Assignment 1: Moore's Law and MNIST Digits

1. Moore's Law

Use the scripts from here to download a large amount of data relating to CPU specs. The scripts might take as long as an hour, depending on your connection speed. (Pay attention to the line "If you want to skip the steps in this section, you can simply download the aggregated result files from http://preshing.com/files/specdata20120207.zip and extract them to this folder." This will be faster and save you some troubles while providing the same dataset.)

This will save the data in the following format:

testid	benchName	base	peak
cpu95-19990104-03254	101.tomcatv	19.4	27.1
cpu95-19990104-03254	102.swim	27.2	34.8
cpu95-19990104-03254	103.su2cor	10.1	9.98
cpu95-19990104-03254	104.hydro2d	8.58	8.61

Now do the following:

1. Extract the date and base speed for a benchmark of your choice. Note that the dates contained as part of the testID don't tell us about when the hardware was actually designed, so the test could have been run at a much later date using older hardware. We therefore need the date indicating when the hardware was first available (hwAvail) from the summaries file to really test Moore's Law.
2. Plot the data in a semi-log plot
3. Now train a linear model to fit your plot.
4. How well is Moore’s law holding up?

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
import sklearn.metrics as metrics
import datetime as dt
```

In [2]:
```
# Load Data and pre-processing

foo = pd.read_csv("benchmarks.txt", dtype = str)
foo['base'] = pd.to_numeric(foo['base'])

# Using summary.txt data to get date of manufacture
summary = pd.read_csv('summaries.txt', encoding = 'unicode-escape')

new = foo['testID'].str.split("-", n =2, expand = True)

# conversions to date time
foo['test-date'] = new[1]
foo['test-date'] = pd.to_datetime(foo['test-date'])
```

```
summary['hwAvail'] = pd.to_datetime(summary['hwAvail'])

foo.head(n=3)
```

Out[2]:

	testID	benchName	base	peak	test-date
0	cpu95-19990104-03254	101.tomcatv	19.4	27.1	1999-01-04
1	cpu95-19990104-03254	102.swim	27.2	34.8	1999-01-04
2	cpu95-19990104-03254	103.su2cor	10.1	9.98	1999-01-04

In [3]:

```
# Choose bench name
zeus = foo[foo['benchName'] == '434.zeusmp']
zeus.name = "Zeus Data Set"

# We use the date in summary.txt specifically in hwAvail
# since it has the date of manufacturing not the testing date
# The reason for this addition is to have a more accurate Moore's Law test
zeus_testID = np.array(zeus['testID'])
manf_date = []

for testID in zeus_testID:
    ID = summary[summary['testID'] == testID]
    date = ID['hwAvail'].iloc[0]

    manf_date.append(date)
```

In [4]:

```
zeus['manf-date'] = manf_date
zeus.head(n = 3)
```

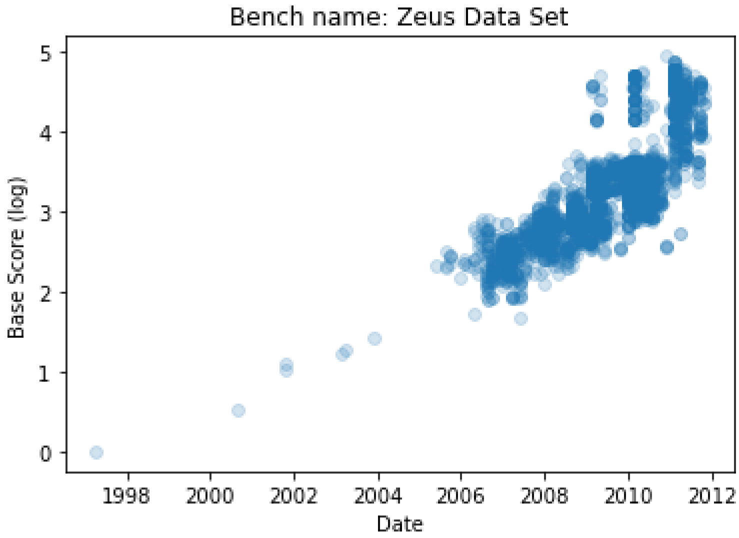
```
<ipython-input-4-6e4df2e95749>:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/st
able/user_guide/indexing.html#returning-a-view-versus-a-copy
  zeus['manf-date'] = manf_date
```

Out[4]:

	testID	benchName	base	peak	test-date	manf-date
45843	cpu2006-20060513-00002	434.zeusmp	1.00	NaN	2006-05-13	1997-04-01
45884	cpu2006-20060513-00009	434.zeusmp	8.92	8.92	2006-05-13	2006-01-01
45913	cpu2006-20060513-00013	434.zeusmp	10.10	NaN	2006-05-13	2006-05-01

In [5]:

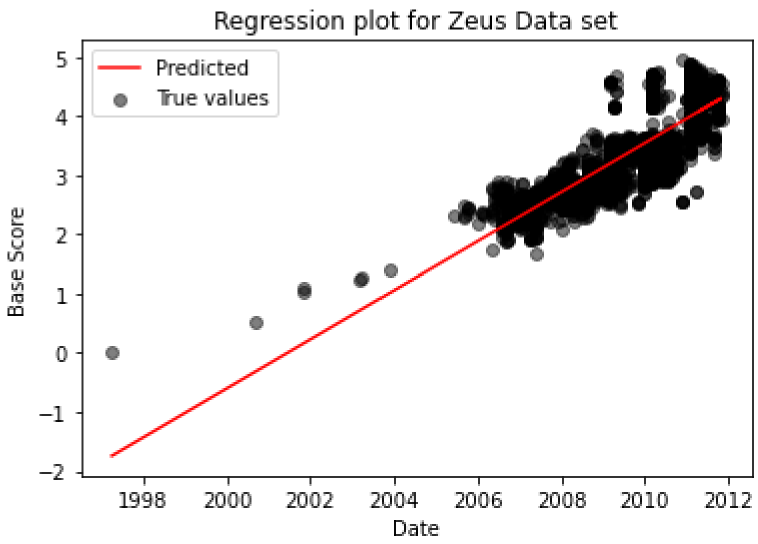
```
plt.scatter(zeus['manf-date'], np.log(zeus['base']), alpha = 0.2)
plt.xlabel("Date")
plt.ylabel("Base Score (log)")
plt.title(f"Bench name: {zeus.name}")
plt.show()
```



In [6]:

```
# Assign variables
X = zeus['manf-date']
y = zeus['base']

# Keeping date structure
X_date = zeus['manf-date']

# Changing date time to ordinal so we can use it for prediction in the regres
X = X.map(dt.datetime.toordinal)

# Reshape data
def change_dim(data):
    series_obj = pd.Series(data)

    data = series_obj.values
    return data.reshape((len(data),1))

X = change_dim(X)
y = change_dim(y)

# log variable
y = np.log(y)

# Create model
lm = linear_model.LinearRegression()

fit = lm.fit(X,y)
predicted = lm.predict(X)

# The metrics
def get_metrics(X, y, pred):
    coef = lm.coef_
    intercept = lm.intercept_
    MSE = metrics.mean_squared_error(y, pred)
    RMSE = metrics.mean_squared_error(y, pred, squared = False)
    R2 = metrics.r2_score(y, pred)

    measures = ['Coefficient', 'Intercept', 'Mean Squared Error',
                'Root Mean Squared Error', 'R-squared']

    metric_results = [coef[0][0], intercept[0], MSE, RMSE, R2]

    return pd.DataFrame(data = {'Metrics' : measures, 'Results' : metric_resu
```

```
# Plot regression

plt.scatter(X_date, y, color = 'black', label = 'True values', alpha = 0.5)
plt.plot(X_date, predicted, color = 'red', label = 'Predicted')
plt.title("Regression plot for Zeus Data set")
plt.xlabel("Date")
plt.ylabel("Base Score")
plt.legend()
plt.show()

metric_results = get_metrics(X, y, predicted)
metric_results
```



Out[6]:

	Metrics	Results
0	Coefficient	0.001135
1	Intercept	-829.631271
2	Mean Squared Error	0.157131
3	Root Mean Squared Error	0.396398
4	R-squared	0.685014

Answer

Moore's Law still holds. Based on the plot, we can see an increasing linear trend following the logged data points. Additionally. R-squared shows an good measure of 0.69. MSE seems to be very low as well, making our model plot a good fit for the data.

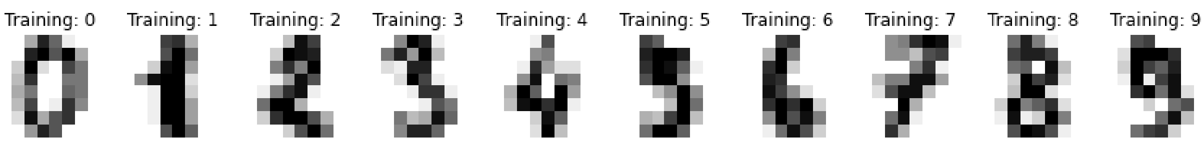
2. MNIST Digits

No machine learning course would be complete without using the MNIST dataset. This dataset was a hugely influential dataset of handwriting digits (0-9).

- Using scikit-learn, load the MNIST digits (see here).
- Plot some of the examples.
- Choose two digit classes (e.g. 7s and 3s), and train a k-nearest neighbor classifier.
- Report your error rates on a held out part of the data.
- (*optional*) Test your model on the full dataset (available from here).

In [1]:
```
# Standard scientific Python imports
import matplotlib.pyplot as plt

# Import datasets, classifiers and performance metrics
from sklearn import datasets, svm, metrics
from sklearn.model_selection import train_test_split

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib.colors import ListedColormap
from sklearn import neighbors, datasets

# Load data set
digits = datasets.load_digits()
```

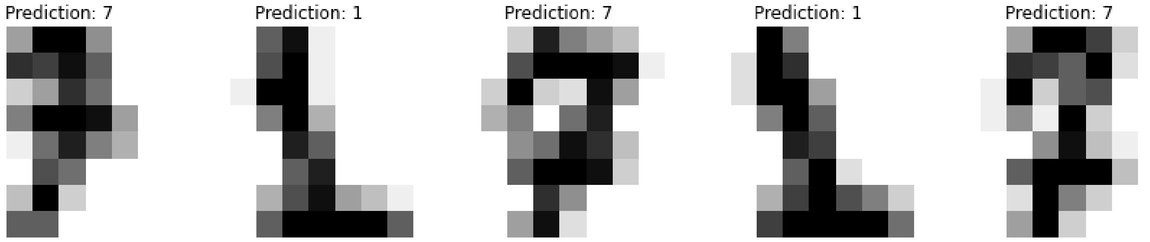
In [2]:
```
# Plotting some examples

_, axes = plt.subplots(nrows=1, ncols=10, figsize=(15, 3))
for ax, image, label in zip(axes, digits.images, digits.target):
    ax.set_axis_off()
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title('Training: %i' % label)
```



In [3]:
```
# Choosing only data targets 1 and 7
data = digits.images[np.logical_or(digits.target == 1, digits.target ==7)]
y_target = digits.target[np.logical_or(digits.target == 1, digits.target ==7)]
```

In [4]:
```
# flattening the images
# n_samples = len(digits.images)
n_samples = len(data)
data = data.reshape((n_samples, -1)) # reshaping to focus on the number of pi

# Creating the classifier
n_neighbors = 5 # arbitrary
clf = neighbors.KNeighborsClassifier(n_neighbors, weights = 'uniform')

# Split data into 50% train and 50% test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, y_target, test_size=0.5, shuffle=False)

# Fit and predict
clf.fit(X_train, y_train)

predicted = clf.predict(X_test)
```

In [5]:
```
# Show predicted samples
_, axes = plt.subplots(nrows=1, ncols=5, figsize=(15, 3))
for ax, image, prediction in zip(axes, X_test, predicted):
    ax.set_axis_off()
    image = image.reshape(8, 8)
    ax.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest')
    ax.set_title(f'Prediction: {prediction}')
```

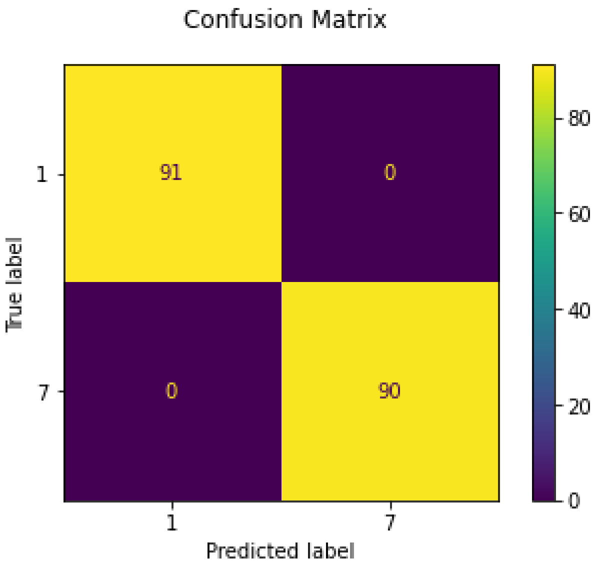


In [6]:
```
print(f"Classification report for classifier {clf}:\n"
      f"{metrics.classification_report(y_test, predicted)}\n")
```

```
Classification report for classifier KNeighborsClassifier():
              precision    recall  f1-score   support

           1       1.00      1.00      1.00        91
           7       1.00      1.00      1.00        90

    accuracy                           1.00       181
   macro avg       1.00      1.00      1.00       181
weighted avg       1.00      1.00      1.00       181
```

In [9]:
```
# Plot confusion matrix
disp = metrics.plot_confusion_matrix(clf, X_test, y_test)
disp.figure_.suptitle("Confusion Matrix")
print(f"Confusion matrix:\n{disp.confusion_matrix}")

plt.show()

print("Accuracy:", metrics.accuracy_score(y_test, predicted))
```

```
Confusion matrix:
[[91  0]
 [ 0 90]]
```



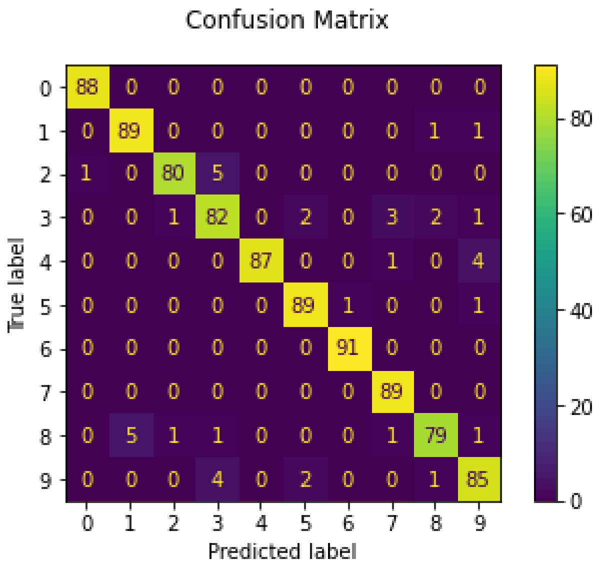

```
Accuracy: 1.0
```

Optional: Testing on the entire data set

In [10]:
```
digits = datasets.load_digits()

data = digits.images

n_samples = len(data)
data = data.reshape((n_samples, -1)) # reshaping to focus on the number of pi
```

```
# Creating the classifier
n_neighbors = 5 # arbitrary
clf = neighbors.KNeighborsClassifier(n_neighbors, weights = 'uniform')

# Split data into 50% train and 50% test subsets
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.5, shuffle=False)

# Fit and predict
clf.fit(X_train, y_train)

predicted = clf.predict(X_test)

print(f"Classification report for classifier {clf}:\n"
      f"{metrics.classification_report(y_test, predicted)}\n")

# Create confusion matrix

disp = metrics.plot_confusion_matrix(clf, X_test, y_test)
disp.figure_.suptitle("Confusion Matrix")
print(f"Confusion matrix:\n{disp.confusion_matrix}")

plt.show()

print("Accuracy:", metrics.accuracy_score(y_test, predicted))
```

```
Classification report for classifier KNeighborsClassifier():
              precision    recall  f1-score   support

           0       0.99      1.00      0.99        88
           1       0.95      0.98      0.96        91
           2       0.98      0.93      0.95        86
           3       0.89      0.90      0.90        91
           4       1.00      0.95      0.97        92
           5       0.96      0.98      0.97        91
           6       0.99      1.00      0.99        91
           7       0.95      1.00      0.97        89
           8       0.95      0.90      0.92        88
           9       0.91      0.92      0.92        92

    accuracy                           0.96       899
   macro avg       0.96      0.96      0.96       899
weighted avg       0.96      0.96      0.96       899


Confusion matrix:
[[88  0  0  0  0  0  0  0  0  0]
 [ 0 89  0  0  0  0  0  0  1  1]
 [ 1  0 80  5  0  0  0  0  0  0]
 [ 0  0  1 82  0  2  0  3  2  1]
 [ 0  0  0  0 87  0  0  1  0  4]
 [ 0  0  0  0  0 89  1  0  0  1]
 [ 0  0  0  0  0  0 91  0  0  0]
 [ 0  0  0  0  0  0  0 89  0  0]
 [ 0  5  1  1  0  0  0  1 79  1]
 [ 0  0  0  4  0  2  0  0  1 85]]
```




```
Accuracy: 0.9555061179087876
```

In [ ]: