In [1]:
```python
import numpy as np
import pandas as pd
```

Problem 2 (Chapter 8, Exercise 8)

In [2]:
```python
carseats = pd.read_csv("data/Carseats.csv")

print(carseats.info())
carseats.head()
```

```
<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
None
```

Out[2]:

	Sales	CompPrice	Income	Advertising	Population	Price	ShelveLoc	Age	Education	Urba
0	9.50	138	73	11	276	120	Bad	42	17	Ye
1	11.22	111	48	16	260	83	Good	65	10	Ye
2	10.06	113	35	10	269	80	Medium	59	12	Ye
3	7.40	117	100	4	466	97	Medium	55	14	Ye
4	4.15	141	64	3	340	128	Bad	38	13	Ye

In [3]:
```python
from sklearn.preprocessing import OneHotEncoder

# fix categorical columns
for cat in ["Urban", "US"]:
    carseats[cat] = carseats[cat].astype('category').cat.codes

enc = OneHotEncoder(sparse=False)
shelve_loc = enc.fit_transform(carseats["ShelveLoc"].to_numpy().reshape(-1, 1

carseats["ShelveLocBad"]    = shelve_loc[:, 0]
carseats["ShelveLocMedium"] = shelve_loc[:, 1]
carseats["ShelveLocGood"]   = shelve_loc[:, 2]

carseats = carseats.drop("ShelveLoc", axis=1)

print(carseats.info())
carseats.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 13 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   Age              400 non-null    int64  
 7   Education        400 non-null    int64  
 8   Urban            400 non-null    int8   
 9   US               400 non-null    int8   
 10  ShelveLocBad     400 non-null    float64
 11  ShelveLocMedium  400 non-null    float64
 12  ShelveLocGood    400 non-null    float64
dtypes: float64(4), int64(7), int8(2)
memory usage: 35.3 KB
None
```

Out[3]:

	Sales	CompPrice	Income	Advertising	Population	Price	Age	Education	Urban	US	She
0	9.50	138	73	11	276	120	42	17	1	1	
1	11.22	111	48	16	260	83	65	10	1	1	
2	10.06	113	35	10	269	80	59	12	1	1	
3	7.40	117	100	4	466	97	55	14	1	1	
4	4.15	141	64	3	340	128	38	13	1	0	

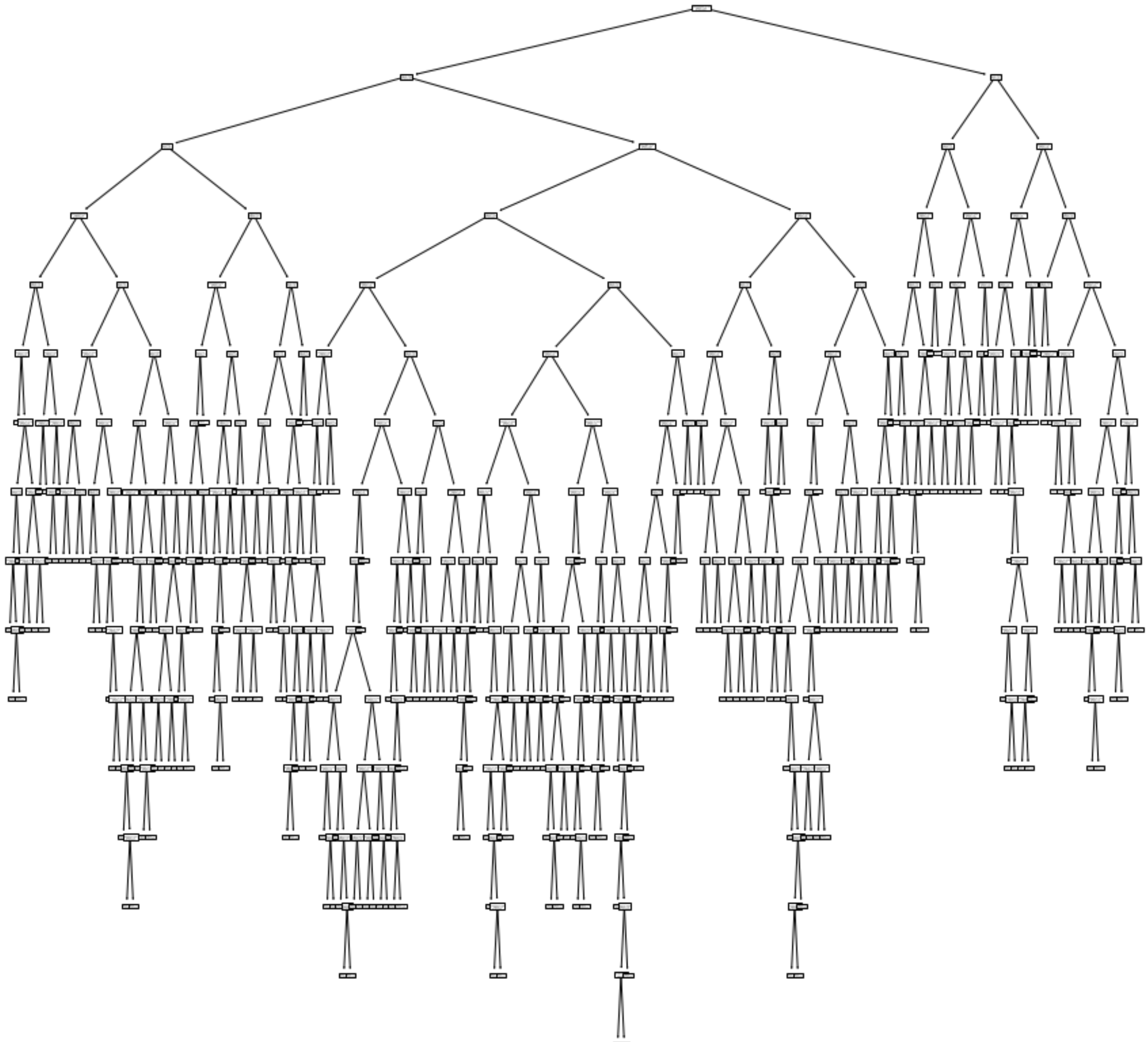
Problem 2(a)

In [4]:
```
from sklearn.model_selection import train_test_split

X = carseats.drop("Sales", axis=1)
y = carseats["Sales"]

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
```

Problem 2(b)

In [5]:
```
from sklearn.tree import DecisionTreeRegressor, plot_tree

reg_tree = DecisionTreeRegressor(random_state=0)
reg_tree.fit(X_train, y_train);
```

In [6]:
```
from matplotlib import pyplot as plt

fig = plt.figure(figsize=(15,15))
ax  = fig.gca()
plot_tree(reg_tree, ax=ax, feature_names=carseats.drop("Sales", axis=1).colum
          class_names="Sales", impurity=False);
```



The tree is very highly branched and possibly overfit to the dataset, so we may want to do some pruning.

In [7]:
```
from sklearn.metrics import mean_squared_error

mse = mean_squared_error(y_test, reg_tree.predict(X_test))
print("decision tree test mse: %.3f" % mse)
```

```
decision tree test mse: 5.160
```

Problem 2(c)

In [8]:
```
from sklearn.model_selection import cross_validate

min_decr_list = np.logspace(-3, 3, 13)

for min_decr in min_decr_list:

    reg_tree = DecisionTreeRegressor(random_state=0, min_impurity_decrease=mi

    cv_results = cross_validate(reg_tree, X, y, cv=10, scoring='neg_mean_squa

    avg_test_mse = np.mean(-1*cv_results['test_score'])

    print("min split decr: %5.3e" % min_decr, " |  test mse:", avg_test_mse)
```

```
min split decr: 1.000e-03  |  test mse: 5.021373217986111
min split decr: 3.162e-03  |  test mse: 4.878396738488783
min split decr: 1.000e-02  |  test mse: 4.661581790056902
min split decr: 3.162e-02  |  test mse: 4.418206140487827
min split decr: 1.000e-01  |  test mse: 4.756998382990023
min split decr: 3.162e-01  |  test mse: 4.903125250448153
min split decr: 1.000e+00  |  test mse: 6.114254875860025
min split decr: 3.162e+00  |  test mse: 8.024303693441357
min split decr: 1.000e+01  |  test mse: 8.024303693441357
min split decr: 3.162e+01  |  test mse: 8.024303693441357
min split decr: 1.000e+02  |  test mse: 8.024303693441357
min split decr: 3.162e+02  |  test mse: 8.024303693441357
min split decr: 1.000e+03  |  test mse: 8.024303693441357
```

In [9]:
```
max_nodes_list = np.logspace(np.log10(2), np.log10(1000), 20)

for max_nodes in max_nodes_list:

    reg_tree = DecisionTreeRegressor(random_state=0, max_leaf_nodes=round(max_

    cv_results = cross_validate(reg_tree, X, y, cv=10, scoring='neg_mean_squa

    avg_test_mse = np.mean(-1*cv_results['test_score'])

    print("max leaf nodes: %4d" % round(max_nodes), " |  test mse:", avg_test_
```

```
max leaf nodes:    2  |  test mse: 6.021211549802172
max leaf nodes:    3  |  test mse: 5.463939411950188
max leaf nodes:    4  |  test mse: 5.350477382126139
max leaf nodes:    5  |  test mse: 4.761552963590645 5
max leaf nodes:    7  |  test mse: 4.869271383110496
max leaf nodes:   10  |  test mse: 4.74810583970232
max leaf nodes:   14  |  test mse: 4.80136840944212
max leaf nodes:   20  |  test mse: 4.639243894469821
max leaf nodes:   27  |  test mse: 4.62718150075911
max leaf nodes:   38  |  test mse: 4.415070541600826
max leaf nodes:   53  |  test mse: 4.466614060143851
max leaf nodes:   73  |  test mse: 4.55241205750468 6
max leaf nodes:  101  |  test mse: 4.759964075058629
max leaf nodes:  141  |  test mse: 4.94207044564928
max leaf nodes:  195  |  test mse: 5.063093958541667
max leaf nodes:  270  |  test mse: 5.071329741319444
max leaf nodes:  375  |  test mse: 5.07338375
max leaf nodes:  520  |  test mse: 5.07338375
max leaf nodes:  721  |  test mse: 5.07338375
max leaf nodes: 1000  |  test mse: 5.07338375
```

In [10]:

```python
max_depth_list = np.logspace(np.log10(2), np.log10(100), 20)

for max_depth in max_depth_list:

    reg_tree = DecisionTreeRegressor(random_state=0, max_depth=round(max_dept

    cv_results = cross_validate(reg_tree, X, y, cv=10, scoring='neg_mean_squa

    avg_test_mse = np.mean(-1*cv_results['test_score'])

    print("max tree depth: %3d" % round(max_depth), " |  test mse:", avg_test_
```

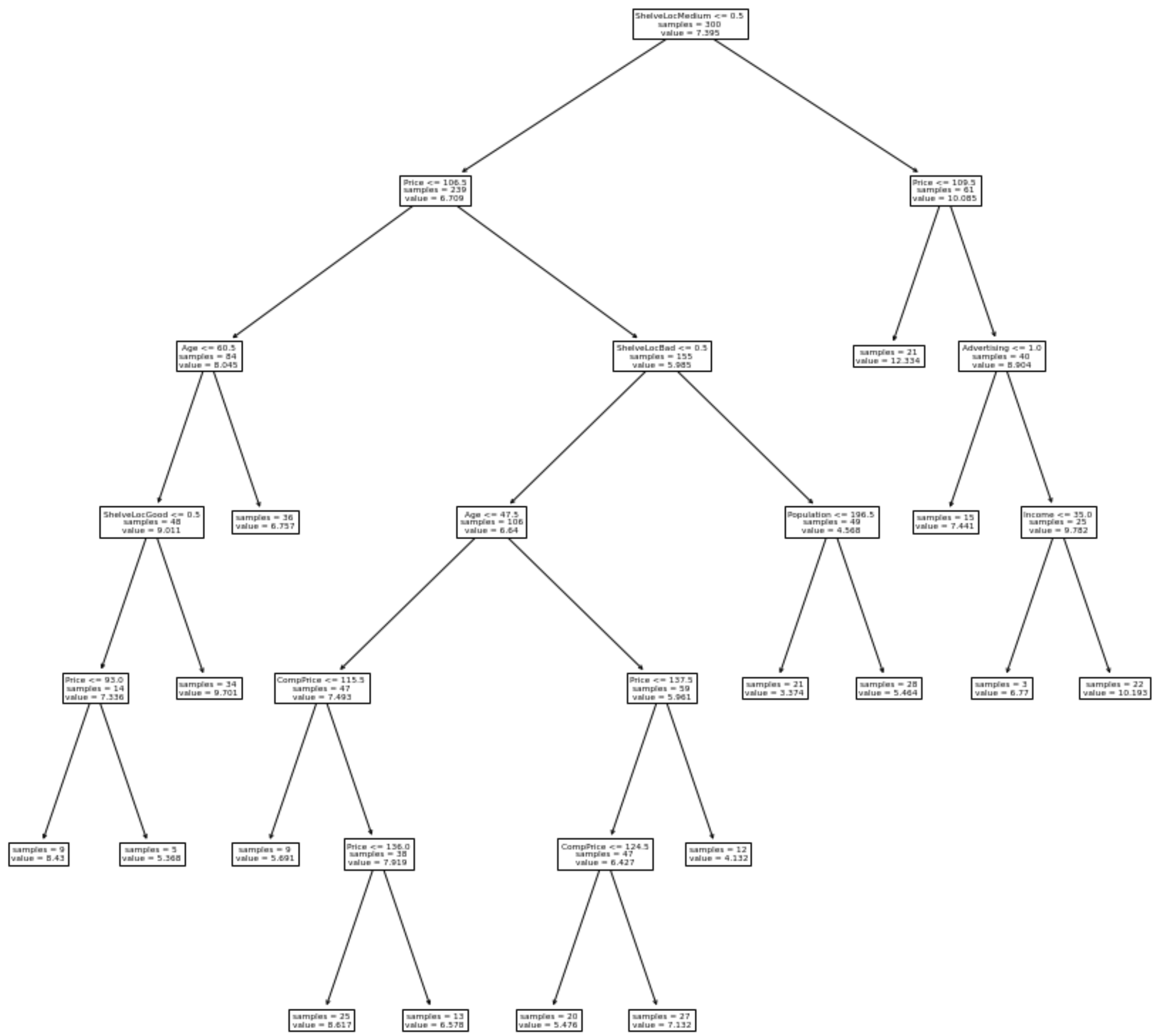
```
max tree depth:   2  |  test mse: 5.175545607747162
max tree depth:   2  |  test mse: 5.175545607747162
max tree depth:   3  |  test mse: 4.759024284061128 5
max tree depth:   4  |  test mse: 4.882876270347563
max tree depth:   5  |  test mse: 4.646330628189368
max tree depth:   6  |  test mse: 4.502933622738348 5
max tree depth:   7  |  test mse: 4.625605916258284
max tree depth:   8  |  test mse: 4.601803437534455
max tree depth:  10  |  test mse: 5.085113419041463 5
max tree depth:  13  |  test mse: 5.103489814687501
max tree depth:  16  |  test mse: 5.111214562500001
max tree depth:  19  |  test mse: 4.9669585
max tree depth:  24  |  test mse: 4.9669585
max tree depth:  29  |  test mse: 4.9669585
max tree depth:  36  |  test mse: 4.9669585
max tree depth:  44  |  test mse: 4.9669585
max tree depth:  54  |  test mse: 4.9669585
max tree depth:  66  |  test mse: 4.9669585
max tree depth:  81  |  test mse: 4.9669585
max tree depth: 100  |  test mse: 4.9669585
```

The optimal tree complexity is with min split decrease of ~3-10 E-2, max leaf nodes of ~50, and max tree depth of ~6.

In [11]:

```
reg_tree = DecisionTreeRegressor(random_state=0, min_impurity_decrease=1E-1, r
reg_tree.fit(X_train, y_train)

fig = plt.figure(figsize=(15,15))
ax  = fig.gca()
plot_tree(reg_tree, ax=ax, feature_names=carseats.drop("Sales", axis=1).colum
          class_names="Sales", impurity=False);

mse = mean_squared_error(y_test, reg_tree.predict(X_test))
print("decision tree test mse: %.3f" % mse)
```

```
decision tree test mse: 4.515
```



Pruning does indeed help improve the MSE.

Problem 2(d)

In [12]:
```
from sklearn.ensemble import BaggingRegressor

bag_reg = BaggingRegressor(DecisionTreeRegressor(random_state=0), random_stat
bag_reg.fit(X_train, y_train);
```

In [13]:
```
mse = mean_squared_error(y_test, bag_reg.predict(X_test))
print("bagged decision tree test mse:", mse)
```

```
bagged decision tree test mse: 2.6994652600000006
```

In [14]:
```
feature_importances = np.mean([tree.feature_importances_ for tree in bag_reg.

idxs = np.argsort(feature_importances)[::-1]

for col, imp in zip(X.columns[idxs], feature_importances[idxs]):
    print("%15s" %col, " |  %3f" %imp)

print("Top 3 features are:", X.columns[idxs][0], "", X.columns[idxs][1], "",
```

```
          Price  |  0.292875
ShelveLocMedium  |  0.215514
            Age  |  0.121255
      CompPrice  |  0.095484
    Advertising  |  0.077952
         Income  |  0.050261
   ShelveLocBad  |  0.040722
     Population  |  0.036205
  ShelveLocGood  |  0.033244
      Education  |  0.027782
          Urban  |  0.006050
             US  |  0.002655
Top 3 features are: Price  ShelveLocMedium  Age
```

Problem 2(e)

In [15]:
```
from sklearn.ensemble import RandomForestRegressor

rf_reg = RandomForestRegressor(random_state=0)
rf_reg.fit(X_train, y_train);
```

In [16]:
```
mse = mean_squared_error(y_test, rf_reg.predict(X_test))
print("random forest test mse:", mse)
```

```
random forest test mse: 2.263103768200001
```

In [17]:
```
feature_importances = np.mean([tree.feature_importances_ for tree in rf_reg.e

idxs = np.argsort(feature_importances)[::-1]

for col, imp in zip(X.columns[idxs], feature_importances[idxs]):
    print("%15s" %col, " |  %3f" %imp)

print("Top 3 features are:", X.columns[idxs][0], "", X.columns[idxs][1], "",
```

```
           Price  |  0.290875
 ShelveLocMedium  |  0.195175
             Age  |  0.105819
       CompPrice  |  0.098693
     Advertising  |  0.067502
    ShelveLocBad  |  0.064920
          Income  |  0.052249
   ShelveLocGood  |  0.043733
      Population  |  0.038941
       Education  |  0.031178
              US  |  0.005607
           Urban  |  0.005308
Top 3 features are: Price  ShelveLocMedium  Age
```

In [18]:
```python
ms = np.arange(1, 12)

for m in ms:

    rf_reg = RandomForestRegressor(random_state=0, max_features=m)
    rf_reg.fit(X_train, y_train);

    mse = mean_squared_error(y_test, rf_reg.predict(X_test))

    print("m: %2d" % m, " |  MSE: %.3f" % mse)
```

```
m:  1  |  MSE: 3.131
m:  2  |  MSE: 2.800
m:  3  |  MSE: 2.484
m:  4  |  MSE: 2.439
m:  5  |  MSE: 2.350
m:  6  |  MSE: 2.242
m:  7  |  MSE: 2.185
m:  8  |  MSE: 2.288
m:  9  |  MSE: 2.316
m: 10  |  MSE: 2.240
m: 11  |  MSE: 2.328
```

As m is increased, we increase model flexibility up to a certain point, after which the model begins to have trouble identifying the signal in the data, as seen by the increase in MSE after about m=~7.

Problem 3 (Chapter 8, Exercise 10)

In [19]:
```python
hitters = pd.read_csv("data/Hitters.csv")

print(hitters.info())
print(hitters.head())

# drop hitter names
hitters = hitters.drop("Unnamed: 0", axis=1)

# fix categorical columns
for col in ["League", "Division", "NewLeague"]:
  hitters[col] = hitters[col].astype('category').cat.codes
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 322 entries, 0 to 321
Data columns (total 21 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   Unnamed: 0  322 non-null    object 
 1   AtBat       322 non-null    int64  
 2   Hits        322 non-null    int64  
 3   HmRun       322 non-null    int64  
 4   Runs        322 non-null    int64  
 5   RBI         322 non-null    int64  
 6   Walks       322 non-null    int64  
 7   Years       322 non-null    int64  
 8   CAtBat      322 non-null    int64  
 9   CHits       322 non-null    int64  
 10  CHmRun      322 non-null    int64  
 11  CRuns       322 non-null    int64  
 12  CRBI        322 non-null    int64  
 13  CWalks      322 non-null    int64  
 14  League      322 non-null    object 
 15  Division    322 non-null    object 
 16  PutOuts     322 non-null    int64  
 17  Assists     322 non-null    int64  
 18  Errors      322 non-null    int64  
 19  Salary      263 non-null    float64
 20  NewLeague   322 non-null    object 
dtypes: float64(1), int64(16), object(4)
memory usage: 53.0+ KB
None
          Unnamed: 0  AtBat  Hits  HmRun  Runs  RBI  Walks  Years  CAtBat  \
0     -Andy Allanson    293    66      1    30   29     14      1     293   
1        -Alan Ashby    315    81      7    24   38     39     14    3449   
2       -Alvin Davis    479   130     18    66   72     76      3    1624   
3      -Andre Dawson    496   141     20    65   78     37     11    5628   
4  -Andres Galarraga    321    87     10    39   42     30      2     396   

   CHits  ...  CRuns  CRBI  CWalks  League Division PutOuts  Assists  Errors  
\
0     66  ...     30    29      14       A        E     446       33      20   
1    835  ...    321   414     375       N        W     632       43      10   
2    457  ...    224   266     263       A        W     880       82      14   
3   1575  ...    828   838     354       N        E     200       11       3   
4    101  ...     48    46      33       N        E     805       40       4   

   Salary  NewLeague  
0     NaN          A  
1   475.0          N  
2   480.0          A  
3   500.0          N  
4    91.5          N  

[5 rows x 21 columns]
```

Problem 3(a)

In [20]:
```
hitters = hitters.dropna()
hitters.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 263 entries, 1 to 321
Data columns (total 20 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   AtBat      263 non-null    int64  
 1   Hits       263 non-null    int64  
 2   HmRun      263 non-null    int64  
 3   Runs       263 non-null    int64  
 4   RBI        263 non-null    int64  
 5   Walks      263 non-null    int64  
 6   Years      263 non-null    int64  
 7   CAtBat     263 non-null    int64  
 8   CHits      263 non-null    int64  
 9   CHmRun     263 non-null    int64  
 10  CRuns      263 non-null    int64  
 11  CRBI       263 non-null    int64  
 12  CWalks     263 non-null    int64  
 13  League     263 non-null    int8   
 14  Division   263 non-null    int8   
 15  PutOuts    263 non-null    int64  
 16  Assists    263 non-null    int64  
 17  Errors     263 non-null    int64  
 18  Salary     263 non-null    float64
 19  NewLeague  263 non-null    int8   
dtypes: float64(1), int64(16), int8(3)
memory usage: 37.8 KB
```

In [21]:
```python
hitters.head()
```

Out[21]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWa
1	315	81	7	24	38	39	14	3449	835	69	321	414	
2	479	130	18	66	72	76	3	1624	457	63	224	266	
3	496	141	20	65	78	37	11	5628	1575	225	828	838	
4	321	87	10	39	42	30	2	396	101	12	48	46	
5	594	169	4	74	51	35	11	4408	1133	19	501	336	

In [22]:
```python
hitters["Salary"] = np.log(hitters["Salary"])
hitters.head()
```

Out[22]:

	AtBat	Hits	HmRun	Runs	RBI	Walks	Years	CAtBat	CHits	CHmRun	CRuns	CRBI	CWa
1	315	81	7	24	38	39	14	3449	835	69	321	414	
2	479	130	18	66	72	76	3	1624	457	63	224	266	
3	496	141	20	65	78	37	11	5628	1575	225	828	838	
4	321	87	10	39	42	30	2	396	101	12	48	46	
5	594	169	4	74	51	35	11	4408	1133	19	501	336	

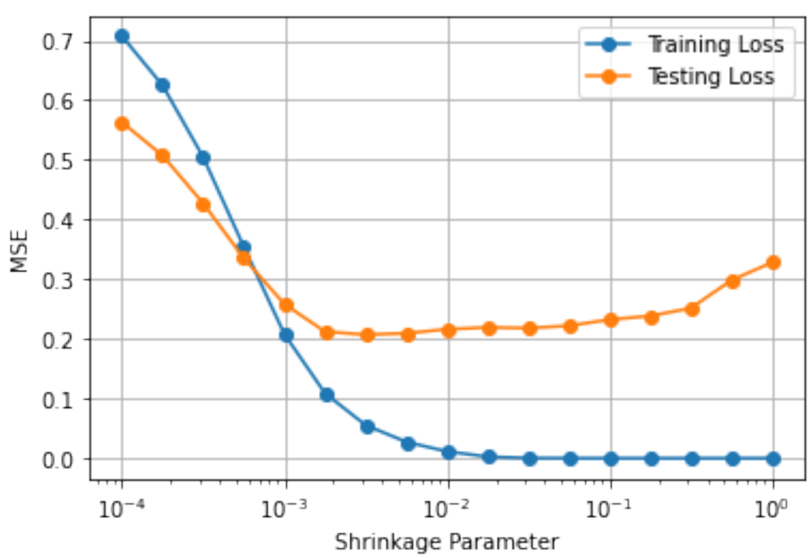
Problem 3(b)

In [23]:
```
X = hitters.drop("Salary", axis=1)
y = hitters["Salary"]

X_train = X.iloc[:200, :]
y_train = y.iloc[:200]
X_test  = X.iloc[200:, :]
y_test  = y.iloc[200:]
```

Problem 3(c) & 3(d)

In [24]:
```
from sklearn.ensemble import GradientBoostingRegressor

lrs = np.logspace(-4, 0, 17)
train_mse_hist = []
test_mse_hist  = []

for lr in lrs:

    gb_reg = GradientBoostingRegressor(random_state=0, learning_rate=lr, n_es
    gb_reg.fit(X_train, y_train)

    test_mse = mean_squared_error(y_test, gb_reg.predict(X_test))

    train_mse_hist.append(gb_reg.train_score_[-1])
    test_mse_hist. append(test_mse)
```

In [25]:
```
plt.plot(lrs, train_mse_hist, marker="o", label="Training Loss")
plt.plot(lrs, test_mse_hist,  marker="o", label="Testing Loss")
plt.xlabel("Shrinkage Parameter")
plt.ylabel("MSE")
plt.xscale("log")
plt.legend()
plt.grid()
```



Problem 3(e)

In [26]:
```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

mse = mean_squared_error(y_test, lin_reg.predict(X_test))

print("gradient boosting test MSE: %.3f" % np.min(test_mse_hist))
print("linear regression test MSE: %.3f" % mse)
```

```
gradient boosting test MSE: 0.207
linear regression test MSE: 0.492
```

Problem 3(f)

In [27]:
```
gb_reg = GradientBoostingRegressor(random_state=0, learning_rate=lrs[np.argmi
                                   n_estimators=1000, criterion="mse")
gb_reg.fit(X_train, y_train)

idxs = np.argsort(gb_reg.feature_importances_)[::-1]

for col, imp in zip(X.columns[idxs], gb_reg.feature_importances_[idxs]):
    print("%15s" %col, " |  %3f" %imp)

print("Top 3 features are:", X.columns[idxs][0], "", X.columns[idxs][1], "", 
```

```
         CAtBat  |  0.558630
          CHits  |  0.095260
          CRuns  |  0.053545
          AtBat  |  0.049422
          Walks  |  0.045620
           CRBI  |  0.038976
         CWalks  |  0.034677
         CHmRun  |  0.029765
          Years  |  0.023699
           Hits  |  0.022652
            RBI  |  0.016421
        PutOuts  |  0.011897
           Runs  |  0.010749
         Errors  |  0.003875
        Assists  |  0.002180
          HmRun  |  0.001302
      NewLeague  |  0.000923
         League  |  0.000211
       Division  |  0.000196
Top 3 features are: CAtBat  CHits  CRuns
```

Problem 3(g)

In [28]:
```
bag_reg = BaggingRegressor(DecisionTreeRegressor(random_state=0), random_stat
bag_reg.fit(X_train, y_train);

mse = mean_squared_error(y_test, bag_reg.predict(X_test))

print("gradient boosting test MSE: %.3f" % np.min(test_mse_hist))
print("bagging test MSE:           %.3f" % mse)
```

```
gradient boosting test MSE: 0.207
bagging test MSE:           0.258
```

Problem 4 (Chapter 8, Exercise 11)

In [29]:
```
caravan = pd.read_csv("data/Caravan.csv")
```

Problem 4(a)

In [30]:
```
print(caravan.head())

# fix categorical column
enc = OneHotEncoder(categories=[["Yes", "No"]], sparse=False)
caravan["Purchase"] = enc.fit_transform(caravan["Purchase"].to_numpy().reshap
```

```
   MOSTYPE  MAANTHUI  MGEMOMV  MGEMLEEF  MOSHOOFD  MGODRK  MGODPR  MGODOV  \
0       33         1        3         2         8       0       5       1
1       37         1        2         2         8       1       4       1
2       37         1        2         2         8       0       4       2
3        9         1        3         3         3       2       3       2
4       40         1        4         2        10       1       4       1

   MGODGE  MRELGE  ...  APERSONG  AGEZONG  AWAOREG  ABRAND  AZEILPL  APLEZIER
\
0       3       7  ...         0        0        0       1        0         0
1       4       6  ...         0        0        0       1        0         0
2       4       3  ...         0        0        0       1        0         0
3       4       5  ...         0        0        0       1        0         0
4       4       7  ...         0        0        0       1        0         0

   AFIETS  AINBOED  ABYSTAND  Purchase
0       0        0         0        No
1       0        0         0        No
2       0        0         0        No
3       0        0         0        No
4       0        0         0        No

[5 rows x 86 columns]
```

In [31]:
```
X = caravan.drop("Purchase", axis=1)
y = caravan["Purchase"]

X_train = X.iloc[:1000, :]
y_train = y.iloc[:1000]
X_test  = X.iloc[1000:, :]
y_test  = y.iloc[1000:]
```

Problem 4(b)

In [32]:
```python
from sklearn.ensemble import GradientBoostingClassifier

gb_clf = GradientBoostingClassifier(random_state=0, learning_rate=0.01, n_est
gb_clf.fit(X_train, y_train)

idxs = np.argsort(gb_clf.feature_importances_)[::-1]

print("First ten features by importance:")
for col, imp in zip(X.columns[idxs][:10], gb_clf.feature_importances_[idxs][:
    print("%15s" %col, " |  %3f" %imp)

print("Top 3 features are:", X.columns[idxs][0], "", X.columns[idxs][1], "", 
```

```
First ten features by importance:
       PPERSAUT  |  0.074507
        MOSTYPE  |  0.065550
         ABRAND  |  0.056751
         MGODGE  |  0.052928
       MKOOPKLA  |  0.047574
       MOPLHOOG  |  0.045789
       MBERMIDD  |  0.040154
         MGODPR  |  0.032920
       PPLEZIER  |  0.031407
         PBRAND  |  0.031169
Top 3 features are: PPERSAUT  MOSTYPE  ABRAND
```

Problem 4(c)

In [33]:
```python
y_pred = gb_clf.predict_proba(X_test)[:, 1]
y_pred = y_pred > 0.2
```

In [34]:
```python
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[4336  197]
 [ 251   38]]
```

In [35]:
```python
tn, fp, fn, tp = cm.ravel()

gb_precision = tp / (tp + fp)

print("gradient boosting precision: %.3f" % gb_precision)
```

```
gradient boosting precision: 0.162
```

In [36]:
```
from sklearn.linear_model import LogisticRegression

log_reg = LogisticRegression(random_state=0, max_iter=1E6)
log_reg.fit(X_train, y_train)

y_pred = log_reg.predict_proba(X_test)[:, 1]
y_pred = y_pred > 0.2

cm = confusion_matrix(y_test, y_pred)
print(cm)

tn, fp, fn, tp = cm.ravel()

log_reg_precision = tp / (tp + fp)
```

```
[[4293  240]
 [ 239   50]]
```

In [37]:
```
print("gradient boosting precision:   %.3f" % gb_precision)
print("logistic regression precision: %.3f" % log_reg_precision)
```

```
gradient boosting precision:   0.162
logistic regression precision: 0.172
```

Gradient boosting does slightly better on the specified precision metric.