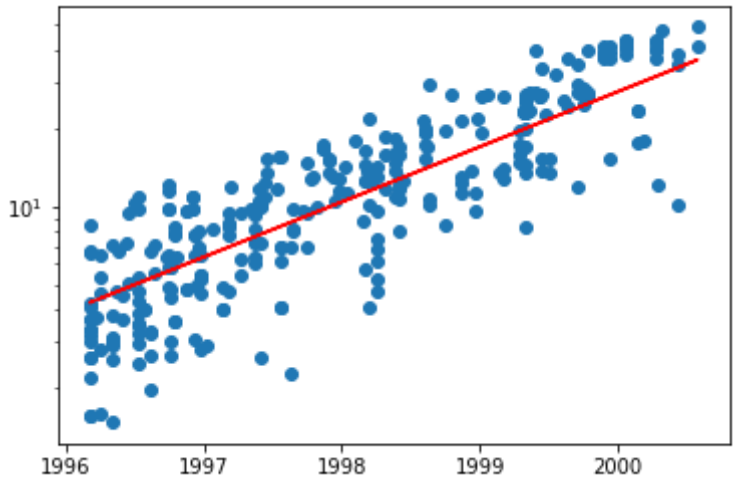
In [ ]:
```
# Moore's law
```

In [2]:
```
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from datetime import datetime
from math import log, exp
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random

random.seed(1)

data = pd.read_csv("benchmarks.txt")

# obtain list of all 'benchName' in descending order by count
bench_names = data.groupby(["benchName"]).count().index.tolist()

selected_bench_names = random.sample(bench_names, 3)
for selected_bench_name in selected_bench_names:
    bench_data = data[data["benchName"] == selected_bench_name]
    X = []
    y = []
    for test_id, base_performance in zip(bench_data["testID"], bench_dat
a["base"]):
        try:
            test_date = test_id.split('-')[1]
            if len(test_date) == 6:
                test_date = '19' + test_date
            test_date = datetime.strptime(test_date, '%Y%m%d')
            test_date = test_date.year + (test_date.month)/12 + (test_da
te.day)/365
            X.append(test_date)
            y.append(base_performance)
        except IndexError:
            pass
    lm = linear_model.LinearRegression()
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=
300, random_state=1)
    X_train = np.array(X_train).reshape(len(X_train),-1)
    X_test = np.array(X_test).reshape(len(X_test),-1)
    y_train_log = [log(num) for num in y_train]
    y_test_log = [log(num) for num in y_test]
    lm.fit(X_train,y_train_log)
    predicted_y_test_log = lm.predict(X_test)
    predicted_y_test = [exp(num) for num in predicted_y_test_log]
    print ("- For bench", selected_bench_name)
    plt.scatter(X_test,y_test)
    plt.plot(X_test,predicted_y_test, "r")
    plt.yscale('log')
    plt.show()
    print("Predicted coefficient:", lm.coef_[0], "\n")
    print ("Training set accuracy:", lm.score(X_train,y_train_log))
    print ("Test set accuracy:", lm.score(X_test,y_test_log), "\n\n")
```

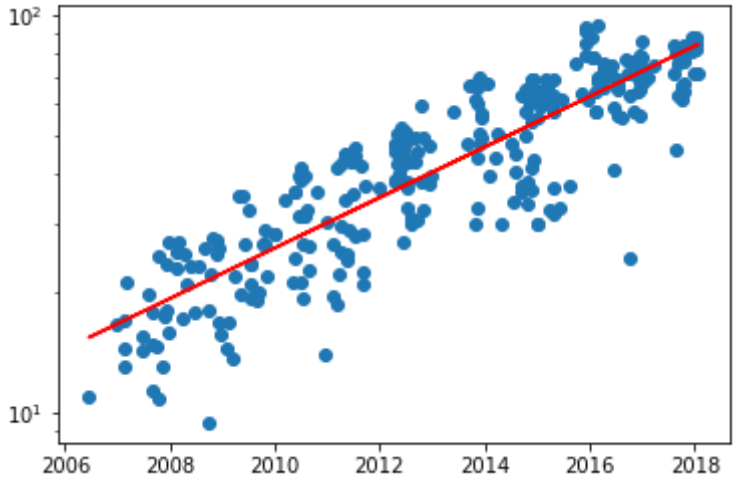
```
/Library/Frameworks/Python.framework/Versions/3.6/lib/python3.6/site-pa
ckages/IPython/core/interactiveshell.py:2785: DtypeWarning: Columns (3)
have mixed types. Specify dtype option on import or set low_memory=Fals
e.
  interactivity=interactivity, compiler=compiler, result=result)
```

- For bench 147.vortex



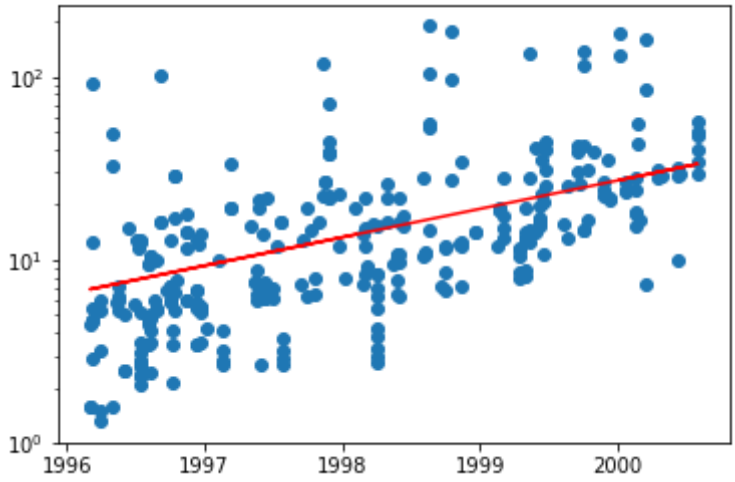

```
Predicted coefficient: 0.48811966537504897

Training set accuracy: 0.6956216871244124
Test set accuracy: 0.7261213081813807
```

- For bench 483.xalancbmk




```
Predicted coefficient: 0.14638029671365446

Training set accuracy: 0.7514623935717057
Test set accuracy: 0.7824379130679922
```

- For bench 125.turb3d

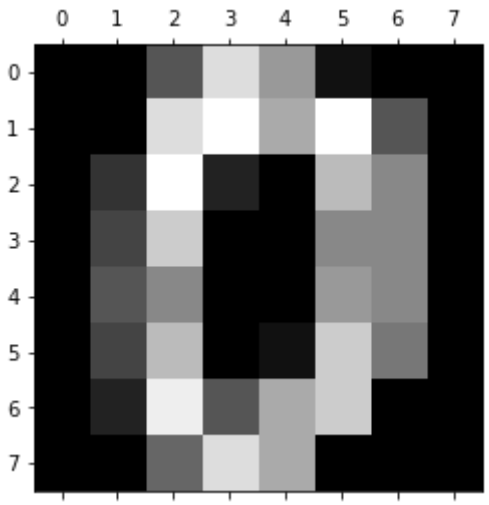


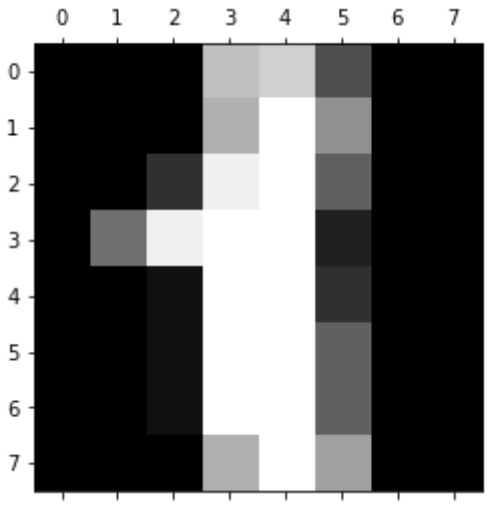

```
Predicted coefficient: 0.3565191166537102

Training set accuracy: 0.3006079678599781
Test set accuracy: 0.33478016794026433
```

In [ ]:
```
# MNIST Digits
```

In [3]:
```
from sklearn.datasets import load_digits
import matplotlib.pyplot as plt

# load MNIST dataset
digits = load_digits()
print("Dimensions of data:", digits.data.shape)

plt.matshow(digits.images[0])
plt.gray()
plt.matshow(digits.images[1])
plt.gray()
plt.show()
```

```
Dimensions of data: (1797, 64)
```





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

x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.
target, test_size=0.25, random_state=0)
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(x_train, y_train)

y_prediction = knn.predict(x_test)

print ("Score:", knn.score(x_test, y_test))
print ("MSE:", (((y_prediction-y_test)**2).sum()) / len(y_prediction))
```

```
Score: 0.9866666666666667
MSE: 0.20444444444444446
```

In [10]:
```
true_postive = dict()
false_postive = dict()
false_negative = dict()

for real, predict in zip(y_test, y_prediction):
    if real == predict:
        true_postive.setdefault(real, 0)
        false_postive.setdefault(real, 0)
        false_negative.setdefault(real, 0)
        true_postive[real] += 1
    elif real != predict:
        true_postive.setdefault(real, 0)
        true_postive.setdefault(predict, 0)
        false_postive.setdefault(real, 0)
        false_postive.setdefault(predict, 0)
        false_negative.setdefault(real, 0)
        false_negative.setdefault(predict, 0)
        false_negative[real] += 1
        false_postive[predict] += 1

d = {'number': [], 'precision': [], 'recall': [], 'true_pos': [], 'false
_pos': [], 'false_neg': []}
for number, tp in sorted(true_postive.items(), key=lambda x: x[0]):
    d['number'].append(number)
    d['precision'].append(np.divide(tp, (tp + false_postive[number])))
    d['recall'].append(np.divide(tp, (tp + false_negative[number])))
    d['true_pos'].append(tp)
    d['false_pos'].append(false_postive[number])
    d['false_neg'].append(false_negative[number])

df = pd.DataFrame(d)
df = df.set_index('number')
print (df)
```

```
        precision    recall  true_pos  false_pos  false_neg
number
0        1.000000  1.000000        37          0          0
1        1.000000  0.976744        42          0          1
2        0.977778  1.000000        44          1          0
3        0.956522  0.977778        44          2          1
4        1.000000  0.973684        37          0          1
5        0.979167  0.979167        47          1          1
6        1.000000  1.000000        52          0          0
7        0.979592  1.000000        48          1          0
8        1.000000  0.958333        46          0          2
9        0.979167  1.000000        47          1          0
```