Getting Datasets

Using the Scikit-learn Dataset

In [1]:
```
from sklearn import datasets
iris = datasets.load_iris()   # raw data of type Bunch
```

In [2]:
```
print(iris.DESCR)
```

```
Iris Plants Database
====================

Notes
-----
Data Set Characteristics:
    :Number of Instances: 150 (50 in each of three classes)
    :Number of Attributes: 4 numeric, predictive attributes and the class
    :Attribute Information:
        - sepal length in cm
        - sepal width in cm
        - petal length in cm
        - petal width in cm
        - class:
                - Iris-Setosa
                - Iris-Versicolour
                - Iris-Virginica
    :Summary Statistics:

    ============== ==== ==== ======= ===== ====================
                    Min  Max   Mean    SD   Class Correlation
    ============== ==== ==== ======= ===== ====================
    sepal length:   4.3  7.9   5.84   0.83    0.7826
    sepal width:    2.0  4.4   3.05   0.43   -0.4194
    petal length:   1.0  6.9   3.76   1.76    0.9490  (high!)
    petal width:    0.1  2.5   1.20  0.76     0.9565  (high!)
    ============== ==== ==== ======= ===== ====================

    :Missing Attribute Values: None
    :Class Distribution: 33.3% for each of 3 classes.
    :Creator: R.A. Fisher
    :Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
    :Date: July, 1988

This is a copy of UCI ML iris datasets.
http://archive.ics.uci.edu/ml/datasets/Iris (http://archive.ics.uci.edu/m
l/datasets/Iris)

The famous Iris database, first used by Sir R.A Fisher

This is perhaps the best known database to be found in the
pattern recognition literature.  Fisher's paper is a classic in the field
and
is referenced frequently to this day.  (See Duda & Hart, for example.)  T
he
data set contains 3 classes of 50 instances each, where each class refers
to a
type of iris plant.  One class is linearly separable from the other 2; th
e
latter are NOT linearly separable from each other.

References
----------
   - Fisher,R.A. "The use of multiple measurements in taxonomic problems"
```

```
     Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions
 to
     Mathematical Statistics" (John Wiley, NY, 1950).
   - Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analy
sis.
     (Q327.D83) John Wiley & Sons.  ISBN 0-471-22361-1.  See page 218.
   - Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System
     Structure and Classification Rule for Recognition in Partially Expos
ed
     Environments".  IEEE Transactions on Pattern Analysis and Machine
     Intelligence, Vol. PAMI-2, No. 1, 67-71.
   - Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule".  IEEE Transa
ctions
     on Information Theory, May 1972, 431-433.
   - See also: 1988 MLC Proceedings, 54-64.  Cheeseman et al"s AUTOCLASS
 II
     conceptual clustering system finds 3 classes in the data.
   - Many, many more ...
```

In [3]:

```python
print(iris.data)              # Features
```

```
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
```

In [4]:

```python
print(iris.feature_names)     # Feature Names
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal wid
th (cm)']
```

In [5]:
```
print(iris.target)              # Labels
print(iris.target_names)        # Label names
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2
2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2
 2 2]
['setosa' 'versicolor' 'virginica']
```

In [6]:
```
import pandas as pd
df = pd.DataFrame(iris.data)    # convert features
                                # to dataframe in Pandas
print(df.head())
```

```
     0    1    2    3
0  5.1  3.5  1.4  0.2
1  4.9  3.0  1.4  0.2
2  4.7  3.2  1.3  0.2
3  4.6  3.1  1.5  0.2
4  5.0  3.6  1.4  0.2
```

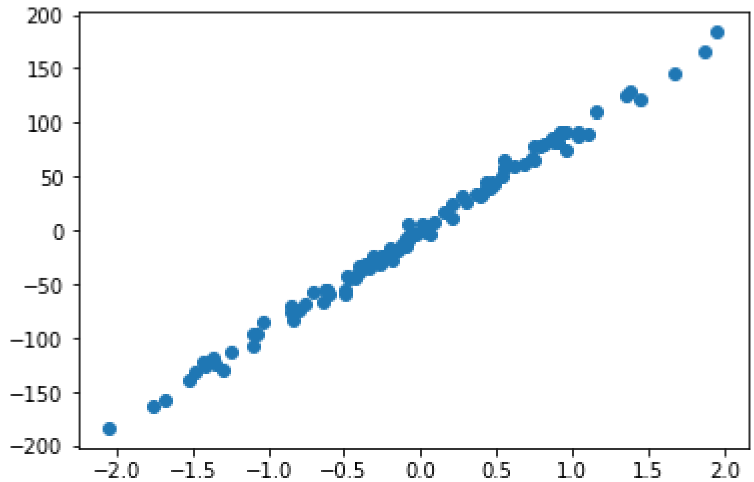
In [7]:
```
# data on breast cancer
breast_cancer = datasets.load_breast_cancer()

# data on diabetes
diabetes = datasets.load_diabetes()

# dataset of 1797 8x8 images of hand-written digits
digits = datasets.load_digits()
```

Generating Your Own Dataset

Linearly Distributed Dataset

In [8]:
```
%matplotlib inline
from matplotlib import pyplot as plt
from sklearn.datasets.samples_generator import make_regression

X, y = make_regression(n_samples=100, n_features=1, noise=5.4)
plt.scatter(X,y)
```

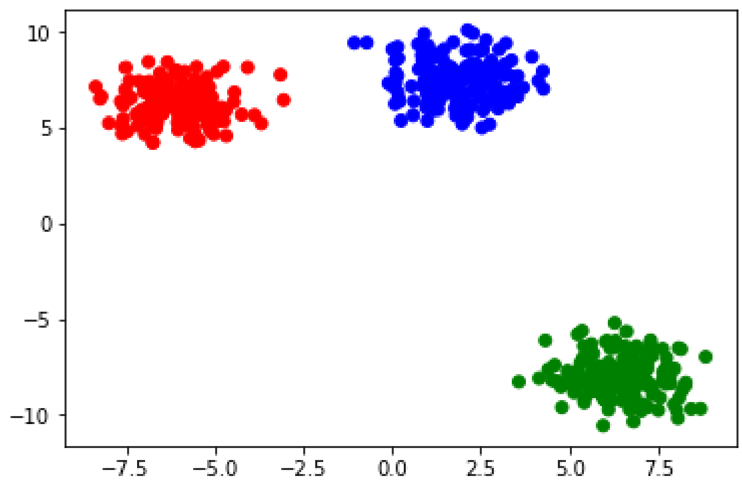
Out[8]: <matplotlib.collections.PathCollection at 0x1a233d4dd8>



Clustered Dataset

In [9]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_blobs

X, y = make_blobs(500, centers=3)  # Generate isotropic Gaussian
                                   # blobs for clustering

rgb = np.array(['r', 'g', 'b'])

# plot the blobs using a scatter plot and use color coding
plt.scatter(X[:, 0], X[:, 1], color=rgb[y])
```

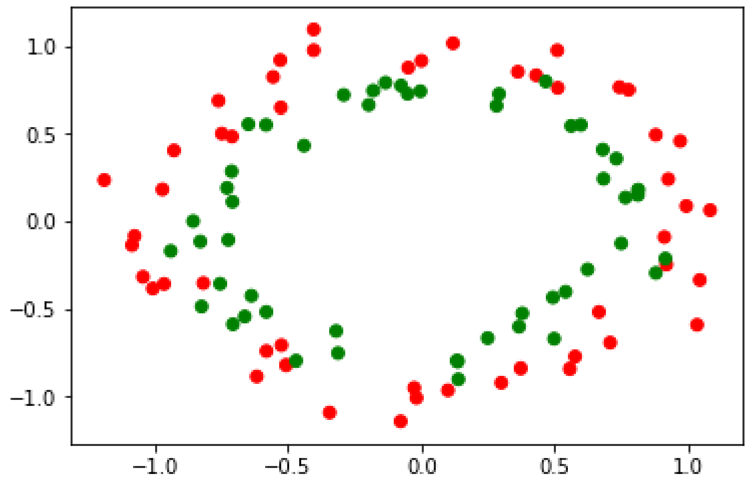
Out[9]:
```
<matplotlib.collections.PathCollection at 0x1a234e9f98>
```



Clustered Dataset Distributed in Circular Fashion

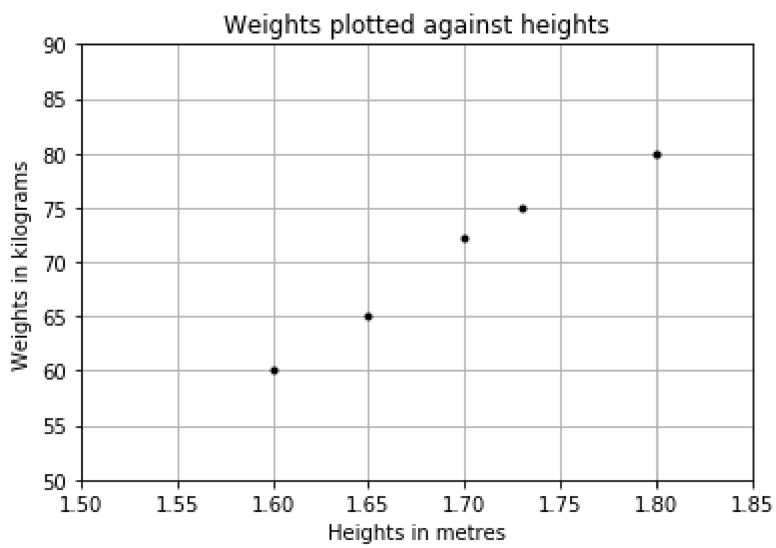
In [10]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
from sklearn.datasets import make_circles

X, y = make_circles(n_samples=100, noise=0.09)

rgb = np.array(['r', 'g', 'b'])
plt.scatter(X[:, 0], X[:, 1], color=rgb[y])
```

Out[10]:
```
<matplotlib.collections.PathCollection at 0x1a235b25c0>
```



Getting Started with Scikit-learn

In [11]:
```
%matplotlib inline
import matplotlib.pyplot as plt

# represents the heights of a group of people in metres
heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]

# represents the weights of a group of people in kgs
weights = [[60], [65], [72.3], [75], [80]]

plt.title('Weights plotted against heights')
plt.xlabel('Heights in metres')
plt.ylabel('Weights in kilograms')

plt.plot(heights, weights, 'k.')

# axis range for x and y
plt.axis([1.5, 1.85, 50, 90])
plt.grid(True)
```



Using the LinearRegression Class for Fitting the Model

In [12]:
```
from sklearn.linear_model import LinearRegression

# Create and fit the model
model = LinearRegression()
model.fit(X=heights, y=weights)
```

Out[12]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fal
se)
```

Making Predictions

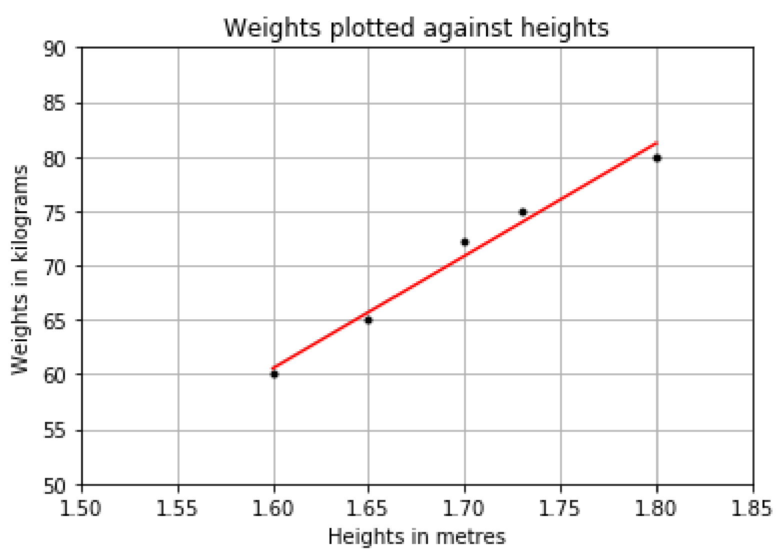
In [13]:
```
# make prediction
weight = model.predict([[1.75]])[0][0]
print(round(weight,2))          # 76.04
```

```
76.04
```

Plotting the Linear Regression Line

In [14]:
```
import matplotlib.pyplot as plt

heights = [[1.6], [1.65], [1.7], [1.73], [1.8]]
weights = [[60], [65], [72.3], [75], [80]]

plt.title('Weights plotted against heights')
plt.xlabel('Heights in metres')
plt.ylabel('Weights in kilograms')

plt.plot(heights, weights, 'k.')

plt.axis([1.5, 1.85, 50, 90])
plt.grid(True)

# plot the regression line
plt.plot(heights, model.predict(heights), color='r')
```

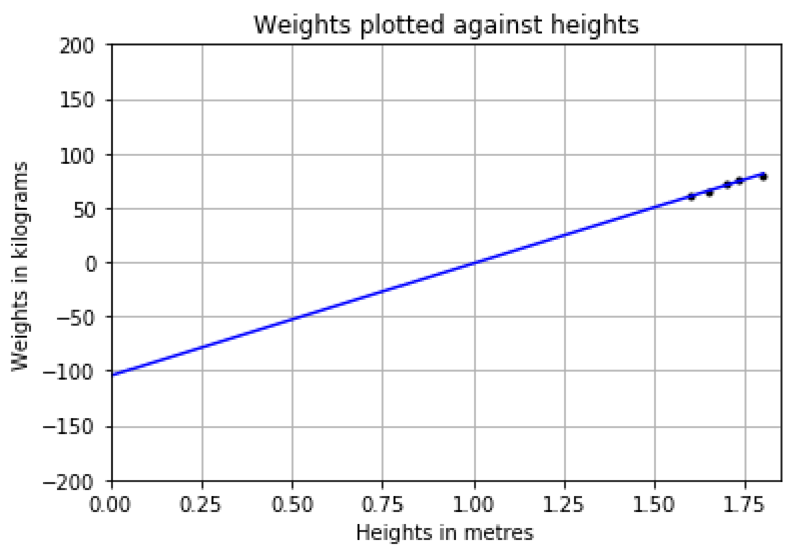
Out[14]:
```
[<matplotlib.lines.Line2D at 0x1a23b640b8>]
```



Getting the Gradient and Intercept of the Linear Regression Line

In [15]:
```
plt.title('Weights plotted against heights')
plt.xlabel('Heights in metres')
plt.ylabel('Weights in kilograms')

plt.plot(heights, weights, 'k.')

plt.axis([0, 1.85, -200, 200])
plt.grid(True)

# plot the regression line
extreme_heights = [[0], [1.8]]
plt.plot(extreme_heights, model.predict(extreme_heights), color='b')
```

Out[15]:
```
[<matplotlib.lines.Line2D at 0x1a23c43f98>]
```



In [16]:
```
round(model.predict([[0]])[0][0],2)    # -104.75
```

Out[16]:
```
-104.75
```

In [17]:
```
print(round(model.intercept_[0],2))    # -104.75
```

```
-104.75
```

In [18]:
```
print(round(model.coef_[0][0],2))      # 103.31
```

```
103.31
```

Examining the Performance of the Model by Calculating the Residual Sum of Squares

In [19]:
```
import numpy as np

print('Residual sum of squares: %.2f' %
       np.sum((weights - model.predict(heights)) ** 2))
```

```
Residual sum of squares: 5.34
```

The RSS should be as small as possible, with 0 indicating that the regression line fits the points exactly (rarely achievable in the real world).

Evaluating the Model Using a Test Dataset

In [20]:
```
# test data
heights_test = [[1.58], [1.62], [1.69], [1.76], [1.82]]
weights_test = [[58], [63], [72], [73], [85]]
```

In [21]:
```
# Total Sum of Squares (TSS)
weights_test_mean = np.mean(np.ravel(weights_test))
TSS = np.sum((np.ravel(weights_test) -
              weights_test_mean) ** 2)
print("TSS: %.2f" % TSS)

# Residual Sum of Squares (RSS)
RSS = np.sum((np.ravel(weights_test) -
              np.ravel(model.predict(heights_test)))
                 ** 2)
print("RSS: %.2f" % RSS)

# R_squared
R_squared = 1 - (RSS / TSS)
print("R-squared: %.2f" % R_squared)
```

```
TSS: 430.80
RSS: 24.62
R-squared: 0.94
```

In [22]:
```
# using scikit-learn to calculate r-squared
print('R-squared: %.4f' % model.score(heights_test,
                                      weights_test))

# R-squared: 0.9429
```

```
R-squared: 0.9429
```

An R-Squared value of 0.9429 (94.29%) indicates a pretty good fit for your test data.

Persisting the Model

In [23]:
```
import pickle

# save the model to disk
filename = 'HeightsAndWeights_model.sav'
# write to the file using write and binary mode
pickle.dump(model, open(filename, 'wb'))
```

In [24]:
```
# load the model from disk
loaded_model = pickle.load(open(filename, 'rb'))
```

In [25]:
```
result = loaded_model.score(heights_test,
                            weights_test)
```

Using the joblib module is very similar to using the pickle module

In [26]:
```
from sklearn.externals import joblib

# save the model to disk
filename = 'HeightsAndWeights_model2.sav'
joblib.dump(model, filename)

# load the model from disk
loaded_model = joblib.load(filename)
result = loaded_model.score(heights_test,
                            weights_test)
print(result)
```

```
0.9428592885995253
```

Data Cleansing

Cleaning Rows with NaNs

In [27]:
```
import pandas as pd
df = pd.read_csv('NaNDataset.csv')
df.isnull().sum()
```

Out[27]:
```
A    0
B    2
C    0
dtype: int64
```

In [28]:
```
print(df)
```

```
    A     B   C
0   1   2.0   3
1   4   NaN   6
2   7   NaN   9
3  10  11.0  12
4  13  14.0  15
5  16  17.0  18
```

Replacing NaN with the Mean of the Column

In [29]:
```
# replace all the NaNs in column B with the average of column B
df.B = df.B.fillna(df.B.mean())
print(df)
```

```
    A     B   C
0   1   2.0   3
1   4  11.0   6
2   7  11.0   9
3  10  11.0  12
4  13  14.0  15
5  16  17.0  18
```

Removing Rows

In [30]:
```
df = pd.read_csv('NaNDataset.csv')
df = df.dropna()                              # drop all rows with NaN
print(df)
```

```
    A     B   C
0   1   2.0   3
3  10  11.0  12
4  13  14.0  15
5  16  17.0  18
```

In [31]:
```
df = df.reset_index(drop=True)                # reset the index
print(df)
```

```
    A     B   C
0   1   2.0   3
1  10  11.0  12
2  13  14.0  15
3  16  17.0  18
```

Removing Duplicate Rows

In [32]:
```
import pandas as pd
df = pd.read_csv('DuplicateRows.csv')
print(df.duplicated(keep=False))
```

```
0    False
1     True
2     True
3    False
4    False
5     True
6     True
7    False
8    False
dtype: bool
```

In [33]:
```
print(df.duplicated(keep="first"))
```

```
0    False
1    False
2     True
3    False
4    False
5    False
6     True
7    False
8    False
dtype: bool
```

In [34]:
```
print(df[df.duplicated(keep=False)])
```

```
    A   B   C
1   4   5   6
2   4   5   6
5  10  11  12
6  10  11  12
```

In [35]:
```
df.drop_duplicates(keep='first', inplace=True)  # remove duplicates and kee
print(df)
```

```
    A   B   C
0   1   2   3
1   4   5   6
3   7   8   9
4   7  18   9
5  10  11  12
7  13  14  15
8  16  17  18
```

In [36]:
```
df.drop_duplicates(subset=['A', 'C'], keep='last',
                            inplace=True)     # remove all duplicates in
                                              # columns A and C and keep
                                              # the last
print(df)
```

```
    A   B   C
0   1   2   3
1   4   5   6
4   7  18   9
5  10  11  12
7  13  14  15
8  16  17  18
```

Normalizing Columns

In [37]:
```
import pandas as pd
from sklearn import preprocessing

df = pd.read_csv('NormalizeColumns.csv')
print(df)

x = df.values.astype(float)

min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled, columns=df.columns)
print(df)
```

```
      A   B   C
0  1000   2   3
1   400   5   6
2   700   6   9
3   100  11  12
4  1300  14  15
5  1600  17  18
     A         B    C
0  0.6  0.000000  0.0
1  0.2  0.200000  0.2
2  0.4  0.266667  0.4
3  0.0  0.600000  0.6
4  0.8  0.800000  0.8
5  1.0  1.000000  1.0
```

Removing Outliers

Tukey Fences

In [38]:
```
import numpy as np

def outliers_iqr(data):
    q1, q3 = np.percentile(data, [25, 75])
    iqr = q3 - q1
    lower_bound = q1 - (iqr * 1.5)
    upper_bound = q3 + (iqr * 1.5)
    return np.where((data > upper_bound) | (data < lower_bound))
```

In [39]:
```
import pandas as pd
df = pd.read_csv("http://www.mosaic-web.org/go/datasets/galton.csv")
print(df.head())
```

```
  family  father  mother sex  height  nkids
0      1    78.5    67.0   M    73.2      4
1      1    78.5    67.0   F    69.2      4
2      1    78.5    67.0   F    69.0      4
3      1    78.5    67.0   F    69.0      4
4      2    75.5    66.5   M    73.5      4
```

In [40]:
```
print("Outliers using outliers_iqr()")
print("=============================")
for i in outliers_iqr(df.height)[0]:
    print(df[i:i+1])
```

```
Outliers using outliers_iqr()
=============================
    family  father  mother sex  height  nkids
288     72    70.0    65.0   M    79.0      7
```

Z-Score

In [41]:
```
def outliers_z_score(data):
    threshold = 3
    mean = np.mean(data)
    std = np.std(data)
    z_scores = [(y - mean) / std for y in data]
    return np.where(np.abs(z_scores) > threshold)
```

In [42]:
```
print("Outliers using outliers_z_score()")
print("=================================")
for i in outliers_z_score(df.height)[0]:
    print(df[i:i+1])
print()
```

```
Outliers using outliers_z_score()
=================================
     family  father  mother sex  height  nkids
125      35    71.0    69.0   M    78.0      5
     family  father  mother sex  height  nkids
288      72    70.0    65.0   M    79.0      7
     family  father  mother sex  height  nkids
672     155    68.0    60.0   F    56.0      7
```

In [ ]: