Problem 3 (Chapter 6, Exercise 9, excluding parts (e), (f), and (g))

In [1]:
```
import pandas as pd

college = pd.read_csv("data/College.csv")

college = college.drop(columns=['Unnamed: 0'])
college['Private'] = college['Private'].astype('category')
college['Private'] = college['Private'].cat.codes
```

Problem 3(a)

In [2]:
```
from sklearn.model_selection import train_test_split

X = college.drop(columns=['Apps'])
y = college['Apps']

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

Problem 3(b)

In [3]:
```
import statsmodels.api as sm
import numpy as np

X_train = sm.add_constant(X_train)
X_test  = sm.add_constant(X_test)

linear = sm.regression.linear_model.OLS(y_train, X_train).fit()

y_pred = linear.predict(X_test)
test_mse = np.mean((y_pred - y_test)**2)

print(test_mse)
```

```
1261087.1761283777
```

Problem 3(c)

In [4]:
```python
from sklearn import linear_model
from sklearn.model_selection import cross_validate

lambdas           = np.logspace(-6, 6, 13)
avg_test_mse_hist = []

for lam in lambdas:

    ridge = linear_model.Ridge(alpha=lam)

    cv_results = cross_validate(ridge, X, y, cv=10, scoring='neg_mean_squared_

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

    # print(lam, avg_test_mse)

min_idx = np.argmin(avg_test_mse_hist)
print("best avg test mse:    ", avg_test_mse_hist[min_idx])
print("corresponding lambda:", lambdas[min_idx])
```

```
best avg test mse:    1287421.609196762
corresponding lambda: 10.0
```

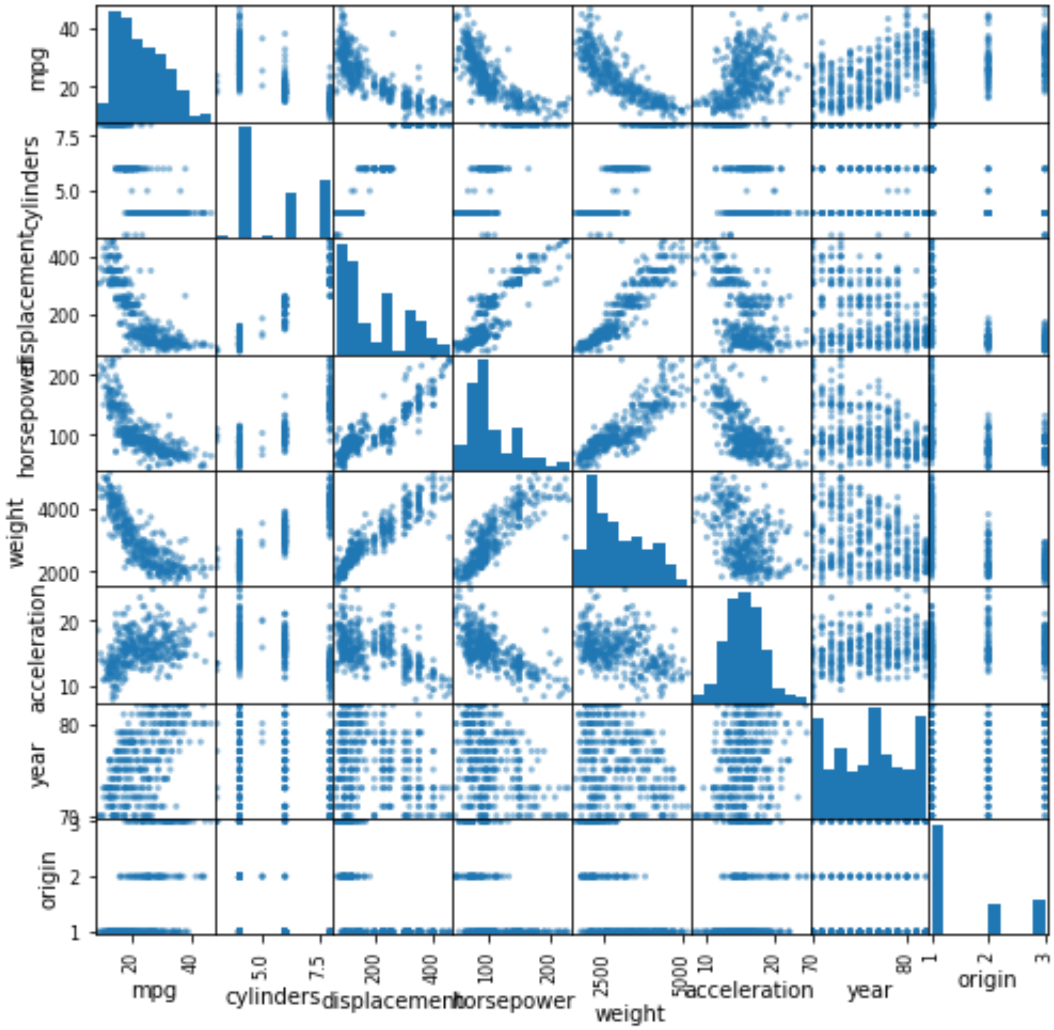
Problem 3(d)

In [5]:
```python
lambdas           = np.logspace(-6, 6, 13)
avg_test_mse_hist = []

for lam in lambdas:

    lasso = linear_model.Lasso(alpha=lam)

    cv_results = cross_validate(lasso, X, y, cv=10, scoring='neg_mean_squared_

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

    # print(lam, avg_test_mse)

min_idx = np.argmin(avg_test_mse_hist)
print("best avg test mse:    ", avg_test_mse_hist[min_idx])
print("corresponding lambda:", lambdas[min_idx])

lasso_opt = linear_model.Lasso(alpha=lambdas[min_idx]).fit(X_train, y_train)
print("num. nonzero coeffs.:", len(lasso_opt.coef_.nonzero()[0]))
```

```
best avg test mse:    1288186.8283163894
corresponding lambda: 1.0
num. nonzero coeffs.: 17
```

Problem 5 (Chapter 7, Exercise 8)

In [6]:
```
auto = pd.read_csv("data/Auto.csv")

auto = auto.drop(columns='name')

pd.plotting.scatter_matrix(auto, figsize=(8,8));
```



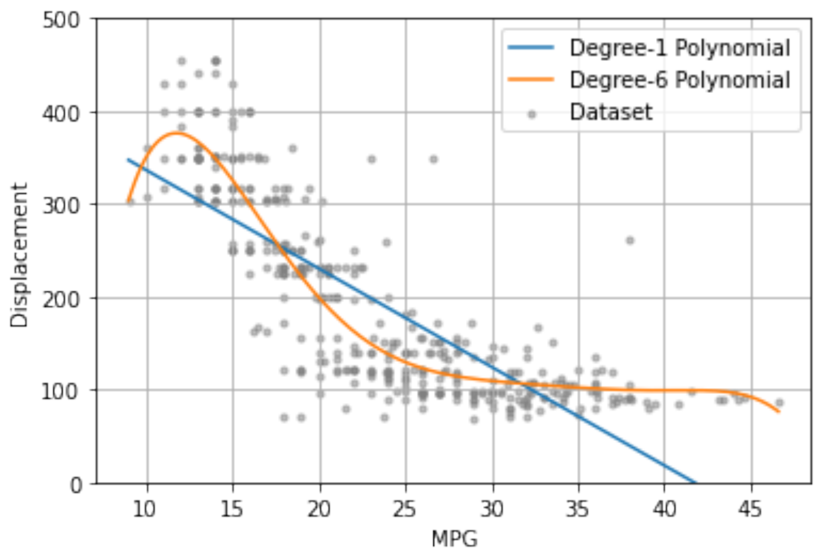
Looks like there are some nonlinear relationships for mpg and displacement predictors. We'll try to fit mpg against displacement. We'll fit using polynomial transformations of the data (which will include linear regression).

In [7]:
```
from sklearn.preprocessing import PolynomialFeatures

X = np.array(auto["mpg"]).reshape(-1, 1)
y = auto["displacement"]

ks                = np.arange(1, 10)
avg_test_mse_hist = []

for k in ks:

    poly    = linear_model.LinearRegression()

    X_poly = PolynomialFeatures(degree=k, include_bias=False).fit_transform(X

    cv_results = cross_validate(poly, X_poly, y, cv=10, scoring='neg_mean_squ

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

    print(k, avg_test_mse)

min_idx = np.argmin(avg_test_mse_hist)
print("best avg test mse:    ", avg_test_mse_hist[min_idx])
print("corresponding degree:", ks[min_idx])
```

```
1 4235.632489029972
2 2581.1908702267283
3 2657.3874258934275
4 2562.680813891059
5 2449.084689600645
6 2429.726587020797
7 2458.309737199786
8 2650.559119261322
9 2754.904014448882
best avg test mse:    2429.726587020797
corresponding degree: 6
```

We see that among degree 1-10 polynomial expansions of the mpg predictor, that the linear regression (degree 1) has the highest (worst) CV error and that the degree 6 polynomial has the lowest (best) CV error.

In [8]:
```python
import matplotlib.pyplot as plt

X_train, X_test, y_train, y_test = train_test_split(X, y)
X_plot = np.linspace(X.min(), X.max(), 200).reshape(-1, 1)

plt.figure()
plt.scatter(X, y, c="gray", alpha=0.5, s=10, label="Dataset")

for k in [1, 6]:

    X_poly_train = PolynomialFeatures(degree=k, include_bias=False).fit_trans
    poly         = linear_model.LinearRegression().fit(X_poly_train, y_train)

    X_plot_poly  = PolynomialFeatures(degree=k, include_bias=False).fit_trans
    y_plot_poly  = poly.predict(X_plot_poly)

    plt.plot(X_plot, y_plot_poly, label="Degree-%d Polynomial" %k)

plt.ylim(0, 500)
plt.xlabel("MPG")
plt.ylabel("Displacement")
plt.legend()
plt.grid()
```



Problem 7 (Chapter 9, Exercise 8)

In [9]:
```python
oj = pd.read_csv("data/OJ.csv")

oj["Purchase"] = oj["Purchase"].astype('category').cat.codes
oj["Store7"]   = oj["Store7"].  astype('category').cat.codes
```

Problem 7(a)

```
In [10]: X = oj.drop(columns="Purchase")
         y = oj["Purchase"]

         X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=800/X.sh

         print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(800, 17) (270, 17) (800,) (270,)
```

Problem 7(b)

```
In [11]: from sklearn.svm import SVC

         svc_linear = SVC(C=0.01, kernel="linear").fit(X_train, y_train)

         print("fit status:                   ", "successful" if svc_linear.fit_status_ ==
         print("number of support vectors:", svc_linear.n_support_)
         print("fit primal intercept:      ", svc_linear.intercept_)
         print("fit primal coefficients:   ", svc_linear.coef_)
```

```
fit status:                   successful
number of support vectors: [314 312]
fit primal intercept:      [3.2709307]
fit primal coefficients:   [[-0.00774799 -0.16455324  0.02911706 -0.06028802 -
0.02391942  0.15750947
  -0.04        0.21571864 -1.04794193 -0.21779748  0.05303648 -0.27083396
  -0.01872651  0.07113893 -0.01269375 -0.08940508 -0.03346765]]
```

Problem 7(c)

```
In [12]: train_error_rate = np.mean(np.abs(y_train - svc_linear.predict(X_train)))
         test_error_rate  = np.mean(np.abs(y_test  - svc_linear.predict(X_test )))

         print("train error rate:", train_error_rate)
         print("test error rate: ", test_error_rate )
```

```
train error rate: 0.20875
test error rate:  0.21851851851851853
```

Problem 7(d)

In [13]:

```python
from warnings import filterwarnings
filterwarnings('ignore')

cs                       = np.logspace(-2, 1, 10)
avg_test_error_rate_hist = []

for c in cs:

    svc_linear = SVC(C=c, kernel="linear", max_iter=1E6)

    cv_results = cross_validate(svc_linear, X, y, cv=10, scoring='accuracy')

    avg_test_error_rate = np.mean(1 - cv_results['test_score'])
    avg_test_error_rate_hist.append(avg_test_error_rate)

    print("%.3f" % c, avg_test_error_rate)

min_idx = np.argmin(avg_test_error_rate_hist)
print("best avg test error rate:", avg_test_error_rate_hist[min_idx])
print("corresponding cost:      ", cs[min_idx])
```

```
0.010 0.22523364485981306
0.022 0.19813084112149534
0.046 0.18037383177570096
0.100 0.18411214953271032
0.215 0.17570093457943928
0.464 0.17102803738317757
1.000 0.1738317757009346
2.154 0.17570093457943928
4.642 0.2018691588785047
10.000 0.26728971962616827
best avg test error rate: 0.17102803738317757
corresponding cost:       0.46415888336127775
```

Problem 7(e)

In [14]:

```python
svc_linear = SVC(C=cs[min_idx], kernel="linear").fit(X_train, y_train)

train_error_rate = np.mean(np.abs(y_train - svc_linear.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_linear.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.16375
test error rate:  0.16666666666666666
```

Problem 7(f)

In [15]:

```python
svc_rbf = SVC(C=0.01, kernel="rbf").fit(X_train, y_train)

print("fit status:               ", "successful" if svc_rbf.fit_status_ == 0
print("number of support vectors:", svc_rbf.n_support_)
print("fit primal intercept:     ", svc_rbf.intercept_)
```

```
fit status:                  successful
number of support vectors: [316 316]
fit primal intercept:      [-0.99956289]
```

In [16]:

```
train_error_rate = np.mean(np.abs(y_train - svc_rbf.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_rbf.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.395
test error rate:  0.37407407407407406
```

In [17]:

```
cs                      = np.logspace(2, 6, 13)
avg_test_error_rate_hist = []

for c in cs:

    svc_rbf    = SVC(C=c, kernel="rbf", max_iter=1E6)

    cv_results = cross_validate(svc_rbf, X, y, cv=10, scoring='accuracy')

    avg_test_error_rate = np.mean(1 - cv_results['test_score'])
    avg_test_error_rate_hist.append(avg_test_error_rate)

    print("%.3f" % c, avg_test_error_rate)

min_idx = np.argmin(avg_test_error_rate_hist)
print("best avg test error rate:", avg_test_error_rate_hist[min_idx])
print("corresponding cost:      ", cs[min_idx])
```

```
100.000 0.3766355140186916
215.443 0.2738317757009346
464.159 0.20560747663551404
1000.000 0.18504672897196267
2154.435 0.18224299065420563
4641.589 0.17476635514018693
10000.000 0.1728971962616822б
21544.347 0.17196261682242991
46415.888 0.1766355140186916
100000.000 0.17009345794392522
215443.469 0.1738317757009346
464158.883 0.1738317757009346
1000000.000 0.17289719626168226
best avg test error rate: 0.17009345794392522
corresponding cost:       100000.0
```

In [18]:

```
svc_rbf = SVC(C=cs[min_idx], kernel="rbf").fit(X_train, y_train)

train_error_rate = np.mean(np.abs(y_train - svc_rbf.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_rbf.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.15875
test error rate:  0.18148148148148148
```

Problem 7(g)

In [19]:
```
svc_poly = SVC(C=0.01, kernel="poly", degree=2).fit(X_train, y_train)

print("fit status:                 ", "successful" if svc_poly.fit_status_ == 0
print("number of support vectors:", svc_poly.n_support_)
print("fit primal intercept:      ", svc_poly.intercept_)
```

```
fit status:                 successful
number of support vectors: [316 316]
fit primal intercept:      [-0.99748713]
```

In [20]:
```
train_error_rate = np.mean(np.abs(y_train - svc_poly.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_poly.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.395
test error rate:  0.37407407407407406
```

In [21]:
```
cs                       = np.logspace(2, 6, 13)
avg_test_error_rate_hist = []

for c in cs:

    svc_poly    = SVC(C=c, kernel="poly", degree=2, max_iter=1E6)

    cv_results = cross_validate(svc_poly, X, y, cv=10, scoring='accuracy')

    avg_test_error_rate = np.mean(1 - cv_results['test_score'])
    avg_test_error_rate_hist.append(avg_test_error_rate)

    print("%.3f" % c, avg_test_error_rate)

min_idx = np.argmin(avg_test_error_rate_hist)
print("best avg test error rate:", avg_test_error_rate_hist[min_idx])
print("corresponding cost:       ", cs[min_idx])
```

```
100.000 0.38130841121495335
215.443 0.3074766355140187
464.159 0.20934579439252338
1000.000 0.1869158878504673
2154.435 0.18224299065420563
4641.589 0.1766355140186916
10000.000 0.17476635514018693
21544.347 0.17196261682242991
46415.888 0.17102803738317757
100000.000 0.17383177570093458
215443.469 0.22336448598130837
464158.883 0.202803738317757
1000000.000 0.25046728971962623
best avg test error rate: 0.17102803738317757
corresponding cost:       46415.888336127726
```

In [22]:
```
svc_poly = SVC(C=cs[min_idx], kernel="poly", degree=2).fit(X_train, y_train)

train_error_rate = np.mean(np.abs(y_train - svc_poly.predict(X_train)))
test_error_rate  = np.mean(np.abs(y_test  - svc_poly.predict(X_test )))

print("train error rate:", train_error_rate)
print("test error rate: ", test_error_rate )
```

```
train error rate: 0.16375
test error rate:  0.15925925925925927
```

Problem 7(h)

The radial basis function SVC seems to give the best results on this dataset as it achieves the lowest cross-validation test error rate of 17.01% misclassification.