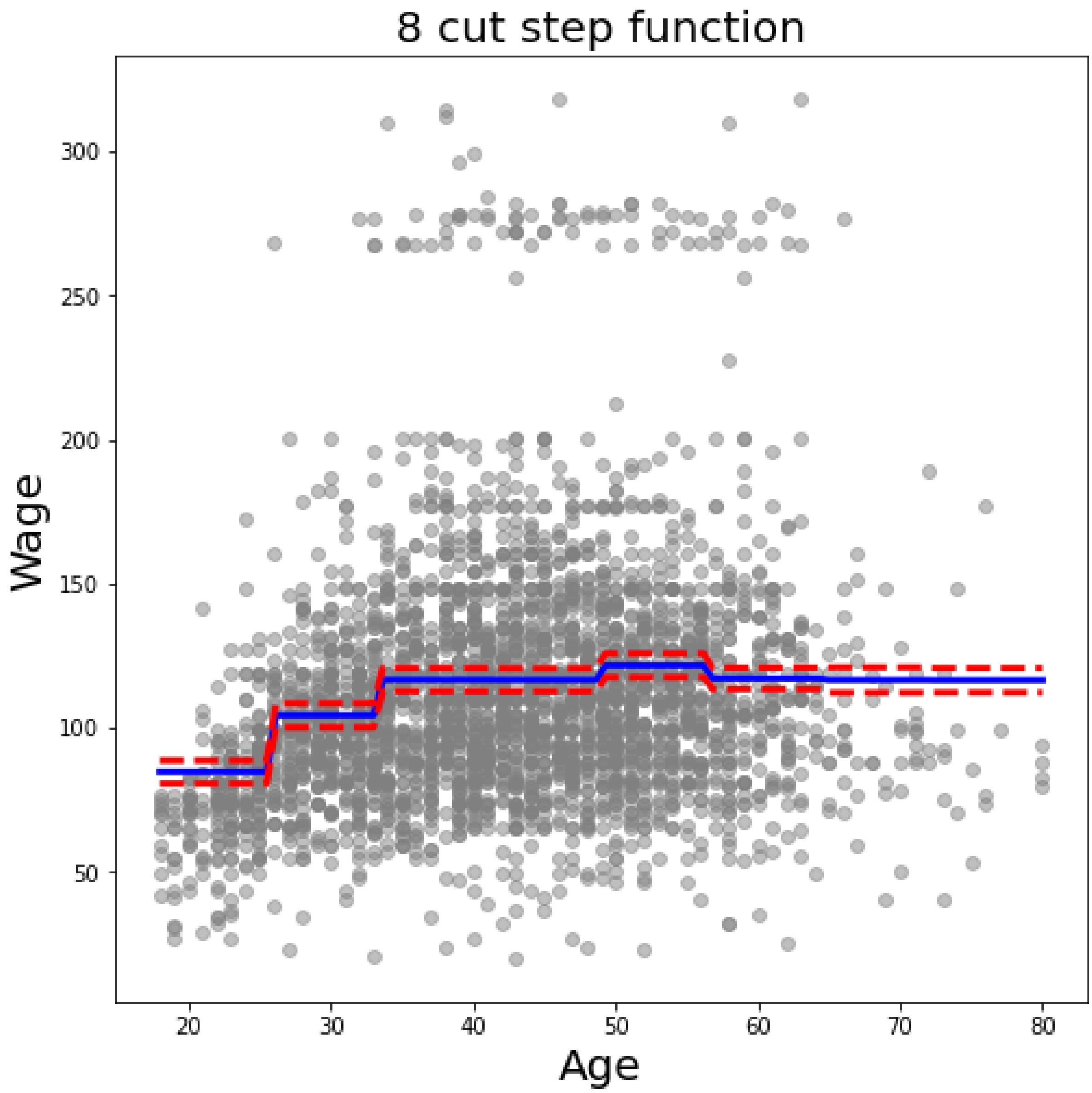
100 rows × 5 columns

In [218]:
```
cut_age
```

Out[218]:
```
0        (17.999, 28.0]
1        (17.999, 28.0]
2        (42.0, 46.375]
3        (42.0, 46.375]
4        (46.375, 51.0]
              ...
2995     (42.0, 46.375]
2996      (28.0, 33.75]
2997     (17.999, 28.0]
2998     (17.999, 28.0]
2999       (51.0, 56.0]
Name: age, Length: 3000, dtype: category
Categories (8, interval[float64, right]): [(17.999, 28.0] < (28.0, 33.75] < (33.75, 38.0] < (38.0, 42.0] < (42.0, 46.375] < (4
6.375, 51.0] < (51.0, 56.0] < (56.0, 80.0]]
```

In [222]:
```
cut_ageX = pd.qcut(age_grid, 8)
cut_ageX
```

```
Out[222]: [(17.999, 25.75], (17.999, 25.75], (17.999, 25.75], (17.999, 25.75], (17.999, 25.75], ..., (72.25, 80.0], (72.25, 80.0], (72.2
5, 80.0], (72.25, 80.0], (72.25, 80.0]]
Length: 100
Categories (8, interval[float64, right]): [(17.999, 25.75] < (25.75, 33.5] < (33.5, 41.25] < (41.25, 49.0] < (49.0, 56.75] < (5
6.75, 64.5] < (64.5, 72.25] < (72.25, 80.0]]

In [223]: cut8M

Out[223]: <statsmodels.regression.linear_model.RegressionResultsWrapper at 0x191449e8fd0>

In [230]: cut_age = pd.qcut(age, 8)

#cut8M=sm.OLS(y, pd.get_dummies(cut_age)).fit()
preds = sm.OLS(y, pd.get_dummies(cut_age)).fit().get_prediction(pd.get_dummies(cut_ageX))

In [231]: #preds = cut8M.get_prediction(cut_ageX)
bands = preds.conf_int(alpha=0.05)

In [232]: fig, ax = subplots(figsize=(8,8))
ax.scatter(age, y, facecolor='gray', alpha=0.5)
for val, ls in zip([preds.predicted_mean,bands[:,0],bands[:,1]],['b','r--','r--']):
    ax.plot(age_df.values, val, ls, linewidth=3)
ax.set_title( '8 cut step function', fontsize=20)
ax.set_xlabel('Age', fontsize=20)
ax.set_ylabel('Wage', fontsize=20);
```



8 cuts seem to be optimal as they give maximum mean_test_score.